Analyze this!

Analytical Power in SQL

Hermann Bär
Director Product Management

hermann.baer@oracle.com

SQL Evolution

1998
- Introduction of Window functions

2001
- Enhanced Window functions (percentile,etc)
- Rollup, grouping sets, cube

2004
- Statistical functions
- Sql model clause
- Partition Outer Join
- Data mining I

2007
- Data mining II
- SQL Pivot
- Recursive WITH
- ListAgg, N_Th value window

2012
- Pattern matching
- Top N clause
- Lateral Views,APPLY
- Identity Columns
- Column Defaults
- Data Mining III

1998 → 2001 → 2002 → 2004 → 2005 → 2007 → 2009 → 2012

SQL Evolution

Flow: Linear

ORACLE 8i INTERNET

- Introduction of Window functions

DATABASE 9i CLUSTER

- Enhanced Window functions (percentile,etc)
- Rollup, grouping sets, cube

ORACLE 10g DATABASE

- Statistical functions
- SQL model clause
- Partition Outer Join
- Data mining I

ORACLE DATABASE 11g

- Data mining II
- SQL Pivot
- Recursive WITH
- ListAgg, Nth value window

ORACLE DATABASE 12c

- **Pattern matching**
- Top N clause
- Identity Columns
- Column Defaults
- Data Mining III

ORACLE

SQL Pattern Matching

"What's this about?"

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)

ORACLE

Pattern Matching in Sequences of Rows

Objective

Provide native SQL language construct

Align with well-known regular expression declaration (PERL)

Apply expressions across rows

Soon to be in ANSI SQL Standard

"Find one or more event A followed by one B followed by one or more C in a 1 minute interval"

EVENT	TIME	LOCATION
A	1	SFO
A	1	SFO
A	2	ATL
A	2	LAX
B	2	SFO
C	2	LAX
C	3	LAS
A	3	SFO
B	3	NYC
C	4	NYC

> 1 min

A+ B C (perl)

A	2	ATL
A	2	LAX
B	2	SFO
C	2	LAX

Pattern Recognition In Sequences of Rows

"SQL Pattern Matching" - Concept

- Recognize patterns in sequences of events using SQL
  - Sequence is a stream of rows
  - Event equals a row in a stream
- New SQL construct MATCH_RECOGNIZE
  - Logically partition and order the data
    - ORDER BY mandatory (optional PARTITION BY)
  - Pattern defined using regular expression using variables
  - Regular expression is matched against a sequence of rows
  - Each pattern variable is defined using conditions on rows and aggregates

ORACLE

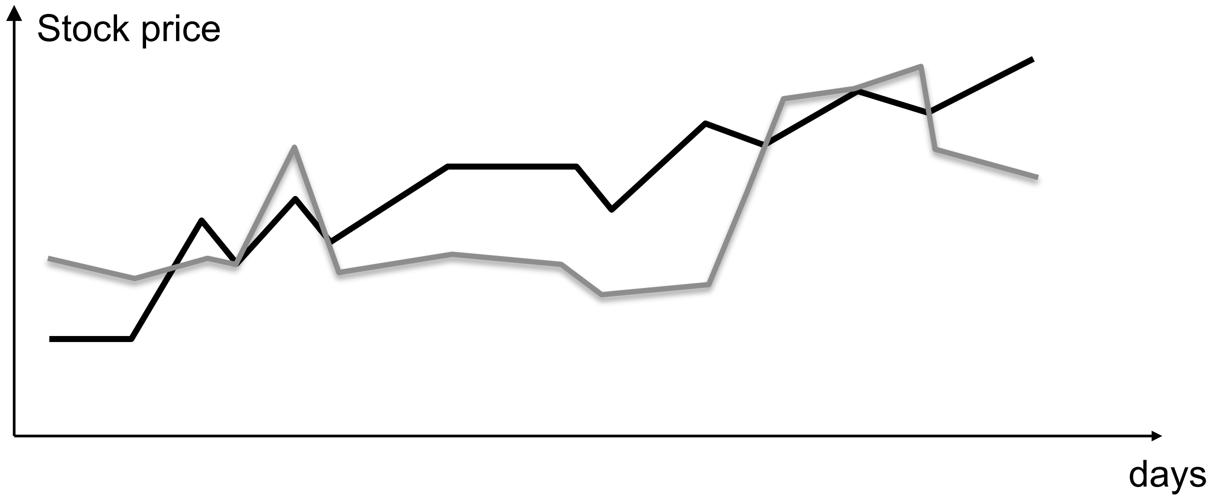
 |

SQL Pattern Matching in action

Example: Find A Double Bottom Pattern (W-shape) in ticker stream

Find a W-shape pattern in a ticker stream:

- Output the **beginning** and **ending** date of the pattern
- Calculate **average price** in the second ascent
- Find only patterns that **lasted less than a week**

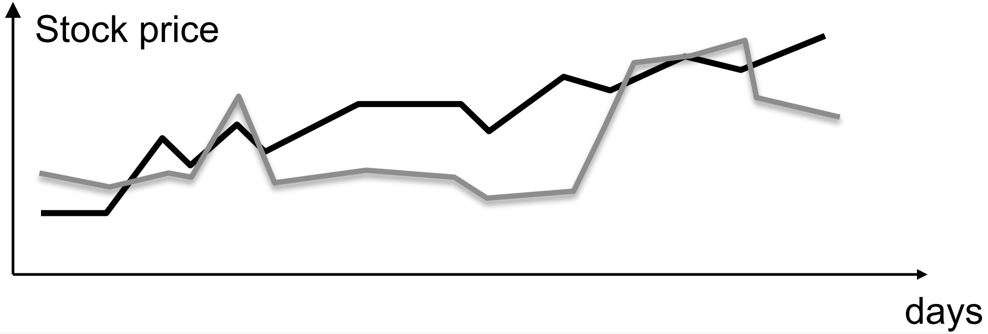


SQL Pattern Matching in action

Example: Find W-Shape

New syntax for discovering patterns using SQL:

MATCH_RECOGNIZE ( )




```
SELECT . . .
FROM ticker MATCH_RECOGNIZE (
      . . .
)
```

SQL Pattern Matching in action

Example: Find W-Shape

Find a W-shape pattern in a ticker stream:

- Set the PARTITION BY and ORDER BY clauses

We will continue to look at the black stock only from now on


```
SELECT …
FROM ticker MATCH_RECOGNIZE (
       PARTITION BY name ORDER BY time
```

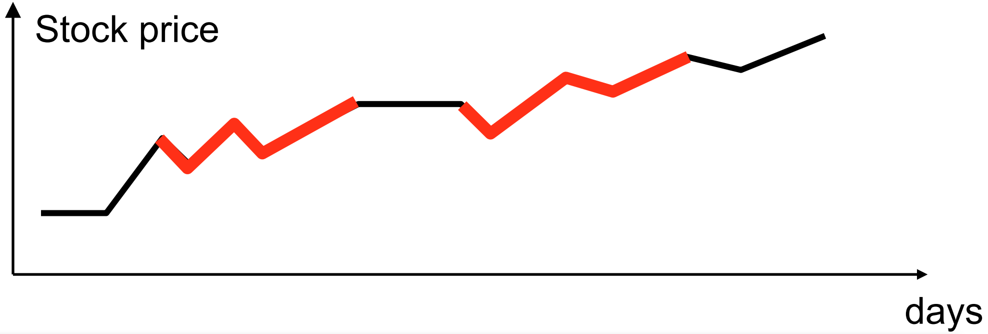
 |

SQL Pattern Matching in action

Example: Find W-Shape

Find a W-shape pattern in a ticker stream:

- Define the **pattern** – the "W-shape"



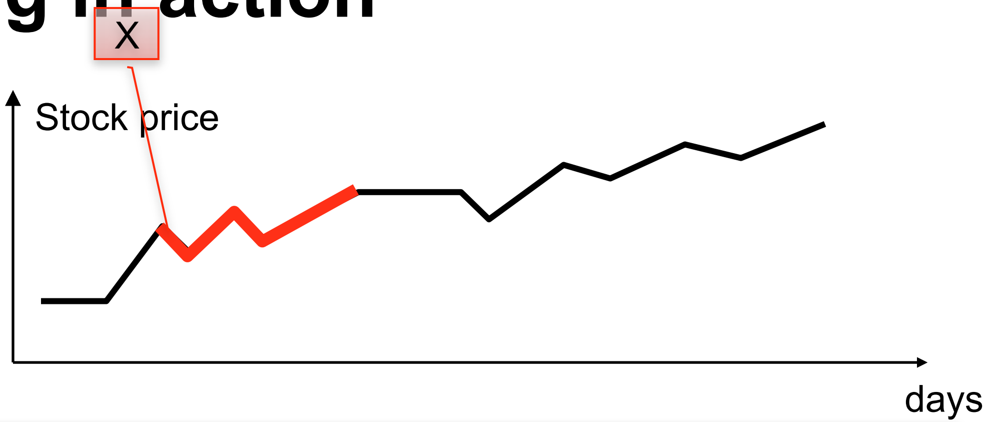

```
SELECT …
FROM ticker MATCH_RECOGNIZE (
       PARTITION BY name ORDER BY time



       PATTERN (X+ Y+ W+ Z+)
```


SQL Pattern Matching in action

Example: Find W-Shape

Find a W-shape pattern in a ticker stream:

- Define the **pattern** – the first down part of the "W-shape"



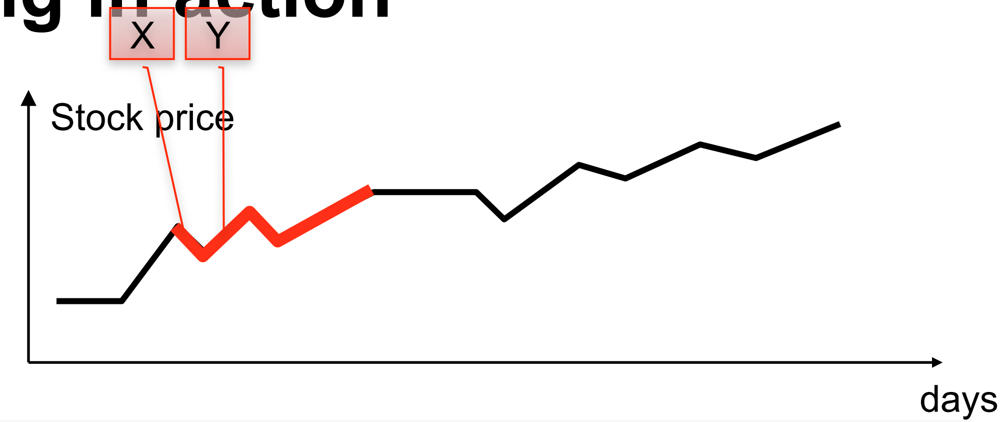

```
SELECT …
FROM ticker MATCH_RECOGNIZE (
       PARTITION BY name ORDER BY time



       PATTERN (X+ Y+ W+ Z+)
       DEFINE X AS (price < PREV(price)),
```


SQL Pattern Matching in action

Example: Find W-Shape

Find a W-shape pattern in a ticker stream:

- Define the **pattern** – the first up part of “W-shape”



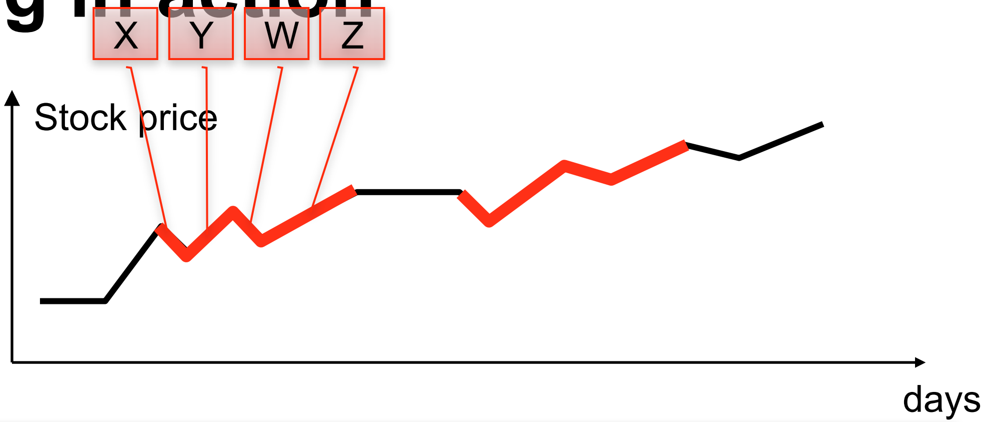

```
SELECT …
FROM ticker MATCH_RECOGNIZE (
       PARTITION BY name ORDER BY time



       PATTERN (X+ Y+ W+ Z+)
       DEFINE X AS (price < PREV(price)),
              Y AS (price > PREV(price)),
```


SQL Pattern Matching in action

Example: Find W-Shape

Find a W-shape pattern in a ticker stream:

- Define the **pattern** – the second down (w) and the second up(z) of the “W-shape”



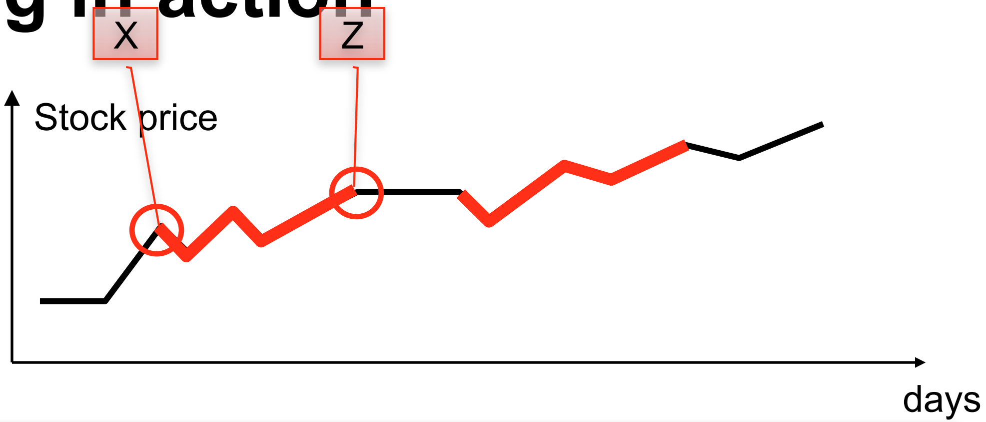

```
SELECT …
FROM ticker MATCH_RECOGNIZE (
       PARTITION BY name ORDER BY time



       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)))
```

ORACLE

 | |

SQL Pattern Matching in action

Example: Find W-Shape



Find a W-shape pattern in a ticker stream:

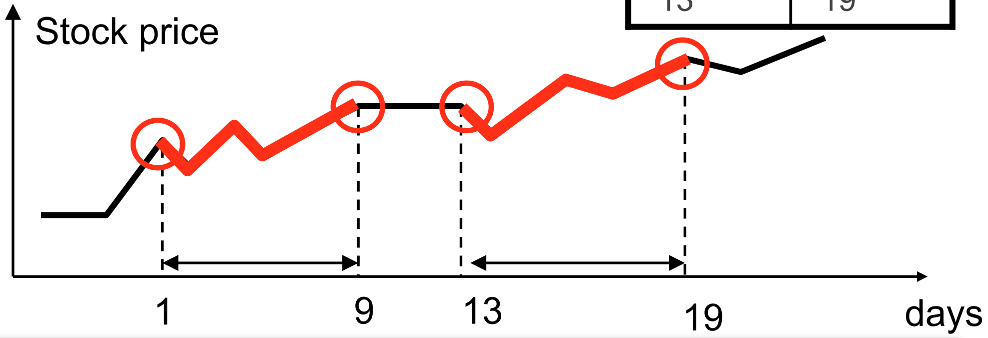
- Define the measures to output once a pattern is matched:
  - **FIRST: beginning date**
  - **LAST: ending date**

```
SELECT …
FROM ticker MATCH_RECOGNIZE (
       PARTITION BY name ORDER BY time
       MEASURES FIRST(x.time) AS first_x,
                LAST(z.time)  AS last_z

       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))
```


SQL Pattern Matching in action

Example: Find W-Shape

First_x	Last_z
1	9
13	19



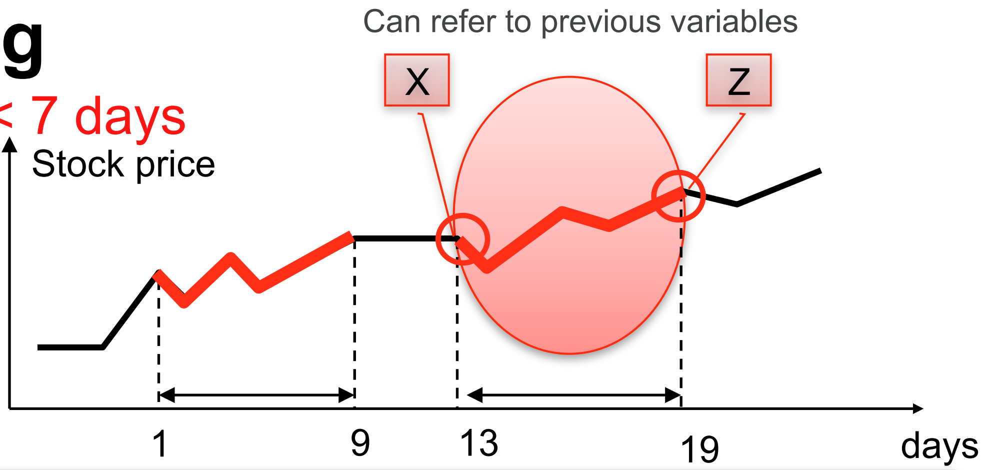
Find a W-shape pattern in a ticker stream:

- Output **one row** each time we find a match to our pattern

```
SELECT first_x, last_z
FROM ticker MATCH_RECOGNIZE (
       PARTITION BY name ORDER BY time
       MEASURES FIRST(x.time) AS first_x,
                LAST(z.time)  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)))
```

SQL Pattern Matching

Example: Find W-Shape lasts < 7 days



Find a W-shape pattern in a ticker stream:

- Extend the pattern to find W-shapes that **lasted less than a week**

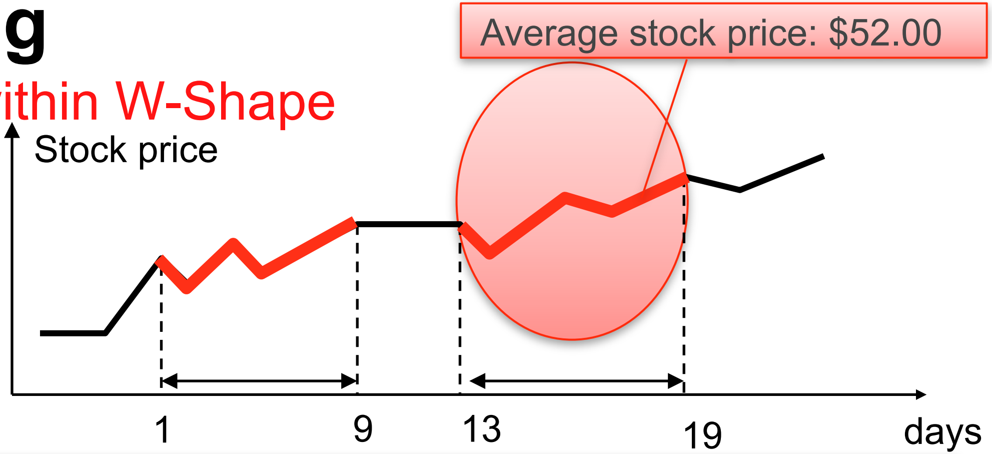
```
SELECT first_x, last_z
FROM ticker MATCH_RECOGNIZE (
       PARTITION BY name ORDER BY time
       MEASURES FIRST(x.time) AS first_x,
                LAST(z.time)  AS last_z
       ONE ROW PER MATCH
       PATTERN (X+ Y+ W+ Z+)
       DEFINE X AS (price < PREV(price)),
              Y AS (price > PREV(price)),
              W AS (price < PREV(price)),
              Z AS (price > PREV(price)  AND
                    z.time - FIRST(x.time) <= 7 ))
```

SQL Pattern Matching

Example: Find average price within W-Shape

Find a W-shape pattern in a ticker stream:

- Calculate **average price** in the second ascent




```
SELECT first_x, last_z
FROM ticker MATCH_RECOGNIZE (
        PARTITION BY name ORDER BY time
        MEASURES FIRST(x.time) AS first_x,
                 LAST(z.time)  AS last_z,
                 AVG(z.price)  AS avg_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.time - FIRST(x.time) <= 7 ))))
```


SQL Pattern Matching

Recap of MATCH_RECOGNIZE Syntax

```
<table_expression> := <table_expression> MATCH_RECOGNIZE
         ( [ PARTITION BY <cols> ]
           [ ORDER BY <cols> ]
           [ MEASURES <cols> ]
           [ ONE ROW PER MATCH | ALL ROWS PER MATCH ]
           [ SKIP_TO_option ]
           PATTERN ( <row pattern> )
           [ SUBSET <subset list> ]
           DEFINE <definition list>
         )
```


SQL Pattern Matching

"Declarative" Pattern Matching

- Matching within a stream of events (ordered partition of data)
  - `MATCH_RECOGNIZE (PARTITION BY stock_name ORDER BY time MEASURES …`
- Use framework of Perl regular expressions
  - Terms are conditions on rows
  - `PATTERN (X+ Y+ W+ Z+)`
- Define matching using Boolean conditions on rows
  - ```
    DEFINE
         X AS (price > 15)
    …
    ```

ORACLE
    ```

SQL Pattern Matching

“Declarative” Pattern Matching, cont.

- Name and refer to previous variables (i.e., rows) in conditions
  - ```
    DEFINE X AS (price < PREV(price,1)),
    Y AS (price > PREV(price,1)),
    W AS (price < PREV(price,1)),
    Z AS (price > PREV(price,1) AND Z.price > X.price)
    ```
- New aggregates: FIRST, LAST
  - ```
    DEFINE X AS (price < PREV(price)),
    Y AS (price > PREV(price)),
    W AS (price < PREV(price)),
    Z AS (price > PREV(price) AND Z.time < FIRST(X.time)+10)
    ```

ORACLE

 |

SQL Pattern Matching

"Declarative" Pattern Matching, cont.

- Running aggregates in conditions on currently defined variables:

```
– DEFINE X AS (price < PREV(price) AND AVG(num_shares) < 10 ),
         Y AS (price > PREV(price)  AND count(Y.price) < 10 ),
         W AS (price < PREV(price)),
         Z AS (price > PREV(price)  AND Z.price > Y.price )
```

- Final aggregates in conditions but only on previously defined variables

```
– DEFINE X AS (price < PREV(price)),
         Y AS (price > PREV(price)),
         W AS (price < PREV(price) AND count(Y.price) > 10 ) ,
         Z AS (price > PREV(price) AND Z.price > LAST(Y.price) )
```


SQL Pattern Matching

“Declarative” Pattern Matching, cont.

- After match SKIP option :
  - `SKIP PAST LAST ROW`
  - `SKIP TO NEXT ROW`
  - `SKIP TO <VARIABLE>`
  - `SKIP TO FIRST(<VARIABLE>)`
  - `SKIP TO LAST (<VARIABLE>)`
- What rows to return
  - `ONE ROW PER MATCH`
  - `ALL ROWS PER MATCH`
  - `ALL ROWS PER MATCH WITH UNMATCHED ROWS`

ORACLE

SQL Pattern Matching

Building Regular Expressions

- Concatenation: no operator
- Quantifiers:
  - `*` 0 or more matches
  - `+` 1 or more matches
  - `?` 0 or 1 match
  - `{n}` exactly n matches
  - `{n,}` n or more matches
  - `{n, m}` between n and m (inclusive) matches
  - `{, m}` between 0 an m (inclusive) matches
  - Reluctant quantifier – an additional ?

SQL Pattern Matching

Building Regular Expressions

- Alternation: |
  - A | B
- Grouping: ()
  - (A | B)+
- Permutation: Permute() – alternate all permutations
  - PERMUTE (A B C) -> A B C | A C B | B A C | B C A | C A B | C B A
- ^: indicates beginning of partition
- $: indicates end of partition

SQL Pattern Matching

Preferment Rules – Follow Perl

- Greedy quantifiers: longer match preferred
- Reluctant quantifiers: shorter match preferred
- Alternation: left to right
- Make local choices
  - Example: for pattern (A | B)*, AAA preferred over BBBBB

SQL Pattern Matching

"Declarative" Pattern Matching

- Can subset variable names
  - ```
    SELECT first_x, avg_xy
    FROM ticker
    MATCH_RECOGNIZE
    (PARTITION BY name ORDER BY time ONE ROW PER MATCH
    MEASURES FIRST(x.time)first_x, AVG(T.price) avg_xy
    PATTERN (X+ Y+ W+ Z+) SUBSET T = (X, Y)
    DEFINE X AS (price < PREV(price)),
           Y AS (price > PREV(price)),
           W AS (price < PREV(price)),
           Z AS (price > PREV(price) AND Z.price > T.price ) );
    ```

ORACLE
    ```

SQL Pattern Matching

“Declarative” Pattern Matching, cont.

- After match SKIP option :
  - `SKIP PAST LAST ROW`
  - `SKIP TO NEXT ROW`
  - `SKIP TO <VARIABLE>`
  - `SKIP TO FIRST(<VARIABLE>)`
  - `SKIP TO LAST (<VARIABLE>)`
- What rows to return
  - `ONE ROW PER MATCH`
  - `ALL ROWS PER MATCH`
  - `ALL ROWS PER MATCH WITH UNMATCHED ROWS`

SQL Pattern Matching

ALL ROWS PER MATCH OPTION

Detect and classify ALL events after privileges have been revoked for the user.

Generate a row for first improper login attempt (event)

```
SELECT name, rev_time, time, clas
FROM event_log
MATCH_RECOGNIZE (PARTITION BY name ORDER BY time
PATTERN (X Y* Z)
MEASURES x.time rev_time, classifier() clas
ALL ROWS PER MATCH
DEFINE X AS (event = 'revoke'),
       Y AS (event NOT IN ('login', 'grant')),
       Z AS (event = 'login' ) )
```

NAME	EVENT	TIME
John	grant	9:00 AM
John	revoke	1:00 PM
John	fired	1:20 PM
John	escorted	1:25 PM
John	left	1:30 PM
John	login	1:50 PM

NAME	REV_TIME	TIME	CLAS
John	1:00 PM	1:00 PM	X
John	1:00 PM	1:20 PM	Y
John	1:00 PM	1:25 PM	Y
John	1:00 PM	1:30 PM	y
John	1:00 PM	1:50 PM	Z

SQL Pattern Matching

ONE ROW PER MATCH OPTION

Detect ALL login events after privileges have been revoked for the user.

Generate a row for first improper login attempt (event) when more than one login attempt within a minute were happening

```
SELECT name, rev_time, first_log
FROM event_log
MATCH_RECOGNIZE (PARTITION BY name ORDER BY time
PATTERN (X Y* Z Z W+)
MEASURES FIRST(x.time) first_log ONE ROW PER MATCH
DEFINE X AS (event = 'revoke'),
       Y AS (event NOT IN ('login', 'grant')),
       Z AS (event = 'login'),
       W AS (event = 'login' AND
             W.time - FIRST(z.time) <= 60) )
```

NAME	EVENT	TIME
John	grant	9:00 AM
John	revoke	1:00 PM
John	fired	1:20 PM
John	left	1:25 PM
John	login	1:30 PM
John	login	1:31 PM
John	login	1:32 PM

NAME	REV_TIME	FIRST_LOG
John	1:00 PM	1:30 PM

ORACLE

SQL Pattern Matching

Real world use cases

SQL Pattern Matching

Example Sessionization for user log

- Define a session as a sequence of one or more events with the same partition key where the inter-timestamp gap is less than a specified threshold
- Example "user log analysis"
  - Partition key: User ID, Inter-timestamp gap: 10 (seconds)
  - Detect the sessions
  - Assign a within-partition (per user) surrogate Session_ID to each session
  - Annotate each input tuple with its Session_ID

ORACLE

SQL Pattern Matching

Example Sessionization for user log

TIME	USER ID
1	Mary
2	Sam
11	Mary
12	Sam
22	Sam
23	Mary
32	Sam
34	Mary
43	Sam
44	Mary
47	Sam
48	Sam
53	Mary
59	Sam
60	Sam
63	Mary
68	Sam

Identify sessions

TIME	USER ID
1	Mary
11	Mary
23	Mary
34	Mary
44	Mary
53	Mary
63	Mary
2	Sam
12	Sam
22	Sam
32	Sam
43	Sam
47	Sam
48	Sam
59	Sam
60	Sam
68	Sam

Number Sessions per user

TIME	USER ID	SESSION
1	Mary	1
11	Mary	1
23	Mary	2
34	Mary	3
44	Mary	3
53	Mary	3
63	Mary	3
2	Sam	1
12	Sam	1
22	Sam	1
32	Sam	1
43	Sam	2
47	Sam	2
48	Sam	2
59	Sam	3
60	Sam	3
68	Sam	3

SQL Pattern Matching

Example Sessionization for user log: ALL ROWS PER MATCH

```
SELECT time, user_id, session_id
FROM Events MATCH_RECOGNIZE
      (PARTITION BY user_ID ORDER BY time
       MEASURES match_number() as session_id
       ALL ROWS PER MATCH
       PATTERN (b s*)
       DEFINE
              s as (s.time - prev(s.time) <= 10)
       );
```


SQL Pattern Matching

Example Sessionization – Aggregation of sessionized data

- Primitive sessionization only a foundation for analysis
  - Mandatory to logically identify related events and group them
- Aggregation for the first data insight
  - How many “events” happened within an individual session?
  - What was the total duration of an individual session?

SQL Pattern Matching

Example Sessionization – Aggregation of sessionized data

TIME	USER ID	SESSION
1	Mary	1
11	Mary	1
23	Mary	2
34	Mary	3
44	Mary	3
53	Mary	3
63	Mary	3
2	Sam	1
12	Sam	1
22	Sam	1
32	Sam	1
43	Sam	2
47	Sam	2
48	Sam	2
59	Sam	3
60	Sam	3
68	Sam	3

TIME	SESSION_ID	START_TIME	NUM EVENTS	DURATION
Mary	1	1	2	10
Mary	2	23	1	0
Mary	3	34	4	29
Sam	1	2	4	30
Sam	2	43	3	5
Sam	3	59	3	9

SQL Pattern Matching

Example Sessionization – Aggregation: ONE ROW PER MATCH

```
SELECT user_id, session_id, start_time, no_of_events, duration
FROM Events MATCH_RECOGNIZE
       ( PARTITION BY user_ID ORDER BY time ONE ROW PER MATCH
         MEASURES match_number() session_id,
                  count(*) as no_of_events,
                  first(time) start_time,
                  last(time) - first(time) duration
         PATTERN (b s*)
         DEFINE
                  s as (s.time - prev(time) <= 10)
       )
ORDER BY user_id, session_id;
```

SQL Pattern Matching

Example Sessionization – using window functions

```
CREATE VIEW Sessionized_Events as
SELECT Time_Stamp, User_ID,
       Sum(Session_Increment) over  (partition by User_ID order by Time_Stampasc) Session_ID
FROM (SELECT Time_Stamp, User_ID,
             CASE WHEN (Time_Stamp – Lag(Time_Stamp) over (partition by User_ID order by Time_Stampasc)) < 10
                  THEN 0 ELSE 1 END Session_Increment
      FROM Events);
```

```
SELECT User_ID,
           Min(Time_Stamp) Start_Time,
           Count(*) No_Of_Events,
           (Max(Time_Stamp) -Min(Time_Stamp)) Duration
FROM Sessionized_Events
GROUP BY User_ID, Session_ID
ORDER BY User_ID, Start_Time;
```


SQL Pattern Matching

Example Call Detail Records Analysis

- Scenario:
  - The same call can be interrupted (or dropped).
  - Caller will call callee within a few seconds of interruption. Still a session
  - Need to know how often we have interrupted calls & effective call duration
- The to-be-sessionized phenomena are characterized by
  - Start_Time, End_Time
  - Caller_ID, Callee_ID

SQL Pattern Matching

Example Call Detail Records Analysis using SQL Pattern Matching

```
SELECT  Caller, Callee, Start_Time,  Effective_Call_Duration,
        (End_Time - Start_Time) -  Effective_Call_Duration
                    AS  Total_Interruption_Duration,
         No_Of_Restarts,  Session_ID
FROM call_details MATCH_RECOGNIZE
        ( PARTITION BY Caller, Callee ORDER BY Start_Time
          MEASURES
             A.Start_Time AS Start_Time,
             B.End_Time AS End_Time,
             SUM(B.End_Time – A.Start_Time) as Effective_Call_Duration,
             COUNT(B.*) as No_Of_Restarts,
             MATCH_NUMBER() as Session_ID
          PATTERN (A B*)
          DEFINE B as B.Start_Time - prev(B.end_Time) < 60) ;
```


SQL Pattern Matching

Example Call Detail Records Analysis prior to Oracle Database 12c

```
With Sessionized_Call_Details as
(select Caller, Callee, Start_Time, End_Time,
      Sum(case when Inter_Call_Intrvl < 60 then 0 else 1 end)
      over(partition by Caller, Callee order by Start_Time) Session_ID
 from (select Caller, Callee, Start_Time, End_Time,
        (Start_Time - Lag(End_Time) over(partition by Caller, Callee order by Start_Time)) Inter_Call_Intrvl
       from Call_Details)),
Inter_Subcall_Intrvls as
(select Caller, Callee, Start_Time, End_Time,
      Start_Time - Lag(End_Time) over(partition by Caller, Callee, Session_ID order by Start_Time)
      Inter_Subcall_Intrvl,
      Session_ID
    from Sessionized_Call_Details)
Select Caller, Callee,
       Min(Start_Time)   Start_Time, Sum(End_Time - Start_Time)  Effective_Call_Duration,
       Nvl(Sum(Inter_Subcall_Intrvl), 0)  Total_Interuption_Duration, (Count(*) - 1) No_Of_Restarts,
       Session_ID
from Inter_Subcall_Intrvls
group by Caller, Callee, Session_ID;
```

SQL Pattern Matching

Example Suspicious Money Transfers

- Detect suspicious money transfer pattern for an account
  - Three or more small amount (<2K) money transfers within 30 days
  - Subsequent large transfer (>=1M) within 10 days of last small transfer.
- Report account, date of first small transfer, date of last large transfer

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

Three small transfers within 30 days

Large transfer within 10 days of last small transfer

SQL Pattern Matching

Example Suspicious Money Transfers

```
SELECT userid, first_t, last_t, amount
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE
(   PARTITION BY userid ORDER BY time
    MEASURES  FIRST(x.time) first_t, y.time last_t, y.amount amount
    PATTERN ( x{3,} Y )
    DEFINE X as (event='transfer' AND amount < 2000),
           Y as (event='transfer' AND amount >= 1000000 AND
                 last(X.time) - first(X.time) < 30  AND
                 Y.time - last(X.time) < 10 ))
```

Three or more transfers of small amount

Within 30 days of each other

Followed by a large transfer

Within 10 days of last small

SQL Pattern Matching

Example Suspicious Money Transfers - Refined

- Detect suspicious money transfer pattern between accounts
  - Three or more small amount (<2K) money transfers within 30 days
    - Transfers to different accounts (total sum of small transfers (20K))
  - Subsequent large transfer (>=1M) within 10 days of last small transfer.
- Report account, date of first small transfer, date last large transfer

TIME	USER ID	EVENT	TRANSFER_TO	AMOUNT
1/1/2012	John	Deposit	-	1,000,000
1/2/2012	John	Transfer	Bob	1,000
1/5/2012	John	Withdrawal	-	2,000
1/10/2012	John	Transfer	Allen	1,500
1/20/2012	John	Transfer	Tim	1,200
1/25/2012	John	Deposit		1,200,000
1/27/2012	John	Transfer	Tim	1,000,000
2/2/20212	John	Deposit	-	500,000

Three small transfers within 30 days to different acct and total sum < 20K

Large transfer within 10 days of last small transfer

SQL Pattern Matching

Example Suspicious Money Transfers - Refined

```
SELECT userid, first_t, last_t, amount
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE
(  PARTITION BY userid ORDER BY time
   MEASURES  FIRST(x.time) first_t, y.time last_t, y.amount amount
   PATTERN ( z x{2,} y )
   DEFINE  z as (event='transfer' and amount < 2000),
           x as (event='transfer' and amount < 2000 AND
                prev(x.transfer_to) <> x.transfer_to ),
           y as (event='transfer' and amount >= 1000000 AND
                   last(x.time) - first(x.time) < 30  AND
                   y.time - last(x.time) < 10 AND
                   SUM(x.amount) + z.amount < 20000 )
)
```

First small transfer

Next two or more small transfers to different accts

Sum of all small transfers less then 20000

 | |

Native Top N Support

Native Support for TOP-N Queries

Natively identify top N in SQL

Significantly simplifies code development

ANSI SQL:2008

"Who are the top 5 money makers in my enterprise?"

```
SELECT empno, ename, deptno
FROM emp
ORDER BY sal, comm FETCH FIRST 5 ROWS ONLY;
```

versus

```
SELECT empno, ename, deptno
FROM (SELECT empno, ename, deptno, sal, comm,
             row_number() OVER (ORDER BY sal,comm) rn
      FROM emp
      )
WHERE rn <=5
ORDER BY sal, comm;
```


Native Support for TOP-N Queries

New offset and fetch_first clause

- ANSI 2008/2011 compliant with some additional extensions
- Specify offset and number or percentage of rows to return
- Provisions to return additional rows with the same sort key as the last row (WITH TIES option)
- Syntax:

```
OFFSET <offset> [ROW | ROWS]
FETCH [FIRST | NEXT]
 [<rowcount> | <percent> PERCENT] [ROW | ROWS]
 [ONLY | WITH TIES]
```

Native Support for TOP-N Queries

Internal processing

- Find 5 percent of employees with the lowest salaries

```
SELECT employee_id, last_name, salary
FROM employees
ORDER BY salary
FETCH FIRST 5 percent ROWS ONLY;
```

ORACLE

Native Support for TOP-N Queries

Internal processing, cont.

- Find 5 percent of employees with the lowest salaries

```
SELECT employee_id, last_name, salary
FROM employees
ORDER BY salary
FETCH FIRST 5 per
```

- Internally the query is transformed into an equivalent query using window functions

```
SELECT employee_id, last_name, salary
FROM (SELECT employee_id, last_name, salary,
             row_number() over (order by salary) rn,
             count(*) over () total
      FROM employee)
WHERE rn <= CEIL(total * 5/100);
```

- Additional Top-N Optimization:
  - SELECT list may include expensive PL/SQL function or costly expressions
  - Evaluation of SELECT list expression limited to rows in the final result set

ORACLE

SQL Evolution

ORACLE DATABASE 12c

- Pattern matching
- Top N clause
- Lateral Views,APPLY
- Identity Columns
- Column Defaults
- Data Mining III

ORACLE DATABASE 11g

- Data mining II
- SQL Pivot
- Recursive WITH
- ListAgg, N_Th value window

ORACLE 10g DATABASE

- Statistical functions
- Sql model clause
- Partition Outer Join
- Data mining I

DATABASE 9i CLUSTER

- Enhanced Window functions (percentile,etc)
- Rollup, grouping sets, cube

ORACLE 8i INTERNET

- Introduction of Window functions

1998 → 2001 → 2002 → 2004 → 2005 → 2007 → 2009 → 2012

ORACLE

Pattern Matching

Finding Double Bottom (W)

```java
        if (!q.isEmpty() && (next.isEmpty() || (gt(q, prev) && eq(q, next)))) {
            state = "E";
            return state;
        }

        if (q.isEmpty() || eq(q, prev)) {
            state = "F";
            return state;
        }

        return state;
    }

    private boolean eq(String a, String b) {
        if (a.isEmpty() || b.isEmpty()) {
            return false;
        }
        return a.equals(b);
    }

    private boolean gt(String a, String b) {
        if (a.isEmpty() || b.isEmpty()) {
            return false;
        }
        return Double.parseDouble(a) > Double.parseDouble(b);
    }

    private boolean lt(String a, String b) {
        if (a.isEmpty() || b.isEmpty()) {
            return false;
        }
        return Double.parseDouble(a) < Double.parseDouble(b);
    }

    public String getState() {
        return this.state;
    }
```

```sql
SELECT first_x, last_z
FROM ticker MATCH_RECOGNIZE (
       PARTITION BY name ORDER BY time
       MEASURES FIRST(x.time) AS first_x,
                LAST(z.time)  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.time - FIRST(x.time) <= 7 ))
```

250+ Lines of Java and PIG

12 Lines of SQL

20x less code, 5x faster

ORACLE®