Analyze this!
Analytical power in SQL, more than you ever dreamt of
Andrew Witkowski
Architect
Analytical SQL in the Database

1998
• Introduction of Window functions
  • Enhanced Window functions (percentile, etc)
  • Rollup, grouping sets, cube

2001
• Statistical functions
  • SQL model clause
  • Partition Outer Join
  • Data mining I

2002
• Pattern matching
  • Top N clause
  • Lateral Views, APPLY
  • Identity Columns
  • Column Defaults
  • Data Mining III

2004
• Data mining II
  • SQL Pivot
  • Recursive WITH

2005
• ListAgg, N_Th value window

2007

2009

2012
SQL Pattern Matching
Pattern Recognition In Sequences of Rows

The Challenge

“Find people that flew from country X to country Y, stayed there 2 days, then went to country Z, stayed there 30 days, contacted person A, and then withdrew $10,000.00”

- Currently pattern recognition in SQL is **difficult**
  - Use multiple self joins (not good for *)
    - T1.person = T2.person AND T1.country= ‘X’ AND T2.country= ‘Y’ & 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)
Pattern Recognition 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"

<table>
<thead>
<tr>
<th>EVENT</th>
<th>TIME</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>SFO</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>SFO</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>ATL</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>LAX</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>SFO</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>LAX</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>LAS</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>SFO</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>NYC</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>NYC</td>
</tr>
</tbody>
</table>

A+ B C - perl
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
SQL Pattern Matching

Example: Find Double Bottom (W)

Find double bottom (W) patterns and report:

- Beginning and ending date of the pattern
- Average Price Increase in the second ascent
- Modify the search to find only patterns that lasted less than a week
SQL Pattern Matching

Example: Find Double Bottom (W)

Find double bottom (W) patterns and report:

- Beginning and ending date of the pattern
- Average Price Increase in the second ascent
- Modify the search to find only patterns that lasted less than a week
SQL Pattern Matching

Example: Find Double Bottom (W)

Find double bottom (W) patterns and report:

• Beginning and ending date of the pattern

• Average Price Increase in the second ascent

• Modify the search to find only patterns that lasted less than a week

PATTERN (X+ Y+ W+ Z+)
DEFINE X AS (price < PREV(price))
SQL Pattern Matching

Example: Find Double Bottom (W)

Find double bottom (W) patterns and report:

- Beginning and ending date of the pattern
- Average Price Increase in the second ascent
- Modify the search to find only patterns that lasted less than a week

PATTERN (X + Y+ W+ Z+)
DEFINE X AS (price < PREV(price))
SQL Pattern Matching

Example: Find Double Bottom (W)

Find double bottom (W) patterns and report:

- Beginning and ending date of the pattern
- Average Price Increase in the second ascent
- Modify the search to find only patterns that lasted less than a week

PATTERN (X+ Y+ W+ Z+)

DEFINE X AS (price < PREV(price))
DEFINE Y AS (price > PREV(price))
SQL Pattern Matching

Example: Find Double Bottom (W)

Find double bottom (W) patterns and report:

- Beginning and ending date of the pattern
- Average Price Increase in the second ascent
- Modify the search to find only patterns 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))
```
Find double bottom (W) patterns and report:

- Beginning and ending date of the pattern
- Average Price Increase in the second ascent
- Modify the search to find only patterns 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)))
```
SQL Pattern Matching

Example: Find Double Bottom (W)

Find double bottom (W) patterns and report:

- Beginning and ending date of the pattern
- Average Price Increase in the second ascent
- Modify the search to find only patterns that lasted less than a week

```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 )
)
```
SQL Pattern Matching

Example: Find Double Bottom (W)

Find double bottom (W) patterns and report:

- Beginning and ending date of the pattern
- Average Price Increase in the second ascent
- Modify the search to find only patterns that lasted less than a week

```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 )
```
**SQL Pattern Matching**

**Example: Find Double Bottom (W)**

Find double bottom (W) patterns and report:

- Beginning and ending date of the pattern
- Average Price in the second ascent
- Modify the search to find only patterns that lasted less than a week

```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,
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 ))
```

Average stock price: $52.00
SQL Pattern Matching

Syntax

```sql
<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 an 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)

...
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)
SQL Pattern Matching

“Declarative” Pattern Matching, cont.

- Running aggregates in conditions on currently defined variables:
  
  - DEFINE X AS (price < PREV(price) AND AVG(num_of_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
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 and 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)
);
```
Detect ALL login events after privileges have been revoked for the user.

Generate a row for first improper login attempt (event)

---

**SQL Pattern Matching**

**ALL ROWS PER MATCH OPTION**

```sql
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')
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>EVENT</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>grant</td>
<td>9:00 AM</td>
</tr>
<tr>
<td>John</td>
<td>revoke</td>
<td>1:00 PM</td>
</tr>
<tr>
<td>John</td>
<td>fired</td>
<td>1:20 PM</td>
</tr>
<tr>
<td>John</td>
<td>escorted</td>
<td>1:25 PM</td>
</tr>
<tr>
<td>John</td>
<td>left</td>
<td>1:30 PM</td>
</tr>
<tr>
<td>John</td>
<td>login</td>
<td>1:50 PM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>REV_TIME</th>
<th>TIME</th>
<th>CLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1:00 PM</td>
<td>1:00 PM</td>
<td>X</td>
</tr>
<tr>
<td>John</td>
<td>1:00 PM</td>
<td>1:20 PM</td>
<td>Y</td>
</tr>
<tr>
<td>John</td>
<td>1:00 PM</td>
<td>1:25 PM</td>
<td>Y</td>
</tr>
<tr>
<td>John</td>
<td>1:00 PM</td>
<td>1:30 PM</td>
<td>y</td>
</tr>
<tr>
<td>John</td>
<td>1:00 PM</td>
<td>1:50 PM</td>
<td>Z</td>
</tr>
</tbody>
</table>
SQL Pattern Matching
ONE ROW PER MATCH OPTION

Detect each 3 or more consecutive login attempt (event) after privileges have been revoked

Login attempts all have to occur within 1 minute

```
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)
```
SQL Pattern Matching

Sample 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
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 for user log**

<table>
<thead>
<tr>
<th>TIME</th>
<th>USER ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary</td>
</tr>
<tr>
<td>2</td>
<td>Sam</td>
</tr>
<tr>
<td>11</td>
<td>Mary</td>
</tr>
<tr>
<td>12</td>
<td>Sam</td>
</tr>
<tr>
<td>22</td>
<td>Sam</td>
</tr>
<tr>
<td>23</td>
<td>Mary</td>
</tr>
<tr>
<td>32</td>
<td>Sam</td>
</tr>
<tr>
<td>34</td>
<td>Mary</td>
</tr>
<tr>
<td>43</td>
<td>Sam</td>
</tr>
<tr>
<td>44</td>
<td>Mary</td>
</tr>
<tr>
<td>47</td>
<td>Sam</td>
</tr>
<tr>
<td>48</td>
<td>Sam</td>
</tr>
<tr>
<td>53</td>
<td>Mary</td>
</tr>
<tr>
<td>59</td>
<td>Sam</td>
</tr>
<tr>
<td>60</td>
<td>Sam</td>
</tr>
<tr>
<td>63</td>
<td>Mary</td>
</tr>
<tr>
<td>68</td>
<td>Sam</td>
</tr>
</tbody>
</table>

**Identify sessions**

<table>
<thead>
<tr>
<th>TIME</th>
<th>USER ID</th>
<th>SESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Mary</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>Mary</td>
<td>2</td>
</tr>
<tr>
<td>34</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>44</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>53</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>63</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>43</td>
<td>Sam</td>
<td>2</td>
</tr>
<tr>
<td>47</td>
<td>Sam</td>
<td>2</td>
</tr>
<tr>
<td>48</td>
<td>Sam</td>
<td>2</td>
</tr>
<tr>
<td>59</td>
<td>Sam</td>
<td>3</td>
</tr>
<tr>
<td>60</td>
<td>Sam</td>
<td>3</td>
</tr>
<tr>
<td>68</td>
<td>Sam</td>
<td>3</td>
</tr>
</tbody>
</table>

**Number Sessions per user**

<table>
<thead>
<tr>
<th>TIME</th>
<th>USER ID</th>
<th>SESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Mary</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>Mary</td>
<td>2</td>
</tr>
<tr>
<td>34</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>44</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>53</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>63</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>43</td>
<td>Sam</td>
<td>2</td>
</tr>
<tr>
<td>47</td>
<td>Sam</td>
<td>2</td>
</tr>
<tr>
<td>48</td>
<td>Sam</td>
<td>2</td>
</tr>
<tr>
<td>59</td>
<td>Sam</td>
<td>3</td>
</tr>
<tr>
<td>60</td>
<td>Sam</td>
<td>3</td>
</tr>
<tr>
<td>68</td>
<td>Sam</td>
<td>3</td>
</tr>
</tbody>
</table>
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: ONE ROW PER MATCH

```sql
SELECT user_id, session_id, start_time, no_of_events, duration
FROM Events MATCH_RECOGNIZE
    ( PARTITION BY User_ID ORDER BY time
    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 – Aggregation of sessionized data

<table>
<thead>
<tr>
<th>TIME</th>
<th>USER ID</th>
<th>SESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Mary</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>Mary</td>
<td>2</td>
</tr>
<tr>
<td>34</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>44</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>53</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>63</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>Sam</td>
<td>1</td>
</tr>
<tr>
<td>43</td>
<td>Sam</td>
<td>2</td>
</tr>
<tr>
<td>47</td>
<td>Sam</td>
<td>2</td>
</tr>
<tr>
<td>48</td>
<td>Sam</td>
<td>2</td>
</tr>
<tr>
<td>59</td>
<td>Sam</td>
<td>3</td>
</tr>
<tr>
<td>60</td>
<td>Sam</td>
<td>3</td>
</tr>
<tr>
<td>68</td>
<td>Sam</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIME</th>
<th>SESSION_ID</th>
<th>START_TIME</th>
<th>NUM EVENTS</th>
<th>DURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Mary</td>
<td>2</td>
<td>23</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Mary</td>
<td>3</td>
<td>34</td>
<td>4</td>
<td>29</td>
</tr>
<tr>
<td>Sam</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>Sam</td>
<td>2</td>
<td>43</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Sam</td>
<td>3</td>
<td>59</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>
**SQL Pattern Matching**

**Example Sessionization – using window functions**

```sql
CREATE VIEW Sessionized_Events as
SELECT Time_Stamp, User_ID,
    Sum(Session_Increment) over (partition by User_ID order by Time_Stamp asc) Session_ID
FROM (SELECT Time_Stamp, User_ID,
    CASE WHEN (Time_Stamp - Lag(Time_Stamp) over (partition by User_ID order by Time_Stamp asc)) < 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
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

<table>
<thead>
<tr>
<th>TIME</th>
<th>USER ID</th>
<th>EVENT</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/2012</td>
<td>John</td>
<td>Deposit</td>
<td>1,000,000</td>
</tr>
<tr>
<td>1/2/2012</td>
<td>John</td>
<td>Transfer</td>
<td>1,000</td>
</tr>
<tr>
<td>1/5/2012</td>
<td>John</td>
<td>Withdrawal</td>
<td>2,000</td>
</tr>
<tr>
<td>1/10/2012</td>
<td>John</td>
<td>Transfer</td>
<td>1,500</td>
</tr>
<tr>
<td>1/20/2012</td>
<td>John</td>
<td>Transfer</td>
<td>1,200</td>
</tr>
<tr>
<td>1/25/2012</td>
<td>John</td>
<td>Deposit</td>
<td>1,200,000</td>
</tr>
<tr>
<td>1/27/2012</td>
<td>John</td>
<td>Transfer</td>
<td>1,000,000</td>
</tr>
<tr>
<td>2/2/2012</td>
<td>John</td>
<td>Deposit</td>
<td>500,000</td>
</tr>
</tbody>
</table>

Three small transfers within 30 days

Large transfer within 10 days of last small transfer
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

<table>
<thead>
<tr>
<th>TIME</th>
<th>USER ID</th>
<th>EVENT</th>
<th>TRANSFER_TO</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/2012</td>
<td>John</td>
<td>Deposit</td>
<td>-</td>
<td>1,000,000</td>
</tr>
<tr>
<td>1/2/2012</td>
<td>John</td>
<td>Transfer</td>
<td>Bob</td>
<td>1,000</td>
</tr>
<tr>
<td>1/5/2012</td>
<td>John</td>
<td>Withdrawal</td>
<td>-</td>
<td>2,000</td>
</tr>
<tr>
<td>1/10/2012</td>
<td>John</td>
<td>Transfer</td>
<td>Allen</td>
<td>1,500</td>
</tr>
<tr>
<td>1/20/2012</td>
<td>John</td>
<td>Transfer</td>
<td>Tim</td>
<td>1,200</td>
</tr>
<tr>
<td>1/25/2012</td>
<td>John</td>
<td>Deposit</td>
<td></td>
<td>1,200,000</td>
</tr>
<tr>
<td>1/27/2012</td>
<td>John</td>
<td>Transfer</td>
<td>Tim</td>
<td>1,000,000</td>
</tr>
<tr>
<td>2/2/2012</td>
<td>John</td>
<td>Deposit</td>
<td>-</td>
<td>500,000</td>
</tr>
</tbody>
</table>

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 )
  )
```
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?"

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

versus

```sql
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:

```sql
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;
```
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 percent ROWS ONLY;
```

- 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
Analytical SQL in the Database

- **1998**
  - Introduction of Window functions
- **2001**
  - Enhanced Window functions (percentile, etc)
  - Rollup, grouping sets, cube
- **2002**
  - Statistical functions
  - Sql model clause
  - Partition Outer Join
  - Data mining I
- **2004**
- **2005**
- **2007**
- **2009**
- **2012**
  - Pattern matching
  - Top N clause
  - Lateral Views, APPLY
  - Identity Columns
  - Column Defaults
  - Data Mining II
  - Data mining III
  - SQL Pivot
  - Recursive WITH
  - ListAgg, N_Th value window
  - Pattern matching
  - Top N clause
  - Lateral Views, APPLY
  - Identity Columns
  - Column Defaults
  - Data Mining II
  - Data mining III
  - SQL Pivot
  - Recursive WITH
  - ListAgg, N_Th value window
Hardware and Software Engineered to Work Together