SQL - the best analysis language for Big Data!

NoCOUG Winter Conference 2014

Hermann Bär, hermann.baer@oracle.com
Data Warehousing Product Management, Oracle
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
- Pattern matching
- Top N clause
- Data Mining III
SQL for Analysis and Reporting

Benefits

- Dramatically enhanced analysis capabilities with SQL
  - Native support, e.g. OBI EE
  - Embedding into SQL views

- Simplified development
  - Investment protection through ANSI standard compliance

- Increased performance
  - New language constructs enable more efficient plans
  - Internal optimizations
Fundamental Concepts
Key Concepts
Same for all functions - Unified SQL

- **Partitions**
  - Groupings of rows within a query result set

- **Orderings**
  - Rows can be ordered within a partition

- **Windows** (logical or physical)
  - A moving group of rows within a partition
  - Defines the range of an aggregate calculation

- **Current Row**
Reporting Aggregates

Compare total sales of regions with total sales

```
SELECT person, region, sales
    SUM (sales) OVER (PARTITION BY (region) ) r_sales
    SUM (sales) OVER () t_sales
FROM sales_table
ORDER BY region, s_rank;
```

<table>
<thead>
<tr>
<th>PERSON</th>
<th>REGION</th>
<th>SALES</th>
<th>R_SALES</th>
<th>T_SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adams</td>
<td>East</td>
<td>200</td>
<td>530</td>
<td>1130</td>
</tr>
<tr>
<td>Connor</td>
<td>East</td>
<td>180</td>
<td>530</td>
<td>1130</td>
</tr>
<tr>
<td>Baker</td>
<td>East</td>
<td>150</td>
<td>530</td>
<td>1130</td>
</tr>
<tr>
<td>Donner</td>
<td>West</td>
<td>300</td>
<td>600</td>
<td>1130</td>
</tr>
<tr>
<td>Edward</td>
<td>West</td>
<td>200</td>
<td>600</td>
<td>1130</td>
</tr>
<tr>
<td>Witkowski</td>
<td>West</td>
<td>100</td>
<td>600</td>
<td>1130</td>
</tr>
</tbody>
</table>
### Lag/Lead Functions

How does sales compare versus this month last year?

```sql
SELECT timekey, sales,
    LAG(sales,12)OVER(ORDER BY timekey) as sales_last_year,
    (sales - sales_last_year) as sales_change
FROM sales;
```

<table>
<thead>
<tr>
<th>TIMEKEY</th>
<th>SALES</th>
<th>SALES LAST YEAR</th>
<th>SALES CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01</td>
<td>1100</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>..</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-01</td>
<td>2000</td>
<td>1100</td>
<td>900</td>
</tr>
<tr>
<td>2010-02</td>
<td>1800</td>
<td>1200</td>
<td>600</td>
</tr>
<tr>
<td>..</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011-01</td>
<td>1900</td>
<td>2000</td>
<td>-100</td>
</tr>
</tbody>
</table>
Without analytical function

```sql
select avg(b.event_datetime - a.prior_event_datetime) as avg_wait
from
(select
  a.*, rownum as row_number
from
  (select
    order_id, event_datetime, new_event_cd as prior_event_cd
  from order_pipeline_events
  where
    warehouse_id = 'RNO1'
    and event_datetime > sysdate - 2
  order by
    order_id, event_datetime
    , new_event_cd
) a
) a ,
(select
  b.*, rownum as row_number
from
  (select
    order_id, event_datetime, new_event_cd as event_cd
  from order_pipeline_events
  where
    warehouse_id = 'RNO1'
    and event_datetime > sysdate - 2
  order by
    order_id, event_datetime
    , new_event_cd
) b
) b
where
  a.order_id = b.order_id
  and a.prior_event_cd = '1001'
  and b.event_cd = '1002'
  and a.row_number = b.row_number - 1;
```

With analytical function

```sql
Select avg(event_datetime - prior_event_datetime) as avg_wait
from
  (select
    new_event_cd as event_cd, event_datetime,
    lag(new_event_cd) over (partition by order_id order by event_datetime, new_event_cd) as prior_event_cd,
    lag(event_datetime) over (partition by order_id order by event_datetime, new_event_cd) as prior_event_datetime
  from order_pipeline_events
  where
    warehouse_id = 'RNO1'
    and event_datetime > sysdate - 2
  ) a,
( select
    b.*, rownum as row_number
  from
    ( select
      order_id, event_datetime, new_event_cd
    from order_pipeline_events
    where
      warehouse_id = 'RNO1'
      and event_datetime > sysdate - 2
    ) b
  ) b
where
  prior_event_cd = '1001'
  and event_cd = '1002';
```
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 ...”

- Prior to Oracle Database 12c 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)
**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"

<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)**

<table>
<thead>
<tr>
<th>EVENT</th>
<th>TIME</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>
Finding Patterns in Big Data

Typical use cases in today’s world of fast exploration of big data

- Financial Services
  - Money Laundering
  - Tracking Stock Market
- Utilities
  - Fraud
  - Network Analysis
- Law & Order
  - Unusual Usage
  - Monitoring Suspicious Activities
- Retail
  - Buying Patterns
  - Returns Fraud
  - Session-ization
- Telcos
  - Call Quality
  - SIM Card Fraud
  - Money Laundering
SQL Pattern Matching

Conceptual Example
SQL Pattern Matching in Action

Basic steps for building the SQL command

1. Define the partitions/buckets and ordering needed to identify the ‘stream of events’ you are analyzing
2. Define the pattern of events and pattern variables identifying the individual events within the pattern
3. Define measures: source data points, pattern data points and aggregates related to a pattern
4. Determine how the output will be generated
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 ( )

1. Define the partitions/buckets and ordering needed to identify the ‘stream of events’ you are analyzing

* For conceptual clarity, the statement is simplified and ignores an always-true start event. See the notes or documentation for further explanation
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.

2. Define the pattern of events and pattern variables identifying the individual events within the pattern

```
SELECT ... 
FROM ticker MATCH RECOGNIZE ( 
    PARTITION BY name ORDER BY time 
)
```

* For conceptual clarity, the statement is simplified and ignores an always-true start event. See the notes or documentation for further explanation.
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+)
```

* For conceptual clarity, the statement is simplified and ignores an always-true start event. See the notes or documentation for further explanation.
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)),
```

* For conceptual clarity, the statement is simplified and ignores an always-true start event. See the notes or documentation for further explanation.
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)),
```

* For conceptual clarity, the statement is simplified and ignores an always-true start event. See the notes or documentation for further explanation.
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”

3. Define measures: source data points, pattern data points and aggregates related to a pattern

* For conceptual clarity, the statement is simplified and ignores an always-true start event. See the notes or documentation for further explanation.
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)),

* For conceptual clarity, the statement is simplified and ignores an always-true start event. See the notes or documentation for further explanation.
Find a W-shape pattern in a ticker stream:

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

SQL Pattern Matching in Action

Example: Find W-Shape*

```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))
)
```

First_x | Last_z
--- | ---
1 | 9
13 | 19

* For conceptual clarity, the statement is simplified and ignores an always-true start event. See the notes or documentation for further explanation
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 ))

* For conceptual clarity, the statement is simplified and ignores an always-true start event. See the notes or documentation for further explanation
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, 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

* For conceptual clarity, the statement is simplified and ignores an always-true start event. See the notes or documentation for further explanation.
SQL Pattern Matching

“Declarative” Pattern Matching

1. Define the partitions/buckets and ordering needed to identify the ‘stream of events’ you are analyzing
   - Matching within a stream of events (ordered partition of data)
     - MATCH_RECOGNIZE (PARTITION BY stock_name ORDER BY time MEASURES ...)

2. Define the pattern of events and pattern variables identifying the individual events within the pattern
   - Use framework of Perl regular expressions (conditions on rows)
     - PATTERN (X+ Y+ W+ Z+)
   - Define matching using Boolean conditions on rows
     - DEFINE X AS (price > 15)
3. Define measures: source data points, pattern data points and aggregates related to a pattern
   - MEASURES FIRST(x.time) AS first_x,
     LAST(z.time)   AS last_z,
     AVG(z.price)  AS avg_price

4. Determine how the output will be generated
   - ONE ROW PER MATCH
SQL Pattern Matching

MATCH_RECOGNIZE Syntax

\[
\text{<table_expression>} := \text{<table_expression>} \ MATCH\_RECOGNIZE \\
\begin{align*}
&\quad ( \ [ \ \text{PARTITION\ BY}\ \text{<cols>} \ ] \\
&\quad [ \ \text{ORDER\ BY}\ \text{<cols>} \ ] \\
&\quad [ \ \text{MEASURES}\ \text{<cols>} \ ] \\
&\quad [ \ \text{ONE\ ROW\ PER\ MATCH}\ |\ \text{ALL\ ROWS\ PER\ MATCH}\ ] \\
&\quad [ \ \text{SKIP\_TO\_option}\ ] \\
&\quad \text{PATTERN}\ (\ \text{<row\ pattern>}\ ) \\
&\quad [ \ \text{SUBSET}\ \text{<subset\ list>}\ ] \\
&\quad \text{DEFINE}\ \text{<definition\ list>} \\
\end{align*}
\]
SQL Pattern Matching

I’ll get my hands dirty …
ACME Data Set
SQL Pattern Matching

Real world use cases
Help about recursive query

This question is Not Answered.

Hi,

I've a table with 4 columns
field1  number
field2  number
field3  char
field4  date

I want to extract field1 and field2 (field1 and 2 must be the same for the rows) where there are 3 consecutive field3="X" ordered by field4 desc.

I want to extract this:

```
field1  field2  field3  field4
1       1       X       10/10/2013
1       1       X       10/9/2013
1       1       X       10/8/2013
```

But not this:

```
field1  field2  field3  field4
1       1       X       10/10/2013
1       1       X       10/9/2013
1       1       Y       10/8/2013
1       1       X       10/8/2013
```
Depending on your answer, you might want something like this:

```sql
WITH got_grp AS
(
    SELECT field1, field2, field3, field4,
    ROW_NUMBER() OVER (PARTITION BY
                     field1, field2, field3,
                     ORDER BY
                     field4) as grp
    FROM a
),
got_grp_cnt AS
(
    SELECT field1, field2, field3, field4,
    COUNT(*) OVER (PARTITION BY field1, field2, field3) as cnt
    FROM got_grp
    WHERE field3 = 'X'
)
SELECT field1, field2, field3, field4
FROM got_grp_cnt
WHERE cnt = 3
ORDER BY field1, field2, field4 DESC;
```

The SQL statement to find the exact three records for each pair (field1,field2) looks as follows:

```sql
SQL> select * from tt
match_recognize (partition by field1, field2 order by field4 desc
all rows per match
pattern (strt a(2))
define
a as field3=prev(field3));
```

<table>
<thead>
<tr>
<th>FIELD1</th>
<th>FIELD2</th>
<th>FIELD4</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>10-OCT-13</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>09-OCT-13</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>08-OCT-13</td>
<td>X</td>
</tr>
</tbody>
</table>
```java
if (q.isEmpty() && (gt(q, prev) || eq(q, next))) {
    state = "S";
    return state;
}
if (gt(q, prev) && gt(q, next)) {
    state = "T";
    return state;
}
if (lt(q, prev) && lt(q, next)) {
    state = "B";
    return state;
}
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;
}
```

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

250+ Lines of Java and PIG

12 Lines of SQL

20x less code, 5x faster
Analytical SQL in the Database

Summary

- Ranking functions
  - rank, dense_rank, cume_dist, percent_rank, ntile
- Window Aggregate functions (moving and cumulative)
  - Avg, sum, min, max, count, variance, stddev, first_value, last_value
- LAG/LEAD functions
  - Direct inter-row reference using offsets
- Reporting Aggregate functions
  - Sum, avg, min, max, variance, stddev, count, ratio_to_report
- Statistical Aggregates
  - Correlation, linear regression family, covariance
- Linear regression
  - Fitting of an ordinary-least-squares regression line to a set of number pairs.
  - Frequently combined with the COVAR_POP, COVAR_SAMP, and CORR functions
- Descriptive Statistics
  - DBMS_STAT_FUNCS: summarizes numerical columns of a table and returns count, min, max, range, mean, stats_mode, variance, standard deviation, median, quantile values, +/- n sigma values, top/bottom 5 values
- Correlations
  - Pearson’s correlation coefficients, Spearman’s and Kendall’s (both nonparametric).
- Cross Tabs
  - Enhanced with % statistics: chi squared, phi coefficient, Cramer’s V, contingency coefficient, Cohen’s kappa
- Hypothesis Testing
  - Student t-test, F-test, Binomial test, Wilcoxon Signed Ranks test, Chi-square, Mann Whitney test, Kolmogorov-Smirnov test, One-way ANOVA
- Distribution Fitting
  - Kolmogorov-Smirnov Test, Anderson-Darling Test, Chi-Squared Test, Normal, Uniform, Weibull, Exponential
Summary
New Database 12c SQL Analytics

- Comprehensive analysis with SQL out of the box
  - ANSI compliant features with some additional extensions
- Common language SQL speeds up adoption
  - Widely known and used
  - 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
SQL - the best development language for Big Data?

Yes, because SQL is….

SIMPLER  RICHER  FASTER
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 then fast! ([Apple iBook](http://oracle-big-data.blogspot.co.uk/))

- Data Warehouse and SQL Analytics blog
  - [http://oracle-big-data.blogspot.co.uk/](http://oracle-big-data.blogspot.co.uk/)
Hardware and Software
Engineered to Work Together