Dream the Stream
High Velocity Event Processing with a Converged Database

Shasank Chavan
Vice President, In-Memory Data Technologies

University of Wisconsin, Madison
October 24, 2022
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, timing, and pricing of any features or functionality described for Oracle’s products may change and remains at the sole discretion of Oracle Corporation.
Agenda

What is Event Stream Processing

Converged vs. Specialized Databases

What does an Event Stream Processing Database Need

Demo Slides – DevOps Monitoring
What is Event Stream Processing
What is Event Stream Processing?

Continuous Ingestion
Continuous ingest of high frequency event data

Real-time Analytics
Unlike batch processing, event processing analytics is performed on data in motion

Data Reorganization
Event data is increasingly compressed and summarized as it ages before finally comes to rest as archived data
What is Event Data?

- Events are discrete data records generated by large farms of data sources
- Data sources are extremely diverse
  - Devices, sensors, meters, servers, desktops, smartphones
- An event typically includes the following information:

  **Time:** 6/16/21:12:05pm  
  **Meter ID:** X45-123  
  **Reading:**  
  - Electricity KW-hrs: 0.4  
  - Water Gallons: 5  
  - Gas therms: 2

  **Time:** 6/11/21:12:12pm  
  **Phone ID:** 1955ABC  
  **Reading:**  
  - Location: 37.6N/112.2W  
  - Battery Level: 60%

  **Time:** 6/17/21:1:00pm  
  **Vehicle ID:** WBG6108  
  **Reading:**  
  - Location: 37.6N/112.2W  
  - Speed: 66.7 mph  
  - Direction: 120.5 degrees
Properties of Event Data

**High Arrival Rate**
- Most event processing systems receive large numbers of events from many different sources
  - E.g. Billing systems receives millions of smart meter readings every few minutes.

**High Obsolescence Rate**
- Recent events are frequently queried for real time analytics while old events are used for historical reporting
- Events are often compressed and summarized at greater and greater levels of data and space reduction as they age
  - E.g. Per minute readings from smart meters converted to hourly summaries after a day and converted to daily summaries after a month
Requirements of an Event Stream Processing System

Flexible data model
Needs to handle heterogeneous event sources
For example, a new device type added to home network

High Speed Ingest
Must be able to sustain billions of events per day

Rich Analytic Query Capability
Requires advanced analytic functionality to filter, aggregate and summarize across moving windows of event stream data

Real-Time Analytics
Instantaneous reporting of timely actions on events
For example, detecting and reporting fraud, fire, leaks, etc.

Automatic Data Lifecycle Management
Automatic data compression, summarization, archiving needed to avoid unbounded data growth
Converged vs. Specialized DBs
The Increasing Complexity of Modern Apps

- Modern Apps use a **new generation of data technologies**:

<table>
<thead>
<tr>
<th>New Types of Data</th>
<th>New Types of Analytics</th>
<th>New Workload Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ JSON } Documents</td>
<td>Machine Learning</td>
<td>Micro Services</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>Geo-Distributed</td>
</tr>
<tr>
<td></td>
<td>Spatial</td>
<td>IoT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Blockchain</td>
</tr>
</tbody>
</table>

- Developing and running modern apps across these many engines become increasingly complicated – **bugs, security, upgrades, downtimes**, etc.
The Increasing Complexity of Modern Apps

- One approach to building modern apps is to use a **specialized database** for each application need.
- Each specialized database excels at one aspect of the app’s requirements.

<table>
<thead>
<tr>
<th>New Types of Data</th>
<th>New Types of Analytics</th>
<th>New Workload Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents</td>
<td>Machine Learning</td>
<td>Micro Services</td>
</tr>
<tr>
<td>Graph</td>
<td>Social Graph</td>
<td>Geo-Distributed</td>
</tr>
<tr>
<td>Text</td>
<td>Location</td>
<td>IoT</td>
</tr>
<tr>
<td>Spatial</td>
<td>Lake House</td>
<td>Blockchain</td>
</tr>
<tr>
<td>JSON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MongoDB Documents</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neo4j Graph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elastic Search</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PostGIS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Machine Learning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Social Graph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spatial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spark</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArangoDB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>esri</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Snowflake</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Machine Learning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Social Graph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lake House</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geo-Distributed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AWS QLDB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blockchain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynamoDB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IoT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2022, Oracle and/or its affiliates
The Problems With Using Specialized Databases

- However, this approach inherently creates an application architecture that is heterogeneous and distributed.
- Built from many moving parts that must be learned, synchronized, secured, maintained, and governed.
- Fragments the data and app, which makes app dev more complex, and compromises security and QoS.

Specialized databases also provide limited ACID consistency requiring developers to code app level consistency.

Building apps using specialized databases forces developers to spend their time integrating instead of innovating.
Event Stream Processing Solutions
Oracle Converged Database

- Oracle is a Converged Database
  - Native support for all modern data types and the latest development paradigms built into one product

- New data management technologies are often implemented as separate products
  - With a converged database, you don’t need to manage and maintain multiple systems
  - No need to worry about having to provide unified security across them.

- A good analogy is a smartphone
  - In the past, separate phone, camera, video recorder, gps, music device
What Does an Event Stream Processing Database Need
Why is JSON Necessary for Event Stream Processing

- Event Stream data is highly dynamic
- Formats can change constantly: between readings; after software update, after new type of device is added
- JSON allows applications to easily adapt to changes in data formats. e.g.:
  - The fixed part of this meter event data (Meter ID, timestamp) could be stored in relational columns while the variable Readings data could be stored as JSON

```
{
  "Time": "6/16/2021:12:05:12",
  "HomeID": "BZ125",
  "Readings": {
    "KWH": "0.4"
  }
}
```
Oracle REST Data Services

- REST is the ideal protocol for Event Stream Processing Ingest from lightweight clients
- Oracle Rest Data Services automatically generates REST endpoints for SQL statements
- Transforms SQL results into JSON or other formats (CSV, etc.)
- REST is stateless, all INSERTs/UPDATEs/DELETEs are auto-committed
- Applications access data like any other service via a REST API
- **Simplifies and standardizes APIs to access data**
High-Speed Ingestion
- A memory optimised mechanism for inserting data into the database
- Ideal for ingesting light weight events
- Event rows are buffered in memory and asynchronously drained to disk
- An API allows developers to check on the durability of their inserts
- Ultra-fast - 25 million inserts per second or 21 trillion per day on two socket server
Real Application Clusters: Industry-Leading Scale-Out Compute

- Transparently scales out a database across a pool of hosts sharing the same storage pool
  - Only scale-out technology capable of running the world’s most complex enterprise workloads
- **Scales performance:** more hosts imply more throughput
  - When more throughput is required, simply add a new host
- **Scales fault tolerance:** more hosts imply more availability
  - When a host goes down, the database remains available
- **Scales user experience:** Constant latency as system grows
- **Scales administration:** Larger clusters no harder to manage than smaller clusters, online upgrades and patching
Partitioning: Efficiency and Parallelism for Event Streams

- Partitioning avoids single table insert scaling limitations (E.g. Contention for space allocation)
- Divides large tables into multiple units for scalable ingest
  - Many different partitioning schemes exist for partitioning and sub-partitioning
  - Event stream data typically partitioned by time interval and sub-partitioned by hash of source ID
- Very important scalability mechanism for event streams:
  - Sub-partitioning by source speeds up ingest
  - Partitioning by time reduces data access for analytics
Sharding: Globally Distributed Database Architecture

- Global-Scale applications may prefer to divide massive databases into a farm of smaller databases known as shards
  - Avoids scalability or availability issues with very large databases
  - Each shard can be replicated via Data Guard or Golden Gate

- Native SQL for sharding tables across up to 1000 Shards
  - Routing of SQL based on shard key, and cross shard queries
  - Online addition and reorganization of shards

- Sharding is the Ultimate scalability mechanism
  - Linear scalability of capacity, throughput, user population
  - Improves availability since shards are fault isolated
  - Scales user experience since shards are performance isolated
  - Scales administration since built-in shard management tools make managing of 100s of shards as simple as managing a single database

One giant database divided into several smaller databases (shards)
Real-Time Analytics
Database In-Memory | Real-Time Analytics with Fast OLTP

- **Single** copy of data, **Two** in-memory formats
- Both row and column format for the same table
  - Simultaneously active and consistent
- OLTP uses existing row format
- Analytics uses In-Memory column format
- Database In-Memory is seamlessly built into the Oracle Database – **not a separate engine**
- All enterprise features work: RAC, Dataguard, Flashback, etc.
Database In-Memory | Columnar Format

• Pure In-Memory column format
  • In-Memory maintenance: Fast OLTP
  • No changes to disk format
  • All features (security, availability) work transparently

• Does not require whole database to be in-memory
  • Can be enabled for hot data, at tablespace, table, partition, level

Copyright © 2022, Oracle and/or its affiliates
Database In-Memory | Technology

Columnar Format
Access only the columns you need

SIMD Vector Processing
Process multiple column values in a single CPU instruction

Storage Indexes
Prune out any unnecessary data from the column

Compression
Scan & filter data in compressed format optimized for space and time
Database In-Memory | Improves All Aspects of Analytics

**Scans**
- Billions of Rows per second scans using SIMD Vectorization

**Joins**
- Convert slower joins into 10x faster filtered column scans leveraging In-Memory Columnar Data formats

**Reporting**
- Run reports with aggregations and joins 10x faster using novel memory-optimized algorithms
Inserting one row into a table requires updating 10-20 analytic indexes: Slow!

Fast analytics only on indexed columns

Analytic indexes increase database size

Column Store not persistent so updates are: Fast!

Fast analytics on any columns

No analytic indexes: Reduces database size
Mixed Workloads

• Dual-Format Architecture enables fast Mixed Workloads and faster Analytics
• Fast In-Memory DML because invalid row is logically removed from column store (just set a bit)
• Analytic query will ignore invalid rows in column store, and just vector process valid rows.
• Invalid rows are then processed.
• Mixed workload performance can suffer if the number of invalid rows accumulates in IMCUs
  • Additional techniques to refresh a dirty IMCU in the background
In-Memory Expressions

- Hot expressions can be stored as additional columns in memory
- All In-Memory optimizations apply to expression columns (e.g. Vector processing, storage indexes)
- Two modes:
  - **Manual**: Declare virtual columns for desired inmemory expressions
  - **Auto**: Auto detect frequent expressions
- **3-5x** faster complex queries
Consider a query to find the *Total sales amount for every month in 2022*

```sql
select extract(month from order_date) MONTH,
sum(order_amount) TOTAL_SALES
from SALES
where extract(year from order_date) = 2022
group by extract(month from order_date);
```

- In-Memory can now run such queries by up-to 6X faster by leveraging the In-Memory Expressions framework
  - Each extracted component (e.g. `MONTH`) for a DATE column adds only a 1B per-row in-memory overhead
  - User can specify which DATE column component should be stored in-memory through a parameter
In-Memory Vectorized Joins

If we know ahead of time what tables will be joined, we can make the join fast

- Create inmemory join group JG (Lineitem(l_orderkey), Orders(o_orderkey))

### Global Dictionary

<table>
<thead>
<tr>
<th>L_orderkey</th>
<th>O_orderkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>105125</td>
<td>1</td>
</tr>
<tr>
<td>151252</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>695825</td>
<td>1523</td>
</tr>
<tr>
<td>915238</td>
<td>2698</td>
</tr>
</tbody>
</table>

Hash Join is now changed into simple Array Lookup of codes:
**Converged Workloads**

**In-Memory Analytics on Spatial, Text, and JSON**

1. Store **Spatial** Summaries in Column Store for Faster Filtering

2. Store Optimized **Text** Index structure in Column Store for fast searches

3. Store **JSON** in optimized binary representation in Column Store

```json
{
    "Theater":"AMC 15",
    "Movie":"Rogue One",
    "Time":2017-01-09 18:45",
    "Tickets":{
        "Adults":2
    }
}
```
In-Memory Columnar JSON

- JSON documents stored in Database In-Memory (and in Cell Memory on Storage Nodes with Exadata) automatically get shredded into columns for faster key/value access:

```json
jdoc
{
  "firstName": "John",
  "gender": "male",
  "age": 34,
  "address": {"city": "Redwood City", "state": "CA"},
}
{
  "firstName": "Alan",
  "gender": "male",
  "age": 24,
  "address": {"city": "New York", "state": "NY"},
}
{
  "firstName": "Clara",
  "gender": "female",
  "age": 53,
  "address": {"city": "Dallas", "state": "TX"},
}
```

<table>
<thead>
<tr>
<th>name</th>
<th>gender</th>
<th>address.city</th>
<th>address.state</th>
<th>doc id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alan</td>
<td>Male</td>
<td>New York</td>
<td>NY</td>
<td>2</td>
</tr>
<tr>
<td>Clara</td>
<td>Female</td>
<td>Dallas</td>
<td>TX</td>
<td>3</td>
</tr>
<tr>
<td>John</td>
<td>Male</td>
<td>Redwood City</td>
<td>CA</td>
<td>1</td>
</tr>
</tbody>
</table>

SELECT count(*) FROM employee WHERE json_exists(jdoc, '$.person?(@.age < 34 && @.name = 'John' && @.address.city = 'Redwood City')')

15X Faster Performance
Rich Analytics Query Functionality
SQL for Event Stream Processing

Analytic Window Functions

- Oracle Database has the industry-leading portfolio of analytic functions for event stream processing:
  - **Row Level functions**: These are standard SQL functions returning a single value for each row of input (e.g. ROUND, TRUNC, UPPER, etc.) can be used for interpolation, smoothing, etc.
  - **Aggregate functions**: Return a single value for a group for rows (e.g. MAX, MIN, AVG, SUM etc.)
  - **Window functions**: Return a single value per row, depending on the group of rows (as specified by a window clause) that the row belongs to

- **Window functions** are especially useful for analyzing events across different time periods, e.g.
  - Max energy consumption within each 1 hour interval
  - Ranking of 10 minute energy consumption intervals within each day
  - Greatest change in consumption from prior interval

---

<table>
<thead>
<tr>
<th>MeterID</th>
<th>Time</th>
<th>KWhrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1XC23</td>
<td>9:00pm</td>
<td>2.0</td>
</tr>
<tr>
<td>1XC23</td>
<td>8:45pm</td>
<td>0.86</td>
</tr>
<tr>
<td>1XC23</td>
<td>8:30pm</td>
<td>0.56</td>
</tr>
<tr>
<td>1XC23</td>
<td>8:15pm</td>
<td>0.23</td>
</tr>
<tr>
<td>1XC23</td>
<td>8:00pm</td>
<td>0.4</td>
</tr>
<tr>
<td>1XC23</td>
<td>7:45pm</td>
<td>0.5</td>
</tr>
<tr>
<td>1XC23</td>
<td>7:30pm</td>
<td>0.8</td>
</tr>
<tr>
<td>1XC23</td>
<td>7:15pm</td>
<td>1.5</td>
</tr>
<tr>
<td>1XC23</td>
<td>7:00pm</td>
<td>0.7</td>
</tr>
<tr>
<td>1XC23</td>
<td>6:45pm</td>
<td>0.6</td>
</tr>
<tr>
<td>1XC23</td>
<td>6:30pm</td>
<td>0.9</td>
</tr>
<tr>
<td>1XC23</td>
<td>6:15pm</td>
<td>0.45</td>
</tr>
<tr>
<td>1XC23</td>
<td>6:00pm</td>
<td>0.86</td>
</tr>
<tr>
<td>1XC23</td>
<td>5:45pm</td>
<td>1.34</td>
</tr>
<tr>
<td>1XC23</td>
<td>5:30pm</td>
<td>0.55</td>
</tr>
<tr>
<td>1XC23</td>
<td>5:15pm</td>
<td>1.02</td>
</tr>
</tbody>
</table>

Select MAX(Energy)
(OVER PARTITION BY TIME_IN_HRS)
FROM MeterReadings;
SQL for Event Stream Processing Pattern Matching

- Event Stream data can be further analyzed using the MATCH_RECOGNIZE construct for SQL pattern matching
- MATCH_RECOGNIZE returns rows from a result set that match a specified pattern within a specified ordering of the result set
- Many use cases – detecting fraud, alerting on high usage, finding anomalies in IoT metrics, etc.:
  - Find meter readings which correspond to two successive periods of increased readings (shown here)
  - Detect a double-dip for a particular stock
  - Detect suspect pattern of credit card charges
- Eliminates the need to write complex SQL with self joins and nested sub-queries
Typical applications of machine learning for event streams include failure prediction from sensor data, fraud detection in financial transactions, sentiment analysis from news feeds, spam filters, detection of correlated failures in event logs.

Oracle Database has a very rich portfolio Machine learning models for Event Streaming use-cases.

Oracle Database also supports Auto ML which helps to select the ideal algorithms for a given data set that work best for provided data, settling on right data samples for the model, identifying features in data that provide good signal & minimize noise.

Inferencing based on the models is easily done in real time without requiring any further data movement to a different store.

https://oracle.com/machine-learning
Integrated APIs
SQL | Python | R | REST

Interfaces
Zeppelin-based collaborative notebooks
AutoML UI – no-code ML modeling
Services – model management and deployment
Use 3rd party IDEs
SQL Developer plug-in Oracle Data Miner

ML techniques
classification | regression | clustering
anomaly detection | time series
feature extraction | attribute importance
ranking | row importance
30+ in-database algorithms

Automation
AutoML API and UI
Algorithm-specific data preparation
Integrated text mining
Partitioned model ensembles

Cloud and on premises
Oracle Database
Oracle Autonomous Database
Oracle Database Cloud Service
Oracle Big Data Service

Big Data
Native and Spark MLlib algorithms
Cloud SQL and Big Data SQL

https://oracle.com/machine-learning
Automatic Data Life-Cycle Management
Event Stream Data Characteristics

- Event streams have high data arrival rates and a decaying rate of relevance
  - It is desirable to organize event data in a way that reflects this pattern of usage
- An ideally organized event stream should thus have three optimization zones
  - **Write Optimized Zone**: Organized for fast Ingest
  - **Read Optimized Zone**: Organized for fast analytics
  - **Space Optimized Zone**: Organized for space savings
- These zones may overlap; you may want high speed analytics on recently ingested data as well as on cooler longer term data
Achieving Read, Write & Space Optimization

- Write optimized partitions should be declared MEMOPTIMIZED FOR WRITE
  - Recent partitions are typically uncompressed to achieve max ingest speed
- Read Optimized partitions should be declared INMEMORY in order to enable real-time analytics
- Space Optimized partitions should be compressed or downsampling
- This gradient of Write, Read, and Space optimization can be achieved with Automatic Data Optimization and DBMS_SCHEDULER
## Downsampling Event Streams

- To save space and to accelerate reports on older data, events are often *downsampled* or *summarized* as they age.
- This downsampling action can be performed via the DBMS_SCHEDULER package:
  - E.g. A smart metering application may receive meter readings every five minutes.
  - A DBMS_SCHEDULER job generates hourly summaries from intervals more than 24 hours old, inserts them into a summary table.
  - Note: The summary table can have a different interval partitioning scheme.
  - Can be used to feed other even more granular summaries (e.g. generate daily summaries after a month).

```sql
Insert into MeterSummaries
Select meter_id, 
time_in_hrs, sum(KWhrs)
From MeterReadings
Where <more than 24hrs old>
Group by meter_id, time_in_hrs
```

<table>
<thead>
<tr>
<th>Time Interval</th>
<th>Consumption Interval</th>
<th>Partitioning Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>6pm - 12am</td>
<td>5 min</td>
<td>60 min</td>
</tr>
<tr>
<td>12am - 6am</td>
<td>5 min</td>
<td>60 min</td>
</tr>
<tr>
<td>6am - 12pm</td>
<td>5 min</td>
<td>60 min</td>
</tr>
<tr>
<td>12pm - 6pm</td>
<td>5 min</td>
<td>60 min</td>
</tr>
<tr>
<td>6pm - 12am</td>
<td>5 min</td>
<td>60 min</td>
</tr>
</tbody>
</table>
Oracle Database as an Event Stream Processing System

Many capabilities result in **class-leading** Event Stream Processing Support

<table>
<thead>
<tr>
<th>Flexible data model:</th>
<th>High Speed Ingest:</th>
<th>Streaming Analytics Functionality:</th>
<th>Automatic Event Lifecycle Management:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best-of-Breed</td>
<td>TimesTen</td>
<td>Analytic Window Functions</td>
<td>Advanced Compression</td>
</tr>
<tr>
<td>Relational, JSON,</td>
<td>Application-Tier Cache</td>
<td>Pattern Matching</td>
<td>Automatic Data Optimization</td>
</tr>
<tr>
<td>Spatial, Text, etc.</td>
<td>Memoptimized Tables</td>
<td>Native Machine Learning</td>
<td>DBMS_SCHEDULER</td>
</tr>
<tr>
<td>Native Rest Services</td>
<td>Partitioning</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Real Application Clusters</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sharding</td>
<td>Database In-Memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exadata Persistent Memory</td>
<td>Exadata In-Flash Column Store</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accelerator</td>
<td>Parallel Query</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Attribute Clustering</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Materialized Views</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2022, Oracle and/or its affiliates
Demo

DevOps Monitoring with
Database In-Memory
DevOps Monitoring

• DevOps has been adopted by companies of all sizes for rapid and continuous delivery of IT applications

• Event stream processing Databases allow businesses to:
  • **Rapidly ingest** streams of metrics emitted by DevOps toolchains
  • Analyze and **predict trends** based on recent and historical data in real-time.
  • Create custom **dashboards** to visualize fleet health
  • Detect **anomalous** behavior and raise alerts
Prometheus is a popular open-source systems monitoring and alerting solution.

We thought, “Wouldn’t it be cool to see how easily the Oracle RDBMS can replace Prometheus’ storage engine to demonstrate it’s Event Stream Processing capabilities?”

EXAMPLE OF METRIC COLLECTED:
name=node_cpu_seconds_total,
time=2021-09-26 03:49:23,
value=152359281
instance='slc15rwe.us.oracle.com:9100,
cpu=1
…
Event Streams POC
Prometheus Ecosystem with Oracle Database Server

1. RAC Database
2. Interval Partitioning
3. PL/SQL
4. Memoptimized Ingest
5. JSON Storage
6. Database In-Memory
7. Oracle Grafana Plugin (PromQL or SQL)
8. DBMS_SCHEDULER
9. ADO/ILM Policies

Exporters

Alert Manager

PromQL

Grafana Dashboards

{:< JSON, :, JSON, :, JSON}
Exadata Real-Time Insights

• Exadata Real-Time Insights is a new and comprehensive monitoring solution for extracting detailed statistics/metrics across the Exadata Machine.
  
  • Over 2000 events are collected and streamed from Exadata Database and Storage Servers every second.

  • Metrics such as CPU utilization, Memory Utilization, Available Disk Space, etc.

• Real-Time Insights brings DevOps Monitoring to Exadata Systems
  
  • A monitoring dashboard can be built using an visualization tool such as Grafana to provide a single portal to observe Exadata metrics over time.
DevOps Monitoring Demo | Setup

- Monitor the health of over 500 Exadata Servers at an Oracle data center in real-time.
- Over 2.5 Billion Exadata metrics are captured in an Oracle Database over a 24h period.
- The demo will show how anomalous behavior can be detected instantly thanks to Database In-Memory.

Sample Metric:
```json
{
  "metric": "CL_TEMP",
  "value": 25,
  "time": "10-13-2022 11:40PM PST",
  "tags": {
    "fleet": "phx",
    "server": "phx123",
    "unit": "C"
  }
}
```
Exadata metrics are stored in an **Hourly Interval Partitioned** table.

- Partitioning the data by `TIME` enables partition pruning which accelerates queries by only scanning data in the desired time window.

- Every metric has a `NAME`, `TAGS` describing the metric and its source as a `JSON`, `VALUE`, and `TIME` it was generated.

- Each `TAG` has 7 labels (e.g. the “server” label represents the name of the server that generated the metric).

- Every row is ~170B in size uncompressed.

---

### Schema

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>METRIC_NAME</td>
<td>VARCHAR2(200)</td>
</tr>
<tr>
<td>TAGS</td>
<td>JSON</td>
</tr>
<tr>
<td>VALUE</td>
<td>NUMBER</td>
</tr>
<tr>
<td>TIME</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

### Sample Metric Row

```
CL_TEMP, /* Storage Server Temp */
{
    "objectName" : "EDSCELL2",
    "unit" : "C",
    "server" : "phxdbfcm99",
    "nodeType" : "STORAGE",
    "fleet" : "phx",
    "pod" : "phxdbfcm99",
    "cluster" : "phxdbfcm"
}, /* Tags describing the metric */
42, /* Value of Temperature */
1666026000 /* Time in epoch secs */
```
Database In-Memory - DevOps Monitoring Demo

Detailed metrics/statistics from a fleet of Exadata Machines were collected, over a period of 1 day, through the Exadata Real-Time Insights monitoring solution, and were ingested into the Oracle Database.

We want to demonstrate how the health of this fleet can be monitored in real-time to quickly identify any anomalous behaviors across the servers being tracked - e.g. servers which are overheating, or servers low on memory or disk space.

This demo will show how the analytic queries used in this DevOps monitoring use-case are accelerated by orders of magnitude using Database In-Memory, which is essential for systems requiring immediate action in real-time.

**Data Overview**
Dashboard that details the storage specifics of the data being analyzed

**Health Summary**
Dashboard that monitors the health of various servers in the system

**Performance Summary**
Dashboard that compares performance with and without In-Memory
Data Overview

This dashboard provides an overview of the storage specifics for metrics ingested from a fleet of 500 Exadata Machines located in Oracle's Phoenix Data Center over the last 24 hours.

Metric data is stored in the database using JSON documents for ultimate flexibility in a schema-less data model.

The queries are evaluated in real-time and involve a) JSON processing over metric data, b) Aggregation for analytic reporting, and c) Time-based filtering.

24-Hour Summary

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Data Size</th>
<th>In-Memory Size</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.44 B</td>
<td>457 GB</td>
<td>27.9 GB</td>
<td>16.4 X</td>
</tr>
</tbody>
</table>
select count(distinct json_value(tags,'$ObjectName')) value from metrics_data where (json_value(tags,'$ObjectName') like '%cell%' or json_value(tags,'$ObjectName') like '%CELL%') and time > :start_time - $offset and time < :end_time - $offset
## Health Summary

This dashboard monitors servers that are operating outside the normal window of temperature, CPU/Memory usage and I/O latency, with and without In-Memory.

The thresholds can be configured by variables at the top of the dashboard.

This demo will show how with Database In-Memory we can quickly identify problematic servers in real-time.
# Health Summary (In-Memory)

## Servers exceeding thresholds

<table>
<thead>
<tr>
<th>Temperature</th>
<th>CPU</th>
<th>Memory</th>
<th>I/O Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64</td>
<td>0</td>
<td>57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Server</th>
<th>Last Occurrence</th>
<th>Temperature</th>
<th>CPU_U</th>
<th>Memory</th>
<th>I/O Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>phxdbfcm99</td>
<td>14-OCT-22 07:51:30</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phxdbfcb94</td>
<td>14-OCT-22 07:59:54</td>
<td>99.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phxdbfcb30</td>
<td>14-OCT-22 07:59:54</td>
<td>99.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phxdbfca83</td>
<td>14-OCT-22 07:59:50</td>
<td>99.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phxdbfcv68</td>
<td>14-OCT-22 07:59:35</td>
<td>99.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phxdbfca17</td>
<td>14-OCT-22 07:59:59</td>
<td>99.7</td>
<td></td>
<td></td>
<td>2.19</td>
</tr>
<tr>
<td>phxdbfcq21</td>
<td>14-OCT-22 07:59:54</td>
<td>99.7</td>
<td></td>
<td></td>
<td>1.14</td>
</tr>
<tr>
<td>phxdbfcq67</td>
<td>14-OCT-22 07:57:02</td>
<td>99.8</td>
<td></td>
<td></td>
<td>1.26</td>
</tr>
<tr>
<td>phxdbfccc06</td>
<td>14-OCT-22 07:55:27</td>
<td>99.8</td>
<td></td>
<td></td>
<td>1.15</td>
</tr>
</tbody>
</table>
# Health Summary (without In-Memory)

## Servers exceeding thresholds

<table>
<thead>
<tr>
<th>Temperature</th>
<th>CPU</th>
<th>Memory</th>
<th>I/O Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Temperature icon]</td>
<td>![CPU icon]</td>
<td>![Memory icon]</td>
<td>![I/O Writes icon]</td>
</tr>
</tbody>
</table>
## Health Summary (In-Memory)

### Servers exceeding thresholds

<table>
<thead>
<tr>
<th>Temperature</th>
<th>CPU</th>
<th>Memory</th>
<th>I/O Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64</td>
<td>0</td>
<td>57</td>
</tr>
</tbody>
</table>

**SERVER**

- **phxdbfcm99**
  - Temperature: 14-Oct-22 07:51:30...
  - CPU: 99.7

**I/O Writes**

- phxdbfca17: 14-Oct-22 07:59:59...
- phxdbfca21: 14-Oct-22 07:59:54...
- phxdbfca83: 14-Oct-22 07:59:50...
- phxdbfca68: 14-Oct-22 07:59:53...

- phxdbfca67: 14-Oct-22 07:57:02...
- phxdbfca60: 14-Oct-22 07:55:27...

**NO DATA**

- phxdbfcb94: 14-Oct-22 07:59:54...
- phxdbfcb30: 14-Oct-22 07:59:54...
- phxdbfca83: 14-Oct-22 07:59:50...
- phxdbfca68: 14-Oct-22 07:59:53...

**Memo**

- No data
Anomalous Server Details

This dashboard is used to investigate various metrics for a server around the time of its anomalous behavior using In-Memory

Storage Server Temperature
Health Summary (without In-Memory)

Servers exceeding thresholds

<table>
<thead>
<tr>
<th>Temperature</th>
<th>CPU</th>
<th>Memory</th>
<th>I/O Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64</td>
<td>0</td>
<td>57</td>
</tr>
</tbody>
</table>

Server: phxdbfcm99
Last Occurrence: 14-DEC-22 07:51:30
Temperature: 42
Performance Overview

This dashboard tracks the performance of the health monitoring queries executed with and without In-Memory.

Database In-Memory improves these queries by 100x or more.
This dashboard tracks the performance of the health monitoring queries executed with and without In-Memory.

Database In-Memory improves these queries by 100x or more.
Ingest Rate

1.24 M
Rows Per Minute

ENABLED
Without Ingestion

In-Memory Scan Rate

1422
Million Rows/s

COMPLETED (.124 secs)
185 M Rows scanned

With Ingestion

In-Memory Scan Rate

876
Million Rows/s

COMPLETED (.202 secs)
186 M Rows scanned
Consider the query to find the **Number of distinct servers that have generated metrics in the last 1 hour**

- A local-partitioned index on `(TIME, JSON_VALUE(TAGS, '$.SERVER'))` can achieve **10X faster** query execution over No In-Memory full table scans
- However, it is still **10X-15X slower** than In-Memory query execution
- In-Memory query execution is super-charged through Aggregation pushdown, optimized JSON evaluation, Min-max pruning, etc. which are not available in indexed execution

Indexes occupy additional space, and the database needs a large buffer cache to avoid I/Os
- For e.g., the index described above is 75GB in size (~18% of data size)
- Different indexes need to be created to accelerate other dashboard queries (e.g. index on `(TIME, JSON_VALUE(TAGS, '$.NODETYPE'))` to find number of distinct storage servers)

Further, indexes require maintenance which can slow down DMLs significantly
- Thus, In-Memory is the only solution that can provide instantaneous **Real-Time analytics**
Demo Summary

• Database In-Memory is essential for use-cases like DevOps Monitoring, where real-time anomaly detection and drill-down analysis is absolutely needed.
  • Any loss of time identifying and triaging irregularities in your data fleet can amount to customer dissatisfaction and loss of revenue.

• The demo showed how Database In-Memory could be used to speed up Exadata metrics monitoring by 400x compared to traditional buffer cache row-based processing.
  • With In-Memory enabled, we were able to detect and identify anomalous servers in the data center, and subsequently drill-down into the metric events to identify a potential root cause, all before the non In-Memory dashboard could even reveal that there were problem servers in the fleet.

• Database In-Memory would still be 10-15X faster than system with analytic indexes. But indexes are not practical because of a) high DML cost, b) high I/O due to space needed.
Research and Development Opportunities

Event Stream Processing and more

• Compression Technology
• Hardware Acceleration
• Approximate Indexes
• Simplifying SQL
• Optimizing Algorithms for Database Operations and SQL Functionality
• Machine Learning and Expert Systems for Data Management
• Mixed Workload – Single Database for Operational Data and Reporting
• In-Memory Technology (Analytics and OLTP)
Thank You.

db_career_us_grp@oracle.com