

# Introduction to solving SQL problems with **MATCH\_RECOGNIZE**



**About me...**

**Keith Laker**

Senior Principal Product Management  
SQL and Data Warehousing

SQL enthusiast, marathon runner, mountain  
biker and coffee connoisseur



**@ASQLBarista**

# Agenda

- 1 What is MATCH\_RECOGNIZE
- 2 Use Case 1: sessionization
- 3 Use Case 2: controlling string concatenation
- 4 Use Case 3: forming contiguous date ranges
- 5 Summary

# Lots of tutorials on <http://livesql.oracle.com>

My Tutorials

Search My Tutorials

**MATCH\_RECOGNIZE - Empty Matches and Unmatched Rows**

The aim of this tutorial is to explain the difference between the various row output options within MATC\_RECOGNIZE, specifically the EMPTY MATCHES and UNMATCHED ROWS

Analytics / Public

**MATCH\_RECOGNIZE - importance of PARTITION BY and ORDER BY**

The aim of this tutorial is to explain the importance of using PARTITION BY and ORDER BY to ensure the correct results are returned and explores how and when predicates are applied.

Analytics / Public

**MATCH\_RECOGNIZE - What to include in the MEASURES clause**

This tutorial will help you understand why you might get errors such as ORA-904 "%s: invalid identifier" or ORA-918 "column ambiguously defined" when you try to run a

SQL Analytics / Public

**MATCH\_RECOGNIZE - SKIP TO where exactly?**

We use the AFTER MATCH SKIP clause to determine the precise point to resume row pattern matching after a non-empty match is found. If you don't supply an AFTER MATCH SKIP clause then

Analytics / Public

**MATCH\_RECOGNIZE - Fraud demo for OracleCODE events**

This is a simple demo showing how to use SQL pattern matching for fraud analysis. It is part of a presentation for the OracleCODE events program for developers.

SQL Analytics / Public

**MATCH\_RECOGNIZE - Using Built-In Measures**

In this tutorial we will review the two built-in measures that are part of MATCH\_RECOGNIZE. These measures are designed to help you understand how your data set is mapped to the pattern

Analytics / Public

**Incorrect CLASSIFIER error**

This tutorial shows incorrect error message when using CLASSIFIER function with FIRST/LAST functions

SQL General / Private

**Managing very long lists in 12.2 with LISTAGG**

The tutorial will help explain the new 12.2 syntax for LISTAGG which provides developers with additional keywords for managing very long lists (i.e. those that exceed the current 4K

SQL General / Private

**New 12.2 Data Validation Features - CAST and VALIDATE\_CONVERSION**

The enhanced CAST function (along with TO\_NUMBER, TO\_BINARY\_FLOAT, TO\_BINARY\_DOUBLE, TO\_DATE, TO\_TIMESTAMP, TO\_TIMESTAMP\_TZ, TO\_DSINTERVAL, and

SQL General / Public

An aerial photograph of the San Francisco-Oakland Bay Bridge, showing its suspension towers and roadway over the water. The background features a dense urban skyline with various buildings and a park area with palm trees.

What is MATCH\_RECOGNIZE

# Pattern Recognition In Sequences of Rows

SQL - a new language for pattern matching

Provide native SQL language construct

With intuitive processing

# Pattern Recognition In Sequences of Rows

SQL - a new language for pattern matching

**Provide native SQL language construct**

- New SQL construct `MATCH_RECOGNIZE`
  - Added as part of the ANSI-2016 SQL standard

**With intuitive processing**

# Pattern Recognition In Sequences of Rows

SQL - a new language for pattern matching

## Provide native SQL language construct

- New SQL construct MATCH\_RECOGNIZE
  - Added as part of the ANSI-2016 SQL standard

## With intuitive processing

- Four logical concepts:
  - Logically partition and order the data
  - Define pattern using regular expression and pattern variables
  - Regular expression is matched against a sequence of rows
  - Each pattern variable is defined using conditions on rows and aggregates

# SQL MATCH\_RECOGNIZE

## “Declarative” pattern matching - 4 simple steps

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)
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)
  - Define matching using Boolean conditions on rows

```
Current time - INTERVAL '10' second) >= previous time
```

# SQL MATCH\_RECOGNIZE

## “Declarative” pattern matching - 4 simple steps

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

- MEASURES
  - . . . Session\_id
  - . . . Number of events
  - . . . Start time
  - . . . End time
  - . . . Duration

4. Determine how the output will be generated

An aerial photograph of the San Francisco-Oakland Bay Bridge, showing its massive steel towers and suspension cables. The bridge spans across a wide body of water, with a dense urban skyline of various skyscrapers and buildings visible in the background. The scene is captured from a high angle, providing a clear view of the bridge's structure and the surrounding city.

# Use Case 1: Sessionization

## Use Case 1: Sessionization

### Web Sessionization



Analyze online customer sessions by identifying each session within a series of clicks and then track user activity that typically involves multiple events

New SQL construct: **MATCH\_RECOGNIZE**  
Define patterns using regular expression syntax

Supports a wide range of use cases

# Source Data Set: JSON Key-Value Pairs Log File

Store log file data as a JSON document

```
CREATE TABLE json_sessionization  
(session_doc CLOB,  
CONSTRAINT "VALID_JSON" CHECK  
(session_doc IS JSON) ENABLE
```

```
SELECT  
  TO_NUMBER(j.session_doc.time_id) as time_id,  
  j.session_doc.user_id as user_id  
FROM json_sessionization j;
```

```
1  {{  
2  "time_id": "1",  
3  "user_id": "Mary"  
4  }  
5  {  
6  "time_id": "1",  
7  "user_id": "Sam"  
8  }  
9  {  
10 "time_id": "11",  
11 "user_id": "Mary"  
12 }  
13 {  
14 "time_id": "12",  
15 "user_id": "Sam"  
16 }  
17 {  
18 "time_id": "22",  
19 "user_id": "Sam"  
20 }  
21 {  
22 "time_id": "23",  
23 "user_id": "Mary"  
24 }  
25 {  
26 "time_id": "32",  
27 "user_id": "Sam"  
28 }  
29 {  
30 "time_id": "34",  
31 "user_id": "Mary"  
32 }
```



TIME_ID	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

# Use Case 1: Sessionization

Define a **session** as a sequence of one or more **events** within the same partition key **where** the inter-timestamp gap is less than a specified threshold

TIME_ID	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

1. Number sessions per user



TIME_ID	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. Aggregate analysis to provide deeper insight



USER_ID	SESSION ID	START TIME	END TIME	NUM EVENTS	DURATION
Mary	1	1	11	2	10
Mary	2	23	23	1	0
Mary	3	34	63	4	29

# Use Case 1: Sessionization

New syntax for discovering patterns using SQL:

**MATCH\_RECOGNIZE ( )**

```
SELECT *  
FROM . . . MATCH_RECOGNIZE  
(  
    . . .  
)
```

# Defining PARTITION BY and ORDER BY Clauses

Find distinct user sessions in a weblog:

**Step 1:** define partitions/ buckets and ordering needed to identify the “stream of events” ...

Set the PARTITION BY and ORDER BY clauses

```
SELECT *  
FROM . . . MATCH_RECOGNIZE  
(  
    PARTITION BY user_id ORDER BY time_id  
  
    . . .  
)
```

# Defining Pattern Statement

**Step 2:** define the pattern of events and pattern variables identifying the individual events within the pattern

Define the **pattern** – identify each “session”

```
SELECT *  
FROM . . . MATCH_RECOGNIZE  
(  
    PARTITION BY user_id ORDER BY time_id  
  
    PATTERN (b s+)  
  
    . . .  
)
```

## Build 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 ?

## Define Pattern Variables...

Define the **pattern variables**  
– specify each variable listed  
in the pattern

*a session is a sequence of  
one or more events within  
the same partition key  
where the inter-timestamp  
gap is less than a 10  
seconds*

```
SELECT *
FROM . . . MATCH_RECOGNIZE
(
    PARTITION BY user_id ORDER BY time_id

    PATTERN (b s+)
    DEFINE
        s as (time_id - prev(time_id)) <=10
        . . .
)
```

# Listing Pattern Measures to be Computed

Step 3: define the measures: source data points, pattern data points and aggregates related to a pattern:

MATCH\_NUMBER()

COUNT(): number of events

FIRST: start time

LAST: end time

```
SELECT *
FROM . . . MATCH_RECOGNIZE
(
  PARTITION BY user_id ORDER BY time_id
  MEASURES user_id,
            match_number() session_id,
            count(*) as no_of_events,
            first(b.time_id) start_time,
            last(s.time_id) end_time,
            last(s.time_id) - first(b.time_id) duration

  PATTERN (b s+)
  DEFINE
    s as (time_id - PREV(time_id)) <=10
    . . .
)
```

# Defining Output Style: Summary vs. Detailed

**Step 4:** determine how the output will be generated

Output **ONE ROW** for each time we find a match to our pattern

```
SELECT *
FROM . . . MATCH_RECOGNIZE
(
    PARTITION BY user_id ORDER BY time_id
    MEASURES user_id
             match_number() session_id,
             count(*) as no_of_events,
             first(time_id) start_time,
             last(s.time_id) end_time,
             last(time_id) - first(time_id) duration
    ONE ROW PER MATCH
    PATTERN (b s+)
    DEFINE
        s as (time_id - PREV(time_id)) <=10
        . . .
)
```

# Pattern Output Options

## Controlling the output

- Which rows to return

- ONE ROW PER MATCH
- ALL ROWS PER MATCH
- ALL ROWS PER MATCH WITH UNMATCHED ROWS

- After match SKIP option :

- SKIP PAST LAST ROW
- SKIP TO NEXT ROW
- SKIP TO <VARIABLE>
- SKIP TO FIRST (<VARIABLE>)
- SKIP TO LAST (<VARIABLE>)

# Live demonstration – Sessionization Tutorial


**ORACLE** Live SQL

Sessionization with MATCH\_RECOGNIZE and JSON View All Tutorials Login and Run Tutorial

Tutorial	Sessionization with MATCH_RECOGNIZE and JSON
Description	How to use new 12c SQL pattern matching match_recognize feature for sessionization analysis based on JSON web log files
Tags	pattern matching, sessionization, MATCH_RECOGNIZE, JSON
Category	SQL Analytics
Contributor	Oracle
Created	Monday April 24, 2017
Modules	13

Module 1	<b>Overview of SQL Pattern Matching</b> <b>Finding patterns with SQL</b> <p>Recognizing patterns in a sequence of rows has been a capability that was widely desired, but not really possible with SQL until now. There were many workarounds, but these were difficult to write, hard to understand, and inefficient to execute.</p> <p>With Oracle Database 12c you can use the MATCH_RECOGNIZE clause to perform pattern matching in SQL to do the following:</p> <ol style="list-style-type: none"><li>1. Logically group and order the data that is used in the MATCH_RECOGNIZE clause using the PARTITION BY and ORDER BY clauses.</li><li>2. Define business rules/patterns using the PATTERN clause. These patterns use regular expressions syntax, a powerful and expressive feature and applied to the pattern variables.</li><li>3. Specify the logical conditions required to map a row to a row pattern variable using the DEFINE clause.</li><li>4. Define output measures, which are expressions within the MEASURES clause.</li><li>5. Control the output (summary vs. detailed) from the pattern matching process.</li></ol>
Module 2	<b>Business Problem</b> <b>Sessionization analysis</b> <p>This tutorial is designed to help show you how you can run sessionization analysis on application logs, web logs, etc. that are in JSON format.</p> <p>For this tutorial we are going to define a session as a sequence of one or more events where the inter-timestamp gap is less than 10 minutes between events.</p> <p>Using our simplified JSON table of click data, which will create in the next step, we will create a sessionization data set which tracks each session, the duration of the session and the number of clicks/events using the new 12c MATCH_RECOGNIZE feature.</p>

[https://livesql.oracle.com/apex/livesql/file/tutorial\\_EWB8G5JBESHAGM9FB2GL4V5CAQ.html](https://livesql.oracle.com/apex/livesql/file/tutorial_EWB8G5JBESHAGM9FB2GL4V5CAQ.html)

An aerial photograph of a suspension bridge, likely the San Francisco-Oakland Bay Bridge, spanning across a body of water. The bridge's towers and cables are prominent, and the surrounding cityscape with various buildings and a park is visible in the background. The text 'Use Case 2: Controlling String Concatenation' is overlaid on the left side of the image.

## Use Case 2: Controlling String Concatenation

# Employee Data Set

Transform EMP table to ...

DEPTNO	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
10	7782	CLARK	MANAGER	7839	09-JUN-81	2450	-
10	7839	KING	PRESIDENT	-	17-NOV-81	5000	-
10	7934	MILLER	CLERK	7782	23-JAN-82	1300	-
20	7369	SMITH	CLERK	7902	17-DEC-80	800	-
20	7566	JONES	MANAGER	7839	02-APR-81	2975	-
20	7788	SCOTT	ANALYST	7566	19-APR-87	3000	-
20	7876	ADAMS	CLERK	7788	23-MAY-87	1100	-
20	7902	FORD	ANALYST	7566	03-DEC-81	3000	-
30	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
30	7521	WARD	SALESMAN	7698	22-FEB-81	1250	500
30	7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400
30	7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-
30	7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0
30	7900	JAMES	CLERK	7698	03-DEC-81	950	-

Table listing employees in each Dept.

DEPTNO	NAMELIST
10	CLARK ; KING
10	MILLER
20	SMITH ; JONES
20	SCOTT ; ADAMS
20	FORD
30	ALLEN ; WARD
30	MARTIN ; BLAKE
30	TURNER ; JAMES

## Partitioning and Ordering the source data

```
SELECT *  
FROM scott.emp  
MATCH_RECOGNIZE(  
    PARTITION BY deptno ORDER BY empno
```

# Create PATTERN Statement and DEFINE Pattern Variables

Add PATTERN statement and DEFINE pattern variables

```
SELECT *  
FROM scott.emp  
MATCH_RECOGNIZE(  
    PARTITION BY deptno ORDER BY empno
```

```
PATTERN (s b*)  
DEFINE  
    b AS LENGTHB(S.ename) +  
    SUM(LENGTHB(CONCAT(B.ename, ';' )))  
    + LENGTHB(';') < = 15
```

## List Measures to be Calculated

Use built-in measure MATCH\_NUMBER() to return a grouping ID

```
SELECT *  
FROM scott.emp  
MATCH_RECOGNIZE(  
    PARTITION BY deptno ORDER BY empno  
    MEASURES match_number() AS mno
```

```
    PATTERN (S B*)  
    DEFINE  
        B AS LENGTHB(S.ename) +  
            SUM(LENGTHB(CONCAT(B.ename, ';' )))  
        + LENGTHB(';') < = 15
```

## Define Type of Output: Detailed vs. Summary

**Return detailed report** – returns one row for each successful match of pattern

```
SELECT *  
FROM scott.emp  
MATCH_RECOGNIZE(  
    PARTITION BY deptno ORDER BY empno  
    MEASURES match number() AS mno
```

**ALL ROWS PER MATCH**

```
PATTERN (S B*)  
DEFINE  
    B AS LENGTHB(S.ename) +  
    SUM(LENGTHB(CONCAT(B.ename, ';' )))  
    + LENGTHB(';') < = 15
```

## Define Where To Resume Searching

Using default SKIP TO... behaviour to control where to start searching for next pattern

```
SELECT *
FROM scott.emp
MATCH_RECOGNIZE(
  PARTITION BY deptno ORDER BY empno
  MEASURES match_number() AS mno
  ALL ROWS PER MATCH
  AFTER MATCH SKIP PAST LAST ROW
  PATTERN (S B*)
  DEFINE
    B AS LENGTHB(S.ename) +
    SUM(LENGTHB(CONCAT(B.ename, ';' )))
    + LENGTHB(';') < = 15
```

## SKIP TO - basic syntax

- **AFTER MATCH SKIP TO NEXT ROW**
  - Resume pattern matching at the row after the first row of the current match.
- **AFTER MATCH SKIP PAST LAST ROW [DEFAULT]**
  - Resume pattern matching at the next row after the last row of the current match.
- **AFTER MATCH SKIP TO FIRST *pattern\_variable***
  - Resume pattern matching at the first row that is mapped to the pattern variable.
- **AFTER MATCH SKIP TO LAST *pattern\_variable***
  - Resume pattern matching at the last row that is mapped to the pattern variable.
- **AFTER MATCH SKIP TO *pattern\_variable***
  - The same as AFTER MATCH SKIP TO LAST *pattern\_variable*.

## Final Output from MATCH\_RECOGNIZE

DEPTNO	EMPNO	MNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
10	7782	1	CLARK	MANAGER	7839	09-JUN-81	2450	-
10	7839	1	KING	PRESIDENT	-	17-NOV-81	5000	-
10	7934	2	MILLER	CLERK	7782	23-JAN-82	1300	-
20	7369	1	SMITH	CLERK	7902	17-DEC-80	800	-
20	7566	1	JONES	MANAGER	7839	02-APR-81	2975	-
20	7788	2	SCOTT	ANALYST	7566	19-APR-87	3000	-
20	7876	2	ADAMS	CLERK	7788	23-MAY-87	1100	-
20	7902	3	FORD	ANALYST	7566	03-DEC-81	3000	-
30	7499	1	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
30	7521	1	WARD	SALESMAN	7698	22-FEB-81	1250	500
30	7654	2	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400
30	7698	2	BLAKE	MANAGER	7839	01-MAY-81	2850	-
30	7844	3	TURNER	SALESMAN	7698	08-SEP-81	1500	0
30	7900	3	JAMES	CLERK	7698	03-DEC-81	950	-

# Final Output from MATCH\_RECOGNIZE

Output from MATCH\_NUMBER part of final grouping within LISTAGG

DEPTNO	EMPNO	MNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
10	7782	1	CLARK	MANAGER	7839	09-JUN-81	2450	-
10	7839	1	KING	PRESIDENT	-	17-NOV-81	5000	-
10	7934	2	MILLER	CLERK	7782	23-JAN-82	1300	-
20	7369	1	SMITH	CLERK	7902	17-DEC-80	800	-
20	7566	1	JONES	MANAGER	7839	02-APR-81	2975	-
20	7788	2	SCOTT	ANALYST	7566	19-APR-87	3000	-
20	7876	2	ADAMS	CLERK	7788	23-MAY-87	1100	-
20	7902	3	FORD	ANALYST	7566	03-DEC-81	3000	-
30	7499	1	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
30	7521	1	WARD	SALESMAN	7698	22-FEB-81	1250	500
30	7654	2	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400
30	7698	2	BLAKE	MANAGER	7839	01-MAY-81	2850	-
30	7844	3	TURNER	SALESMAN	7698	08-SEP-81	1500	0
30	7900	3	JAMES	CLERK	7698	03-DEC-81	950	-

# Controlling String Concatenation

**LISTAGG** - returns list of concatenated strings arranged as groups within each DEPTNO

DEPTNO	NAMELIST
10	CLARK;KING
10	MILLER
20	SMITH;JONES
20	SCOTT;ADAMS
20	FORD
30	ALLEN;WARD
30	MARTIN;BLAKE
30	TURNER;JAMES

```
SELECT
  deptno,
  LISTAGG(ename, ';')
    WITHIN GROUP (ORDER BY empno) AS namelist,
FROM emp_mr
GROUP BY deptno, mno;
```

# Controlling String Concatenation

**LISTAGG** - returns list of concatenated strings arranged as groups within each DEPTNO

DEPTNO	NAMELIST	HOW_LONG
10	CLARK;KING	10
10	MILLER	6
20	SMITH;JONES	11
20	SCOTT;ADAMS	11
20	FORD	4
30	ALLEN;WARD	10
30	MARTIN;BLAKE	12
30	TURNER;JAMES	12

```
SELECT
  deptno,
  LISTAGG(ename, ';')
    WITHIN GROUP (ORDER BY empno) AS namelist,
  LENGTH(LISTAGG(ename, ';')
    WITHIN GROUP (ORDER BY empno)) AS how_long
FROM emp_mr
GROUP BY deptno, mno;
```

# Live Demonstration – MATCH\_RECOGNIZE and LISTAGG

The screenshot shows the Oracle Live SQL interface. The top navigation bar includes 'ORACLE Live SQL', 'Feedback', 'Help', and the user 'keith.laker@oracle.com'. The left sidebar contains navigation options: Home, SQL Worksheet, My Session, Schema, Design, My Scripts, My Tutorials (selected), and Code Library. The main content area displays a tutorial titled 'Managing strings with MATCH\_RECOGNIZE' under the 'Tutorial' section. The tutorial details include:
 

- Tutorial Name: Managing strings with MATCH\_RECOGNIZE
- Area: SQL Analytics
- Published: Yes
- Overview: This tutorial is linked to one of my blog posts "Managing overflows in LISTAGG" - https://oracle-big-data.blogspot.co.uk/2015/03/managing-overflows-in-listagg.html
- Modules: 5
- Share Link: https://livesql.oracle.com/apex/livesql/file/tutorial\_EWCF5RFYTP2OAXEFX4IXKHOC.html
- Tags: -
- Last Updated: 44 hours ago (Created 45 hours ago)

 Below the tutorial details is a 'Modules' table with the following data:
 


Module	Sequence	Content
Overview	10	<p>This is an interesting problem that has come up a few times in discussions (and I think it has been mentioned on the SQL forums as well). When using LISTAGG on very large data sets you can sometimes create a list that is too long and consequently get ...
Our test data	20	<p>For example, let's assume that we have the following statement (to keep things relatively simple let's use the EMP table in the schema SCOTT)</p><code>SELECT deptno, LISTAGG(name, ',') WITHIN GROUP (ORDER BY empno) AS namelist FROM scott.emp G...
Chopping up the list of values	30	<p>We can use the Database 12c SQL pattern matching function, MATCH_RECOGNIZE, to return a list of values that does not exceed 15 characters. First step is to wrap the processing in a view so that we can then get data from this view to feed our LISTAGG fu...
New string groupings	40	<p>You might well ask: why don't we put the LISTAGG function inside the measure clause? At the moment it is not possible to include analytical functions such as LISTAGG in the measure clause. Therefore, we have put the LISTAGG function in a separate SQL s...
Changing the cut-off point	50	<p>You can change the cut-off point for the string truncation process by changing the value shown in bold on the last line of the code...<em>in this version I have changed the truncation point to 25 characters</em></p><code>CREATE OR REPLACE VIEW emp_m...

 At the bottom of the main content area is a 'Links' table:
 

Link Name	Sequence	Link Target	Updated

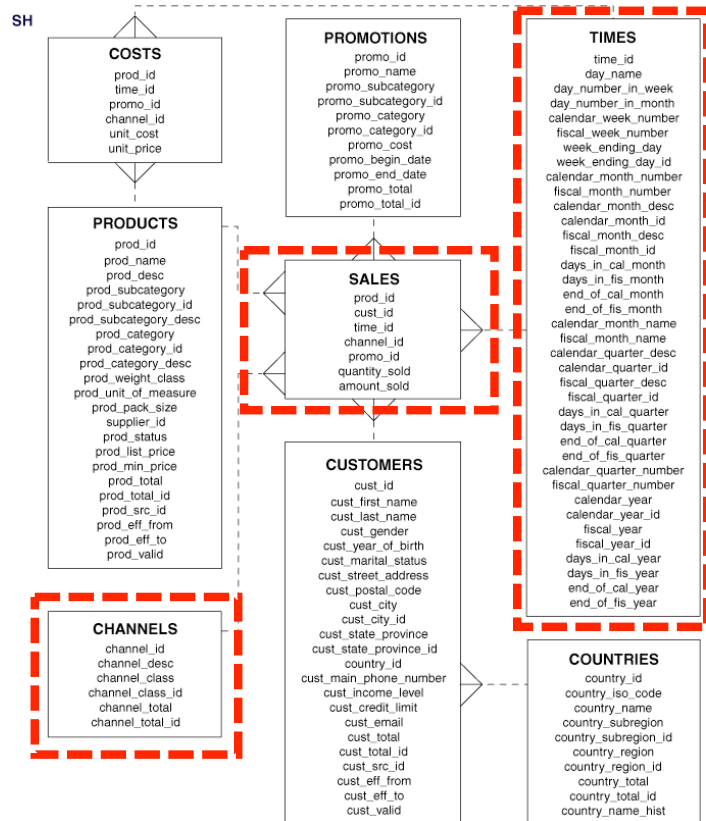
 On the right side of the interface, there is a vertical toolbar with buttons: Run, Edit, Add Module, Reorder Modules, Add Link, and Extract Code.

[https://livesql.oracle.com/apex/livesql/file/tutorial\\_EWCF5RFYTP2OAXEFX4IXKHOC.html](https://livesql.oracle.com/apex/livesql/file/tutorial_EWCF5RFYTP2OAXEFX4IXKHOC.html)

An aerial photograph of a suspension bridge, likely the San Francisco-Oakland Bay Bridge, spanning across a body of water. The bridge features a prominent white tower with a lattice structure. The background shows a dense urban skyline with various buildings and a green park area near the water's edge. The text "Use Case 3: Building Contiguous Date Ranges" is overlaid on the left side of the image.

**Use Case 3:  
Building  
Contiguous  
Date Ranges**

# Data Set



Return result set showing:

- 1) Start date of contiguous range
- 2) End of date of contiguous range
- 3) Number of days in contiguous range

## Define SELECT statement for source data

```
SELECT start_day, end_day, count_day  
FROM
```

```
(SELECT  
  DISTINCT s.time_id AS day_id,  
  t.calendar_year AS cal_yr  
FROM sh.sales s, sh.times t  
WHERE channel_id = 4 AND t.time_id= s.time_id)
```

## Partitioning and Ordering the source data

```
SELECT start_day, end_day, count_day
FROM
(SELECT
  DISTINCT s.time_id AS day_id,
  t.calendar_year AS cal_yr
FROM sh.sales s, sh.times t
WHERE channel_id = 4 AND t.time_id= s.time_id)
MATCH_RECOGNIZE(
  PARTITION BY cal_yr ORDER BY day_id
```

## Create PATTERN Statement and DEFINE Pattern Variables

Add PATTERN statement – *includes ALWAYS TRUE variable* - DEFINE pattern variables

```
SELECT start_day, end_day, count_day
FROM
(SELECT
  DISTINCT s.time_id AS day_id,
  t.calendar_year AS cal_yr
FROM sh.sales s, sh.times t
WHERE channel_id = 4 AND t.time_id= s.time_id)
MATCH_RECOGNIZE(
  PARTITION BY cal_yr ORDER BY day_id
```

```
PATTERN (strt a+)
DEFINE a AS day_id = PREV(day_id)+1);
```

## List Measures to be Calculated

Use new functions **FIRST** and **LAST** to return values from start and end of pattern

```
SELECT start_day, end_day, count_day
FROM
(SELECT
  DISTINCT s.time_id AS day_id,
  t.calendar_year AS cal_yr
FROM sh.sales s, sh.times t
WHERE channel_id = 4 AND t.time_id= s.time_id)
MATCH_RECOGNIZE(
  PARTITION BY cal_yr ORDER BY day_id
  MEASURES
    FIRST(strt.day_id) AS start_day,
    LAST(a.day_id) AS end_day,
    COUNT(day_id) AS count_day

  PATTERN (strt a+)
  DEFINE a AS day_id = PREV(day_id)+1);
```

## Define Type of Output: Detailed vs. Summary

**Return summary report – returns one row for each successful match of pattern**

```
SELECT start_day, end_day, count_day
FROM
(SELECT
  DISTINCT s.time_id AS day_id,
  t.calendar_year AS cal_yr
FROM sh.sales s, sh.times t
WHERE channel_id = 4 AND t.time_id= s.time_id)
MATCH_RECOGNIZE(
  PARTITION BY cal_yr ORDER BY day_id
  MEASURES
    FIRST(strt.day_id) AS start_day,
    LAST(a.day_id) AS end_day,
    COUNT(day_id) AS count_day
  ONE ROW PER MATCH
  PATTERN (strt a+)
  DEFINE a AS day_id = PREV(day_id)+1);
```

## Finding Contiguous Date Ranges...

<b>START_DAY</b>	<b>END_DAY</b>	<b>COUNT_DAY</b>
01-JAN-98	28-FEB-98	59
02-MAR-98	06-MAR-98	5
08-MAR-98	01-APR-98	25
04-APR-98	06-APR-98	3
08-APR-98	11-APR-98	4
13-APR-98	18-APR-98	6
22-APR-98	06-MAY-98	15
08-MAY-98	12-MAY-98	5
14-MAY-98	24-MAY-98	11
26-MAY-98	06-JUN-98	12
08-JUN-98	11-JUN-98	4
13-JUN-98	18-JUN-98	6

# Live demonstration – Finding Contiguous Date Ranges

The screenshot shows the Oracle Live SQL interface. The main content area displays a script titled "Finding contiguous date ranges". The script's description is "Using SQL pattern matching to find contiguous date ranges. Uses the sales history (SH) schema: SALES fact table, TIMES dimension table and CHANNELS dimension table." The script results are shown as a table with two columns: DAY\_ID and CAL\_YR. The results are as follows:

DAY_ID	CAL_YR
02-JAN-98	1998
08-JAN-98	1998
09-JAN-98	1998
17-JAN-98	1998
27-JAN-98	1998
31-JAN-98	1998
02-FEB-98	1998

The script code is as follows:

```
SELECT
DISTINCT s.time_id AS day_id,
t.calendar_year AS cal_yr
FROM sh.sales s, sh.times t
WHERE t.calendar_year = '1998'
AND s.time_id = t.time_id
AND s.channel_id = 4
```

A callout box explains: "This statement returns all the days on which sales were made through the internet channel (d=4) during 1998. You should get 348 rows returned."

[https://livesql.oracle.com/apex/livesql/file/content\\_EWNZJ82L6J0JSVINVL2GAV7DC.html](https://livesql.oracle.com/apex/livesql/file/content_EWNZJ82L6J0JSVINVL2GAV7DC.html)

# Summary



# Typical Pattern Matching LOB Use Cases

	Input Data	Pattern	Result
Sessionization	Weblogs	continuous clicks by same user	Generate reports on number of distinct sessions, average page views per session, etc
Fraud	Credit card transactions	two transactions in different locations within a short period of time	Find cases in which a credit card may have been used fraudulently since a physical person cannot be in two places at once
In-game purchases	Games logs	events leading up to an in-game purchase	Detect common sequences of event that results in an in-game purchase
Fraud (mobiles)	CDR logs	SIM card being used in multiple handsets	Flag individual SIM cards being used by multiple handsets within a specified time period
Stock market analysis	Ticker logs	Track possible fraudulent linked patterns of behavior	Track known patterns of behavior such as head and shoulders, triangles, channels and wedges

# Typical Pattern Matching LOB Use Cases

	Input Data	Pattern	Result
Auditing/ Compliance	Application logs	Analyze changes to secure customer data	Find instances where operator has made suspect modifications to secure client data
Money laundering	Transaction logs	Search for small transfers within a time window following by large transfer within "x" days of last small transfer	Detect suspicious money transfer pattern for an account and report account, date of first small transfer, date of last large transfer
Call service quality	CDR logs	Search for dropped/reconnected calls	Identify how many times calls were restarted in a session, total effective call duration and total interrupted duration

## Summary – You Can Now....

- ✔ Construct a MATCH\_RECOGNIZE statement
  - ✔ Build search criteria using pattern variables
  - ✔ Organize your data correctly to discover the pattern
  - ✔ Control the type of data returned: summary vs. detailed
  - ✔ Understand the power and value of SQL pattern matching
- ✔ Go and use MATCH-RECOGNIZE to your advantage!**

## Where To Get More Information

- [Analytical SQL Home Page](#) on OTN with links to:
  - Training + Oracle By Example
  - Podcasts for pattern matching
  - Whitepapers
  - Sample scripts and simple tutorials for pattern matching on liveSQL
- Data Warehouse and SQL Analytics blog
  - <http://oracle-big-data.blogspot.co.uk/>



ORACLE®

