SQL - the best development language for Big Data?

Exploring the Analytical Power of SQL in Oracle Database 12c
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract.

It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Typical use cases in today’s world of fast exploration of big data
The On-Going Evolution of SQL

- Introduction of Window functions
- Enhanced Window functions (percentile, etc)
- Rollup, grouping sets, cube
- Statistical functions
- SQL model clause
- Partition Outer Join
- Data mining I
- Data mining II
- SQL Pivot
- Recursive WITH
- ListAgg, Nth value window
- Data mining III
- Pattern matching
- Top N clause
- Identity Columns
- Column Defaults
- Data Mining III
Pattern Matching with SQL Analytics

Java vs. SQL: Stock Markets - Searching for ‘W’ Patterns in Trade Data

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)) AND z.time - FIRST(x.time) <= 7 ))

250+ Lines of Java and PIG

12 Lines of SQL

SQL - 20x less code, 5x faster
With Sessionized_Call_Details as
(select Caller, Callee, Start_Time, End_Time,
      Session_ID
from Call_Details)

Inter_Call_Intrvl as
(select Caller, Callee, Start_Time, End_Time,
      Session_ID,
      sum(case when Inter_Call_Intrvl < 60
      then 0 else 1 end) over(partition by Caller, Callee
      order by Start_Time)
from Sessionized_Call_Details)

Select Caller, Callee,
      Min(Start_Time) Start_Time,
      Effective_Call_Duration,
      Total_Interruption_Duration, No_Of_Restarts,
      Session_ID
from Inter_Call_Intrvl
group by Caller, Callee, Session_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

Key Concepts
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 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)),
```

Stock price

days

Find the \text{X}\text{Y} pattern in a ticker stream:
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)))
```
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

Example: Find W-Shape

```sql
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

Find a W-shape pattern in a ticker stream:

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

```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)))
```

<table>
<thead>
<tr>
<th>First_x</th>
<th>Last_z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>19</td>
</tr>
</tbody>
</table>

Stock price

(days)
SQL Pattern Matching in action

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 ))
```

Can refer to previous variables

Stock price

<table>
<thead>
<tr>
<th>days</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>19</td>
</tr>
</tbody>
</table>
SQL Pattern Matching in action

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, avg_price
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 in action

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 in action

#### 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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary</td>
</tr>
<tr>
<td>11</td>
<td>Mary</td>
</tr>
<tr>
<td>23</td>
<td>Mary</td>
</tr>
<tr>
<td>34</td>
<td>Mary</td>
</tr>
<tr>
<td>53</td>
<td>Mary</td>
</tr>
<tr>
<td>63</td>
<td>Mary</td>
</tr>
<tr>
<td>2</td>
<td>Sam</td>
</tr>
<tr>
<td>12</td>
<td>Sam</td>
</tr>
<tr>
<td>22</td>
<td>Sam</td>
</tr>
<tr>
<td>32</td>
<td>Sam</td>
</tr>
<tr>
<td>43</td>
<td>Sam</td>
</tr>
<tr>
<td>47</td>
<td>Sam</td>
</tr>
<tr>
<td>48</td>
<td>Sam</td>
</tr>
<tr>
<td>59</td>
<td>Sam</td>
</tr>
<tr>
<td>60</td>
<td>Sam</td>
</tr>
<tr>
<td>68</td>
<td>Sam</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>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 in action

Example Sessionization for user log: MATCH_RECOGNIZE

```
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 in action

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 in action

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>

#### Aggregate sessions per user

<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>
Example Sessionization – Aggregation: ONE ROW PER MATCH

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;
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 ONLY;
```

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:

```
OFFSET <offset> [ROW | ROWS]
FETCH [FIRST | NEXT]
[<rowcount> | <percent> PERCENT] [ROW | ROWS]
[ONLY | WITH TIES]
```
Summary
New Database 12c SQL Analytics

- ANSI compliant features with some additional extensions
- Common syntax reduces learning curve
- Comprehensive support for SQL based pattern matching
  - Supports a wide range of use cases
  - Simplifies application development
  - Simplifies existing SQL code
- New TOP-N feature
  - Simplifies existing SQL code
SQL - the best development language for Big Data?

Yes, because SQL is….

- SIMPLER
- FASTER
- RICHER
HARDWARE AND SOFTWARE ENGINEERED TO WORK TOGETHER
<table>
<thead>
<tr>
<th>Session</th>
<th>Date</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern Matching Hands-on Lab</td>
<td>Tues - 12:00pm</td>
<td>Marriot Salon 3-4</td>
</tr>
<tr>
<td>Top Tips for Mastering Oracle Partitioning</td>
<td>Tues - 3:45pm</td>
<td>Moscone South 103</td>
</tr>
<tr>
<td>Oracle Optimizer Boot Camp</td>
<td>Tues - 5:15pm</td>
<td>Moscone South 102</td>
</tr>
<tr>
<td>In-Database MapReduce using SQL</td>
<td>Wed - 10:15am</td>
<td>Marriot Salon 7</td>
</tr>
<tr>
<td>Programming with Big Data Connectors</td>
<td>Wed – 3:30pm</td>
<td>Marriot Salon 7</td>
</tr>
<tr>
<td>Data Warehouse &amp; Big Data – Customer panel</td>
<td>Wed – 3:30pm</td>
<td>Moscone South 300</td>
</tr>
<tr>
<td>Your Data is talking to you – Customer panel</td>
<td>Wed – 5:00pm</td>
<td>Moscone South 300</td>
</tr>
</tbody>
</table>
Where to get more information

- SQL Analytics Home Page on OTN
  - Oracle By Example – Pattern matching
  - Podcasts for pattern matching and SQL analytics
  - Data Sheet
  - Whitepapers
    - Patterns Everywhere - Find them fast!
    - Patterns Everywhere - Find them fast! (Apple iBook)

- Data Warehouse and SQL Analytics blog
  - [http://oracle-big-data.blogspot.co.uk/](http://oracle-big-data.blogspot.co.uk/)
THANK YOU FOR JOINING US TODAY
ENJOY OPENWORLD