
ORACLE CODE
developer.oracle.com/code
Simplified and fast Fraud Detection
Live for the Code
ORACLE

ORACLE CODE

developer.oracle.com/code

About me…

Keith Laker

Senior Principal Product Management

SQL and Data Warehousing

Marathon runner, mountain biker and coffee drinker

@ASQLBarista

Live for the Code

ORACLE

Agenda

1. Finding patterns in your data
2. Looking for fraudulent transfers
3. Dealing with new business requirements
4. Combining different types of analytics: Mash-Ups
5. Summary


ORACLE CODE
developer.oracle.com/code
Finding Patterns in Your Data
Live for the Code
ORACLE
Slide - 4

Finding Patterns In Your Data

Basic Concepts and Languages

- Lots of languages natively support “pattern matching”

A
- AWK

C
- COMIT

E
- Elixir (programming language)
- Elm (programming language)
- Erlang (programming language)

F
- F Sharp (programming language)

H
- Haskell (programming language)

I
- Icon (programming language)

L
- LFE (programming language)

M
- Metacompiler
- ML (programming language)

O
- OCaml

P
- Prolog

R
- Refal
- Rust (programming language)

S
- Scala (programming language)
- CSCM (programming language)
- Sed
- SNOBOL
- SPITBOL
- Swift (programming language)

T
- Tom (pattern matching language)

U
- Unicon (programming language)

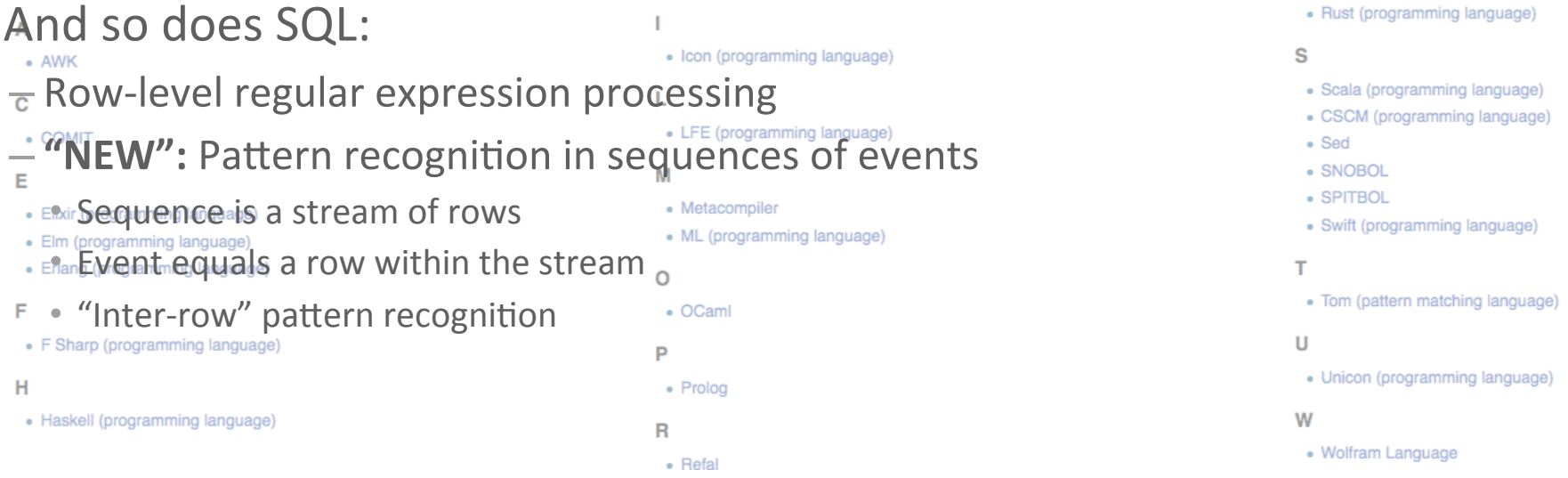
W
- Wolfram Language

Source: wikipedia

Finding Patterns In Your Data

Basic Concepts and Languages

- Lots of languages natively support “pattern matching”
- And so does SQL:
  - Row-level regular expression processing
  - **“NEW”:** Pattern recognition in sequences of events
    - Sequence is a stream of rows
    - Event equals a row within the stream
    - “Inter-row” pattern recognition



Source: wikipedia

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

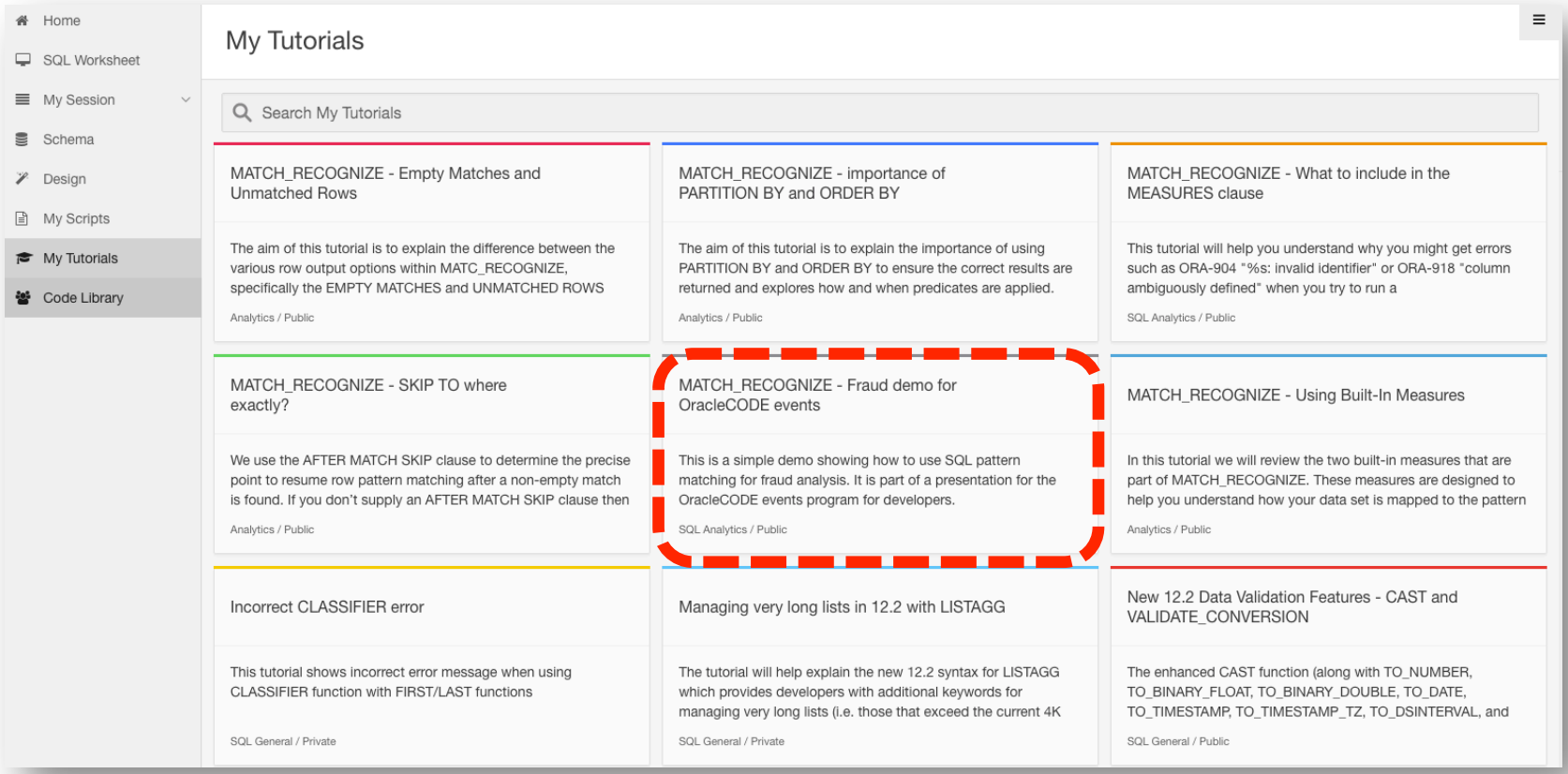
- New SQL construct MATCH_RECOGNIZE
  - Added as part of the ANSI-2016 SQL standard

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


ORACLE CODE
developer.oracle.com/code
Searching for Fraud
Live for the Code
ORACLE
Slide - 10

Demo is on http://livesql.oracle.com



Business Problem: Finding Suspicious Money Transfers

- Suspicious money transfer pattern for an account is:
  - 3 or more small (<2K) money transfers within 30 days
  - Large transfer (>=1M) within 10 days of last small transfer
- Report
  - account
  - date of first small transfer
  - date of last large transfer
  - amount of large transfer

Business Problem: Find Suspicious Money Transfers

- Data stored in JSON logs ..

```
[{
"time_id":"01-JAN-12",
"user_id":"John",
"event_id":"Deposit",
"trans_amount":1000000
},
{
"time_id":"02-JAN-12",
"user_id":"John",
"event_id":"Transfer",
"trans_amount":1000
},
{
"time_id":"05-JAN-12",
"user_id":"John",
"event_id":"Withdrawal",
"trans_amount":2000
},
{
"time_id":"05-JAN-12",
"user_id":"John",
"event_id":"Transfer",
"trans_amount":1500
}
```

Business Problem: Find Suspicious Money Transfers

- Data stored in JSON logs ..

```
[{
"time_id":"01-JAN-12",
"user_id":"John",
"event_id":"Deposit",
"trans_amount":1000000
},
{
"time_id":"02-JAN-12",
"user_id":"John",
"event_id":"Transfer",
"trans_amount":1000
},
{
"time_id":"05-JAN-12",
"user_id":"John",
"event_id":"Withdrawal",
"trans_amount":2000
},
{
"time_id":"05-JAN-12",
"user_id":"John",
"event_id":"Transfer",
"trans_amount":1500
}
```

.. processed as relational table structure

TIME_ID	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
2/2/2012	John	Deposit	500,000

Business Problem: Finding Suspicious Money Transfers

TIME_ID	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
2/2/2012	John	Deposit	500,000

Three small transfers within 30 days

Business Problem: Finding Suspicious Money Transfers

TIME_ID	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
2/2/2012	John	Deposit	500,000

Three small transfers within 30 days

Large transfer within 10 days of last small transfer

Declarative Pattern Matching using SQL

New syntax for discovering patterns using SQL: **finding suspicious money transfers**…

Input into the pattern matching process determined by **FROM** clause

MATCH_RECOGNIZE ( ) – indicates start of pattern matching definition

```
SELECT . . .
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
```

Declarative Pattern Matching using SQL

STEP 1

Need to group and order the data to make the pattern "*visible*" within the sequence of rows…

Set the **PARTITION BY** and **ORDER BY** clauses

```
SELECT . . .
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY user_id ORDER BY time_id
```

Declarative Pattern Matching using SQL

STEP 2

Define the **MEASURES** –

List the columns to be returned

- calculated columns
- existing columns

Report account, date of first small transfer, date of last large transfer

```
SELECT . . .
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY user_id ORDER BY time_id
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.time_id) AS last_t,
           y.amount amount
```

Declarative Pattern Matching using SQL

STEP 3

Control the output in terms of **ROW**s returned

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

```
SELECT . . .
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY userid ORDER BY time
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.time_ID) AS last_t,
           y.amount amount
  ONE ROW PER MATCH
```

Controlling the Output: Summary vs. Detailed

- 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>)

Declarative Pattern Matching using SQL

STEP 4

Describe how to search for the **VARIABLE X** –

Searching for three or more occurrences of **X**

```
SELECT . . .
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY user_id ORDER BY time_id
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.timeid) AS last_t,
           y.amount amount
  ONE ROW PER MATCH
  PATTERN ( X{3,} )
```

Using Regular Expressions To Control Matching

- 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 ?

Declarative Pattern Matching using SQL

STEP 4

Describe how to search for the **VARIABLE Y** –

Searching for only one instance of **Y**

```
SELECT . . .
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY user_id ORDER BY time_id
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.time_id) AS last_t,
           y.amount amount
  ONE ROW PER MATCH
  PATTERN ( X{3,} Y)
```

Declarative Pattern Matching using SQL

STEP 5

DEFINE each variable in terms of what to search for:

For a match on **variable X** we are searching for small transfers of amounts less than 2K and all three transfers must occur within a 30 day window

```
SELECT . . .
FROM
 (SELECT … FROM json_transactions WHERE event ='transfer')
MATCH_RECOGNIZE (
  PARTITION BY user_id ORDER BY time_id
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.time_id) AS last_t,
           y.amount amount
  ONE ROW PER MATCH
  PATTERN ( X{3,} Y)
  DEFINE
       X as (amount < 2000) AND
          LAST(time_id) - FIRST(time_id) < 30,
```

Declarative Pattern Matching using SQL

STEP 5

DEFINE each variable in terms of what to search for:

For a match on **variable Y** we are searching for a large transfer of more than 10K

```
SELECT . . .
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY user_id ORDER BY time_id
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.time_id) AS last_t,
           y.amount amount
  ONE ROW PER MATCH
  PATTERN ( X{3,} Y)
  DEFINE
       X as (amount < 2000) AND
          LAST(time_id) – FIRST(time_id) < 30,
       Y as (amount >= 1000000
```

Declarative Pattern Matching using SQL

STEP 5

DEFINE each variable in terms of what to search for:

For a match on **variable Y** we are searching for a large transfer of more than 10K…

…and the large transfer must occur within 10 days of last small transfer

```
SELECT . . .
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY user_id ORDER BY time_id
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.time_id) AS last_t,
           y.amount amount
  ONE ROW PER MATCH
  PATTERN ( X{3,} Y)
  DEFINE
       X as (amount < 2000) AND
          LAST(time_id) – FIRST(time_id) < 30,
       Y as (amount >= 1000000 AND
             time_id – LAST(x.time_id) < 10 ))
```

Declarative Pattern Matching using SQL

Finally list columns to return by the pattern matching process to the calling application

*SELECT * FROM… is acceptable but should really be avoided!*

```
SELECT user_id, first_t, last_t, amount
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY user_id ORDER BY time_id
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.time_id) AS last_t,
           y.amount AS amount
  ONE ROW PER MATCH
  PATTERN ( X{3,} Y)
  DEFINE
       X as (amount < 2000) AND
            LAST(time_id) – FIRST(time_id) < 30,
       Y as (amount >= 1000000 AND
             time_id – LAST(x.time_id) < 10 ))
```

Live demonstration – Simple Fraud Detection


ORACLE CODE
developer.oracle.com/code
New Requirements
Managing new business requirements, quickly and efficiently with no application code changes
Live for the Code
ORACLE
Slide - 30

Refining The “Suspicious Money Transfers” Pattern

Dealing With New Business Requirements

- **Additional requirement:**
  - Check for transfers are to different accounts
  - Total sum of small transfers must be less than 20K


```
[{
"time_id":"01-JAN-12",
"user_id":"John",
"event_id":"Deposit",
"trans_amount":1000000
},
{
"time_id":"02-JAN-12",
"user_id":"John",
"event_id":"Transfer",
"transfer_id":"John",
"trans_amount":100
},
{
"time_id":"02-JAN-12",
"user_id":"John",
"event_id":"Transfer",
"transfer_id":"Bob",
"trans_amount":1000
},
{
"time_id":"05-JAN-12",
"user_id":"John",
"event_id":"Withdrawal",
"trans_amount":2000
}
```

Business Problem: Find Suspicious Money Transfers

- Data stored in JSON logs ..
- .. processed as relational table structure

```
[{
"time_id":"01-JAN-12",
"user_id":"John",
"event_id":"Deposit",
"trans_amount":1000000
},
{
"time_id":"02-JAN-12",
"user_id":"John",
"event_id":"Transfer",
"transfer_id":"John",
"trans_amount":100
},
{
"time_id":"02-JAN-12",
"user_id":"John",
"event_id":"Transfer",
"transfer_id":"Bob",
"trans_amount":1000
},
{
"time_id":"05-JAN-12",
"user_id":"John",
"event_id":"Withdrawal",
"trans_amount":2000
}
```

TIME_ID	USER_ID	EVENT	TRANSFER_ID	AMOUNT
1/1/2012	John	Deposit		1,000,000
1/2/2012	John	Transfer	John	1,000
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
2/2/2012	John	Deposit		500,000

Refining The “Suspicious Money Transfers” Pattern

TIME_ID	USER ID	EVENT	TRANSFER_TO	AMOUNT
1/1/2012	John	Deposit	-	1,000,000
1/2/2012	John	Transfer	John	100
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
2/2/2012	John	Deposit	-	500,000

Refining The “Suspicious Money Transfers” Pattern

TIME_ID	USER ID	EVENT	TRANSFER_TO	AMOUNT
1/1/2012	John	Deposit	-	1,000,000
1/2/2012	John	Transfer	John	100
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
2/2/2012	John	Deposit	-	500,000

Refining The “Suspicious Money Transfers” Pattern

TIME_ID	USER ID	EVENT	TRANSFER_TO	AMOUNT
1/1/2012	John	Deposit	-	1,000,000
1/2/2012	John	Transfer	John	100
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
2/2/2012	John	Deposit	-	500,000

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

SQL Pattern Matching in action

Modify the pattern variables
DEFINE

- Check the transfer account

```
SELECT user_id, first_t, last_t, amount
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY userid ORDER BY time
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.time_id) AS last_t,
           y.amount AS amount
  ONE ROW PER MATCH
  PATTERN (X{3,} Y)
  DEFINE
       X as (amount < 2000) AND
             LAST(time_id) -  FIRST(time_id) < 30
```

Refining The “Suspicious Money Transfers” Pattern

TIME_ID	USER ID	EVENT	TRANSFER_TO	AMOUNT
1/1/2012	John	Deposit	-	1,000,000
1/2/2012	John	Transfer	John	100
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
2/2/2012	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 in action

Modify the pattern variables
DEFINE

- Check the transfer account

```
SELECT user_id, first_t, last_t, amount
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY userid ORDER BY time
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.time_id) AS last_t,
            y.amount AS amount
  ONE ROW PER MATCH
  PATTERN (X{3,} Y)
  DEFINE
       X as (amount < 2000) AND
             PREV(transfer_id) <> transfer_id AND
             LAST(time_id) – FIRST(time_id) < 30
```

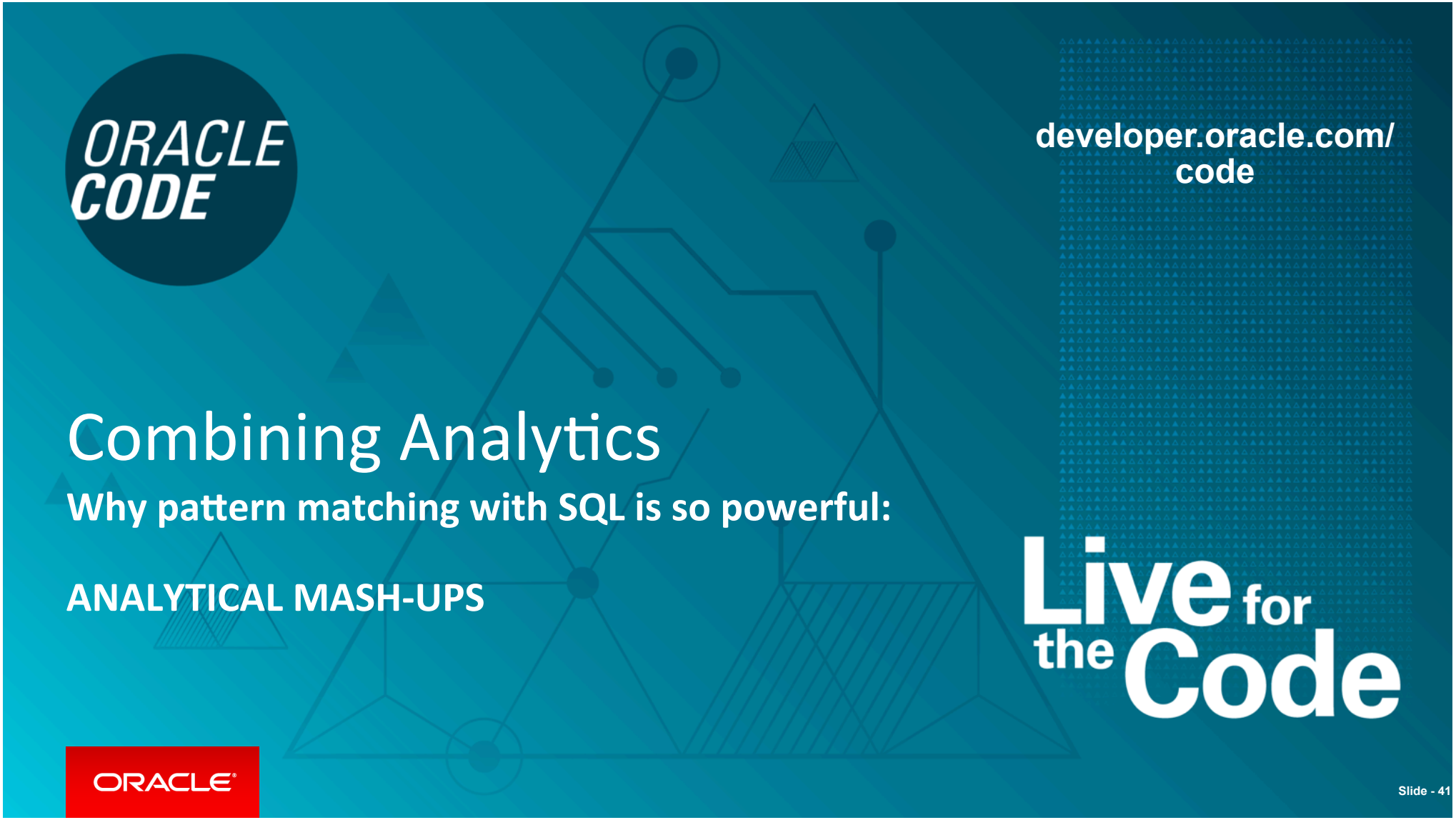
SQL Pattern Matching in action

Modify the pattern variables
DEFINE

- Check the total of the small transfers is less than 20K

```
SELECT user_id, first_t, last_t, amount
FROM
 (SELECT … FROM json_transactions WHERE event = 'transfer')
MATCH_RECOGNIZE (
  PARTITION BY userid ORDER BY time
  MEASURES FIRST(x.time_id) AS first_t,
           LAST(y.time_id) AS last_t,
           y.amount amount
  ONE ROW PER MATCH
  PATTERN (X{3,} Y)
  DEFINE
       X as (amount < 2000) AND
             PREV(transfer_id) <> transfer_id AND
             LAST(time_id) – FIRST(time_id) < 30,
       Y as (amount >= 1000000 AND
             time_id – LAST(x.time_id) < 10 AND
             SUM(x.amount) < 20000 ));
```

Live Demonstration – New Requirements


ORACLE CODE
developer.oracle.com/code
Combining Analytics
Why pattern matching with SQL is so powerful:
ANALYTICAL MASH-UPS
Live for the Code
ORACLE
Slide - 41

Fraud Detection Mash-Up Using SQL

Key benefits of SQL – easy to combine different types of analytics

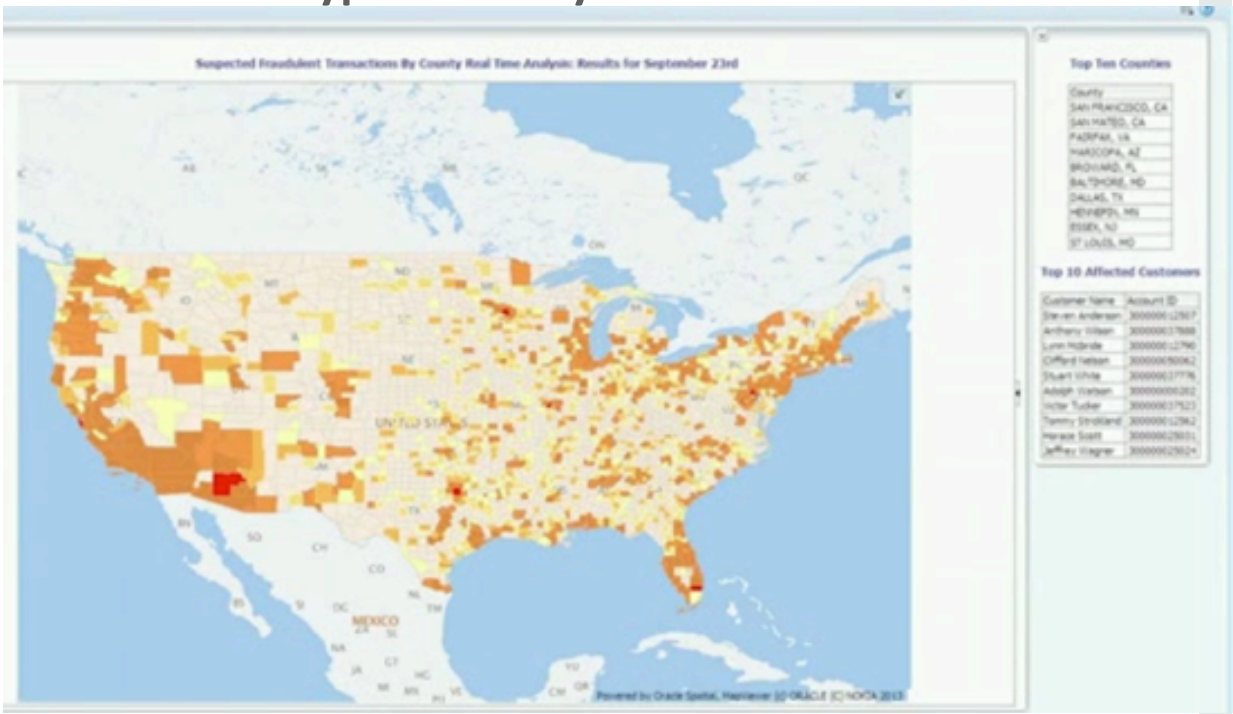
+Real-time streaming data

+Real-time geo-coding

+SQL Pattern Matching

+Spatial Analytics

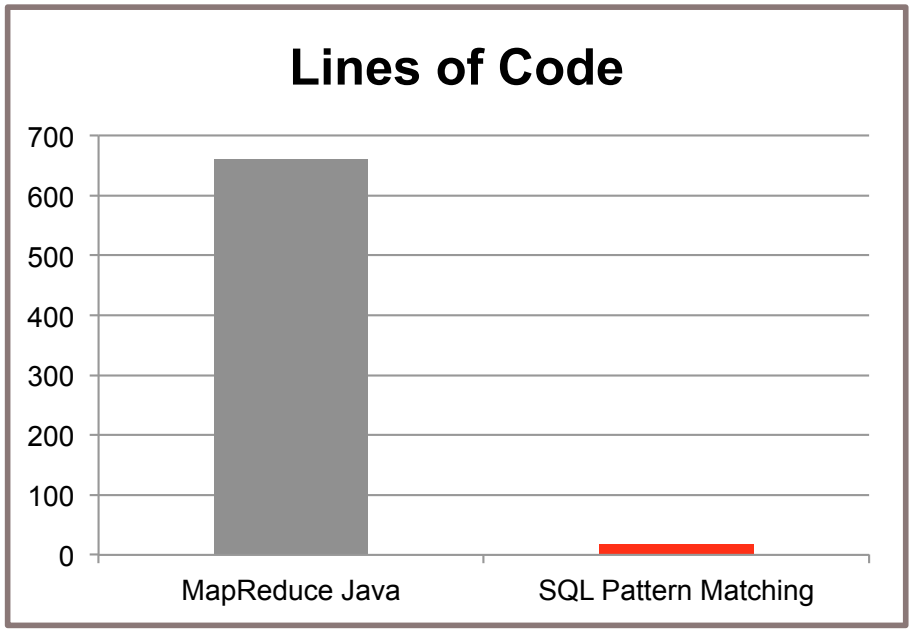
= Real-time fraud detection



http://oracle-big-data.blogspot.com/2013/11/sql-analytical-mash-ups-deliver-real.html

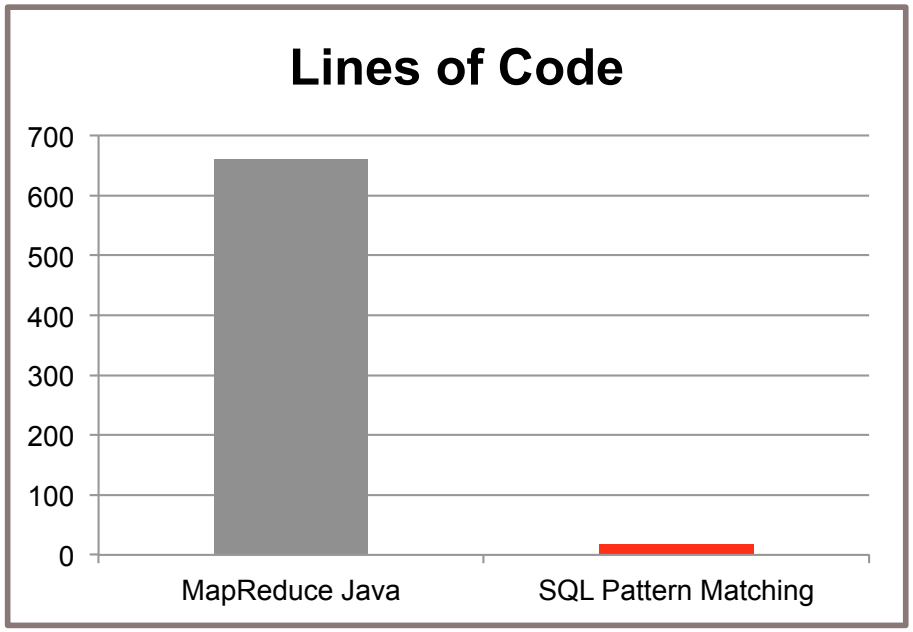
SQL Pattern Matching vs. Java

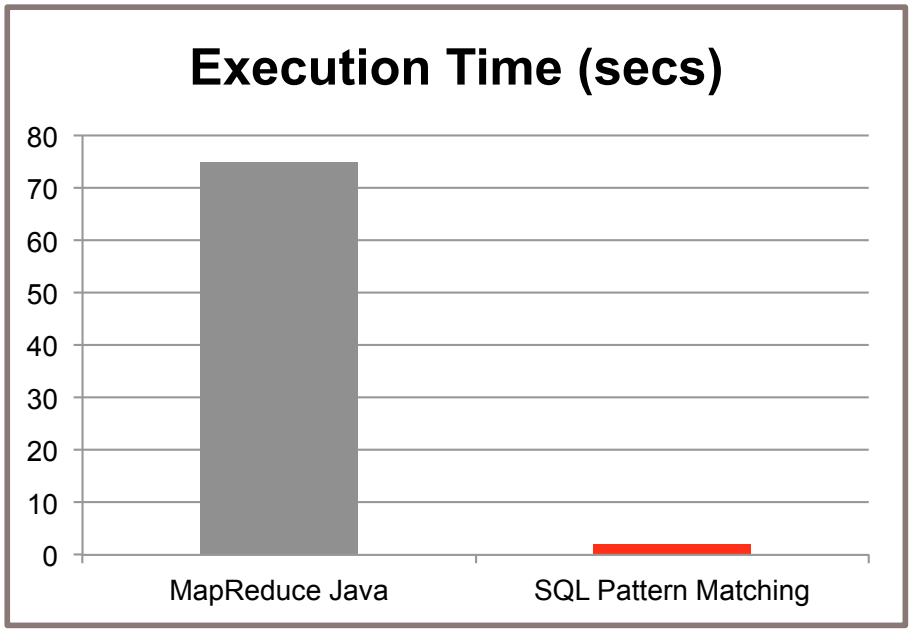
Fraud Detection: Fewer lines of code required….



SQL Pattern Matching vs. Java

Fraud Detection: Fewer lines of code required….






ORACLE
CODE
developer.oracle.com/
code
Summary
Live for
the Code
ORACLE

Summary – You Can Now….

- ✔ Construct a MATCH_RECOGNIZE statement
- ✔ Build search criteria using pattern variables
- ✔ Check if your pattern matching *really* is working correctly
- ✔ Use built-in measures to get additional information
- ✔ Understand the power and value of SQL pattern matching

- ✔ **Go and use SQL Pattern Matching to your advantage!**

SQL Pattern Matching

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/