Using Analytical SQL to Intelligently Explore Big Data

Next generation of SQL makes it easier to ask smarter questions of big data
Who Am I

Keith Laker
Senior Principal Product Manager, Analytical SQL
Oracle

Twitter @BigRedDW
Facebook BigRedDW
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.
Agenda

1. SQL and Big Data
2. Calculating approximate answers
3. Searching for patterns
4. Summary
SQL For Big Data
Conceptual View of Big Data Architecture

Event Engine
- Actionable Events

Data Reservoir
- Actionable Insights

Data Factory
- Enterprise Information Store
- Reporting

Discovery Lab
- Discovery Output

Other Data
- Structured Enterprise Data

Data Streams

Execution

Innovation
- Events & Data
Requires Common Analytical Language = SQL
Requires Common Analytical Language = **SQL**

**SQL**

**FASTER, RICHER, SIMPLER**
Part 1: Creating approximate result-sets
When good enough is in fact good enough
Getting *Approximate* Counts

Answer “*How many*...” type questions

– How many unique sessions today
– How many unique customers logged on
– How many unique events occurred

**COUNT (DISTINCT . . .)**

– returns the exact number of rows that contain distinct values of specified expression
– Can be resource intensive because requires sorting

... significantly faster solution is

`APPROX_COUNT_DISTINCT (expr)`

– processes large amounts of data significantly faster
– uses HyperLogLog algorithm
– negligible deviation from exact result
  • ignores rows containing null values
– supports any scalar data type
  • Does not support BFILE, BLOB, CLOB, LONG, LONG RAW, or NCLOB
Accuracy and Performance

**Results for accuracy**
- Real world customer workload
- Accuracy that is typically **97%** with **95% confidence**

**Performance Results**
- Real world customer workload
- **5-50x** improvement

**Notes:**
- this approach does not use sampling, it uses a hash-based approach
- ignores rows that contain a null value for specified expression
- Supports any scalar data type other than BFILE, BLOB, CLOB, LONG, LONG RAW, or NCLOB
Using SQL to Search for Patterns

Simpler code, faster results
Pattern Recognition In Sequences of Rows

SQL Pattern Matching - Concepts

• Business Challenge
  – 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
  – Pattern defined with regular expressions via variables
    • Regular expression matched against a sequence of rows (forwards/backwards)
    • Each pattern variable is defined using conditions on rows and aggregates
Making Code Simpler: Pattern Matching with SQL
Java vs. SQL: Searching for ‘W’ Patterns in Stock Trade Data

• Find a W-shape pattern in a ticker stream:
  • Output the **beginning** and **ending** date of the pattern
  • Calculate **average price** each the W-shape
  • Find only patterns that lasted less than a week

250+ Lines of Java and PIG
Making Code Simpler: Pattern Matching with SQL

Java vs. SQL: Searching for ‘W’ Patterns in Stock 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)),
    Z AS (price > PREV(price) AND
          z.time - FIRST(x.time) <= 7 ))
```

250+ Lines of Java and PIG

12 Lines of SQL
Making Code Simpler: Pattern Matching with SQL

Java vs. SQL: Searching for ‘W’ Patterns in Stock 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)),
Z 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
Summary
Safe Harbor Statement

The preceding 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.