Oracle Database In-Memory
Power the Real-Time Enterprise

Saurabh K. Gupta
Principal Technologist,
Database Product Management

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.
Who am I?

• Principal Technologist, Database Product Management at Oracle

• Author of Packt’s “Oracle Advanced PL/SQL Developer Professional Guide”, Technical Reviewer

• Blog: www.sbhoracle.wordpress.com

• Social: @SAURABHKG

• Email: saurabh.h.gupta@oracle.com
Oracle Database 12c In-Memory Option Goals

- **Orders of Magnitude Faster** Analytics
  - Real Time Queries on OLTP Database or Data Warehouse

- **Faster** Mixed Workload OLTP

- **Simple:** Transparent to applications and easy to deploy

- **Cost-Effective:** Not require entire database to be in-memory
AGENDA

• In-Memory Column Format
• Getting Started with the IM Column Store
• In-Memory Performance Optimizations
• In-Memory Advisor
• In-Memory Scale-Out
Row Format Databases vs. Column Format Databases

- **Transactions** run faster on row format
  - Example: Insert or query a sales order
  - Fast processing few rows, many columns

- **Analytics** run faster on column format
  - Example: Report on sales totals by region
  - Fast accessing few columns, many rows

Until Now Must Choose One Format and Suffer Tradeoffs
Breakthrough: Dual Format Database

- BOTH row and column formats for same table
- Simultaneously active and transactionally consistent
- OLTP uses proven row format
- Analytics & reporting use new in-memory Column format
  - Same results regardless of which format is used
## Dual-Format Database

<table>
<thead>
<tr>
<th>ID</th>
<th>PROD</th>
<th>STORE</th>
<th>QTY</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>983</td>
<td>MILK</td>
<td>SF</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>111</td>
<td>BREAD</td>
<td>LA</td>
<td>20</td>
<td>45</td>
</tr>
<tr>
<td>695</td>
<td>APPLE</td>
<td>SF</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

**Row Store**

**Column Store**

<table>
<thead>
<tr>
<th>ID</th>
<th>PROD</th>
<th>STORE</th>
<th>QTY</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>983</td>
<td>MILK</td>
<td>SF</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>111</td>
<td>BREAD</td>
<td>LA</td>
<td>20</td>
<td>45</td>
</tr>
<tr>
<td>695</td>
<td>APPLE</td>
<td>SF</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>PROD</th>
<th>STORE</th>
<th>QTY</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>983</td>
<td>MILK</td>
<td>SF</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>111</td>
<td>BREAD</td>
<td>LA</td>
<td>20</td>
<td>45</td>
</tr>
<tr>
<td>695</td>
<td>APPLE</td>
<td>SF</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.
Dual-Format Database

• Dual-Format does not double memory requirements
  – In-Memory Store sized to accommodate objects that must be stored in memory
  – Buffer cache is optimized (since decades) to run on much smaller size than the database
  – However, expect to allocate more memory than previously allocated (typically 20% more)

• No additional disk storage cost

• Optimizer is fully aware of the dual formats
  – Automatically routes analytical queries to column store and OLTP to row store
What is a cache?
- A pool of memory
- Data automatically brought into memory based on access
- Data automatically aged out
- Good example: Oracle Database Buffer Cache
In-Memory A Store – Not A Cache

- What is a store?
- A static pool of memory
- You decide what objects are populated in memory

```
ALTER table SALES INMEMORY
```

- Objects don’t age out
- Objects automatically kept transactionally consistent
Oracle In-Memory Columnar Format

- Available on all platforms that the Oracle Database supports
- Pure in-memory column format
  - Not persistence, and no logging
  - In-memory maintenance: Allows fast OLTP
  - No changes to disk format
- Enabled at tablespace, table, partition, subpartition level
  - Can even be enabled for whole database
- 2x – 20x Compression
In-Memory Area: **New Area within SGA**

**System Global Area SGA**

- Buffer Cache
- Shared Pool
- Log Buffer
- Large Pool
- Other
- **In-Memory Area**

- Contains data in the new In-Memory Column Format
- Controlled by INMEMORY_SIZE parameter
In-Memory Area: Composition

- Contains two sub-pools:
  - IMCU pool: Stores In Memory Compression Units (IMCUs)
  - SMU pool: Stores Snapshot Metadata Units (SMUs)
- IMCUs contain column formatted data
- SMUs contain metadata and transactional information
- Current pool sizes and status visible in V$INMEMORY_AREA view
In-Memory Compression Unit (IMCU)

- Unit of column store allocation
  - Columnar representation of a large number of rows (e.g. 0.5 million)
  - Rows in one or more table extents

- Actual size depends on size of rows, compression factor, etc.

- Each column stored as a separate contiguous **Column Compression Unit** (column CU)
  - Rowids also stored as a Column CU
AGENDA

• In-Memory Column Format
• Getting Started with the IM Column Store
• In-Memory Performance Optimizations
• In-Memory Advisor
• In-Memory Scale-Out
Getting Started with the IM Column Store

1. Set init.ora parameter INMEMORY_SIZE

2. Enable Objects for IM Population
   - CREATE or ALTER <table>
   - INMEMORY

3. Populate Objects (Manual or Automatic)

4. Query

Plan uses IM Column Store (Object in-Memory)

Plan uses Buffer Cache/ Disk (Object NOT In-Memory)

Row Format
In-Memory Column Store Populate: Enabling

ALTER TABLE sales INMEMORY;
ALTER TABLE revenue NO INMEMORY;
CREATE TABLE customers ...... 
PARTITION BY LIST 
(PARTITION p1 ...... INMEMORY, 
(PARTITION p2 ...... NO INMEMORY);
ALTER TABLE accounts INMEMORY
NO INMEMORY (photo);

• Selectively enable in-memory storage
• New INMEMORY clause
  • Eligible: Tables, Partitions, Sub-Partitions, Materialized Views
  • Ineligible: IOTs, Hash clusters (pure OLTP features)
• Can also exclude unneeded columns
In-Memory Column Store Populate: Prioritizing

CREATE TABLE orders
  (c1 number,
   c2 varchar(20),
   c3 number)
INMEMORY PRIORITY CRITICAL;

ALTER TABLE sales
INMEMORY PRIORITY MEDIUM;

ALTER TABLE accounts
INMEMORY PRIORITY NONE;

• PRIORITY sub-clause enables pre-populate
  • On startup, create, alter
• Levels:
  • CRITICAL > HIGH > MEDIUM > LOW
  • Controls order (not speed) of populate
• Default PRIORITY is NONE
  • Populate only on first access
In-Memory Column Store Populate: **Compression**

- Objects compressed during population
- New compression techniques
  - Focused on scan performance
- Controlled by MEMCOMPRESS subclause
- Multiple levels of compression
- Possible to use a different level for different partitions in a table

```sql
ALTER MATERIALIZED VIEW mv1 INMEMORY MEMCOMPRESS FOR QUERY;

CREATE TABLE trades
(Name varchar(20),
 Desc varchar(200))
INMEMORY MEMCOMPRESS FOR DML(desc);
```
In-Memory Column Store Populate: **Compression**

CREATE TABLE ORDERS ......
PARTITION BY RANGE ......
 (PARTITION p1 ......
  INMEMORY NO MEMCOMPRESS
  PARTITION p2 ......
INMEMORY MEMCOMPRESS FOR DML,
  PARTITION p3 ......
INMEMORY MEMCOMPRESS FOR QUERY,
  PARTITION p200 ......
INMEMORY MEMCOMPRESS FOR CAPACITY
);

- Different levels
  - FOR DML
    Use on tables or partitions with very active DML activity
  - FOR QUERY
    Default mode for most tables
  - FOR CAPACITY
    For less frequently accessed segments
- Easy to switch levels as part of ILM strategy
In-Memory Column Store Populate: **Under the Hood**

- **V$IM_SEGMENTS**
  - Shows current size of each segment in memory
  - Shows how much remains to be populated

SQL> select segment_name, populate_status, inmemory_priority, inmemory_size, bytes_not_populated from v$im_segments;

<table>
<thead>
<tr>
<th>SEGMENT_NAME</th>
<th>POPULATE_STATUS</th>
<th>INMEMORY_PRIORITY</th>
<th>INMEMORY_SIZE</th>
<th>BYTES_NOT_POPULATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNTS</td>
<td>STARTED</td>
<td>HIGH</td>
<td>196606</td>
<td>2434886912</td>
</tr>
<tr>
<td>SALES</td>
<td>COMPLETED</td>
<td>CRITICAL</td>
<td>135790592</td>
<td>0</td>
</tr>
</tbody>
</table>
AGENDA

• In-Memory Column Format
• Getting Started with the IM Column Store
• In-Memory Performance Optimizations
• In-Memory Advisor
• In-Memory Scale-Out
In-Memory Performance Optimizations

- Columnar Format
- Vector Processing
- Operation pushdown
- In-Memory Storage Index
- Predicate Optimization
In-Memory Row Format: Slower for Analytics

Buffer Cache

Row Format

SELECT COL4 FROM MYTABLE;

RESULT

Needs to skip over unneeded data
In-Memory Columnar Format: Faster for Analytics

SELECT COL4 FROM MYTABLE;

IM Column Store

<table>
<thead>
<tr>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
<th>COL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Column Format

RESULT

Scans only the data required by the query
Vector Processing: Additional Advantage of Column Format

- Each CPU core scans only required columns
- SIMD vector instructions used to process multiple values in each instruction
  - E.g. Intel AVX instructions with 256 bit vector registers
- Billions of rows/sec scan rate per CPU core
  - Row format is millions/sec

Example: Find all sales in state of CA

> 100x Faster
Operation Pushdown: Reduce Rows Processed by Plan

• When possible, push operations down to In-Memory scan
  – Greatly reduces # rows flowing up through the plan
  – Similar to Exadata smart scan, just more smarter
• In addition to SIMD (on SIMD enabled platforms)
• For example:
  – Predicate Evaluation (for qualifying predicates – equality, range, etc.):
    • Inline predicate evaluation within the scan
    • Each IMCU scan only returns qualifying rows instead of all rows
Operation Pushdown: **Bloom Filter**

**Example:** Find total sales in outlet stores

- **Bloom Filter:**
  - Compact bit vector for set membership testing
  - 10g optimizer feature

- **Bloom filter pushdown:**
  - Filtering pushed down to IMCU scan
  - Returns only rows that are likely to be join candidates

- Joins tables **10x** faster
In-Memory Storage Index: **Eliminate IMCUs from Scan**

- **Min-Max Pruning**
  - Min/Max values serve as storage index
  - Check predicate against min/max values
  - Skip entire IMCU if predicate not satisfied

- Can prune for many predicates including equality, range, inlist, etc.

- Eliminates processing unnecessary IMCUs

*Example:* Find stores with sales greater than $10,000
Predicate Optimization: Reduce Predicate Evaluations

- Avoid evaluating predicates against every column value
  - Check range predicate against min/max values
    - As before, skip IMCUs where min/max disqualifies predicate
  - If min/max indicates all rows will qualify, no need to evaluate predicates on column values
  - Also applies Dictionary based pruning

Example: Find stores with sales between $8000 and $14000

- Min $4000 Max $7000
  - Skip IMCU
- Min $8000 Max $13000
  - ALL ROWS
  - Skip Evaluation
- Min $13000 Max $15000
  - SOME ROWS
  - Needs evaluation
Dictionary-based Pruning: Eliminate IMCUs from Scan

• On IMCUs where Dictionary is built - i.e. Dictionary is encoded
  – Check against the Dictionary first
• Skip entire CU/IMCU if dictionary doesn’t have the needed value
  – Eliminates processing unnecessary IMCUs

Example: Find cars built by GM

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audi</td>
<td>0</td>
</tr>
<tr>
<td>BMW</td>
<td>1</td>
</tr>
<tr>
<td>Cadillac</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audi</td>
<td>0</td>
</tr>
<tr>
<td>GM</td>
<td>1</td>
</tr>
<tr>
<td>Kia</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audi</td>
<td>0</td>
</tr>
<tr>
<td>GM</td>
<td>1</td>
</tr>
<tr>
<td>Kia</td>
<td>2</td>
</tr>
</tbody>
</table>
AGENDA

• In-Memory Column Format
• Getting Started with the IM Column Store
• In-Memory Performance Optimizations
• In-Memory Advisor
• In-Memory Scale-Out
Oracle In-Memory Advisor

- In-Production
- Differentiates and analyzes analytics processing from other workloads via AWR, ASH, plan cardinality, and parallel execution
- Estimates In-Memory size of objects based on heuristics compression factors
- Provides list of objects that would benefit most from being populated into IM column store
- Can be installed on Oracle Database 11.2.0.3 and above
## Oracle In-Memory Advisor – Sample Report

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Object</th>
<th>Compression Type</th>
<th>Estimated In-Memory Size</th>
<th>Estimated Analytics Processing Seconds</th>
<th>Estimated Reduced Analytics Processing Seconds</th>
<th>Estimated Analytics Processing Performance Improvement Factor</th>
<th>Benefit/Cost Ratio (Reduced Analytics Processing / In-Memory Size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>SOE.LOGON</td>
<td>Memory compress for query low</td>
<td>451.76MB</td>
<td>2114</td>
<td>1887</td>
<td>9.3X</td>
<td>20.386 : 1</td>
</tr>
<tr>
<td>TABLE</td>
<td>SOE.CARD_DETAILS</td>
<td>Memory compress for query low</td>
<td>607.32MB</td>
<td>8346</td>
<td>7248</td>
<td>7.6X</td>
<td>12.514 : 1</td>
</tr>
<tr>
<td>TABLE</td>
<td>SOE.ADDRESS</td>
<td>Memory compress for query low</td>
<td>1.09GB</td>
<td>3237</td>
<td>4521</td>
<td>8.3X</td>
<td>7.798 : 1</td>
</tr>
<tr>
<td>PARTITION</td>
<td>SOE.PRODUCT_MOCKUP.Y2014Q1</td>
<td>Memory compress for query low</td>
<td>812.6MB</td>
<td>2003</td>
<td>1489</td>
<td>3.9X</td>
<td>4.799 : 1</td>
</tr>
<tr>
<td>TABLE</td>
<td>SOE.CUSTOMERS</td>
<td>Memory compress for query low</td>
<td>1.16GB</td>
<td>108</td>
<td>95</td>
<td>8.2X</td>
<td>7.455 : 1</td>
</tr>
<tr>
<td>TABLE</td>
<td>SOE.ORDER_ITEMS</td>
<td>Memory compress for query low</td>
<td>2.19GB</td>
<td>7128</td>
<td>6393</td>
<td>9.7X</td>
<td>4.429 : 1</td>
</tr>
<tr>
<td>TABLE</td>
<td>SOE.ORDERS</td>
<td>Memory compress for query low</td>
<td>1.34GB</td>
<td>3512</td>
<td>2917</td>
<td>5.9X</td>
<td>4403 : 1</td>
</tr>
<tr>
<td>TABLE</td>
<td>SOE.PRODUCT_INFORMATION</td>
<td>Memory compress for query low</td>
<td>1.78MB</td>
<td>2873</td>
<td>2205</td>
<td>4.3X</td>
<td>2.416 : 1</td>
</tr>
<tr>
<td>PARTITION</td>
<td>SOE.PRODUCT_MOCKUP.Y2015Q1</td>
<td>Memory compress for query low</td>
<td>1.62</td>
<td>97</td>
<td>1489</td>
<td>3.7X</td>
<td>2.284 : 1</td>
</tr>
</tbody>
</table>
AGENDA

- In-Memory Column Format
- Getting Started with the IM Column Store
- In-Memory Performance Optimizations
- In-Memory Advisor
- In-Memory Scale-Out
In-Memory and RAC

- Each node in RAC has its own IM column store
- Objects are automatically distributed across column store
- Scale-out policy is at segment level (table, (sub)partition)
- In-Memory **queries parallelized** across nodes to access local column data
In-Memory and RAC

```
ALTER TABLE sales INMEMORY
DISTRIBUTE BY PARTITION;

ALTER TABLE COSTS INMEMORY
DISTRIBUTE AUTO;
```

- Policy is user-specifiable
- Controlled by `DISTRIBUTE` subclause
  - Distribute by rowid range
  - Distribute by partition/sub-partition
  - Distribute AUTO (default)
Scale-Out: Distribute by Partition
Oracle Database In-Memory: Unique Fault Tolerance

- Similar to storage mirroring
- Duplicate in-memory columns on another node
  - Enabled per table/partition
  - Application transparent
- Downtime eliminated by using duplicate after failure

Only Available on Engineered Systems
Oracle Database In-Memory: Unique Fault Tolerance

```
ALTER TABLE sales INMEMORY DUPLICATE;

ALTER TABLE COSTS INMEMORY DISTRIBUTUTE AUTO DUPLICATE ALL;
```

- Policy is user-specifiable
- Controlled by DUPLICATE subclause
  - DUPLICATE
  - DUPLICATE ALL
- Can improve performance by enabling co-located joins
DUPLICATE and DISTRIBUTE Example

SALES
Q1 Q2
Q4 Q3

SHIPPING

In-Memory Column Store

In-Memory Column Store

In-Memory Column Store

In-Memory Column Store

4 Node RAC Cluster

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.
Oracle Database In-Memory Top Five Tips

• Remember it’s a store - you control when data goes in & out
• Help the optimizer help you: gather representative set of statistics
• Use partitioning: provides fine grained control on what data is populated
• Remove unnecessary indexes: improves DML performance & reduce index confusion
• Use RAC to scale out: not only increases capacity but performance
Summary

• Dual Format Architecture
  – Fully consistent row and column format
  – Best of both worlds OLTP and Analytics performance.
  – Typically, row format (Buffer cache) memory < 10% of column format memory

• New In-Memory Column Format
  – In-memory only representation
  – Seamlessly built into Oracle Database Engine
  – Compatible with all Oracle Database features

• Cost Effective
  – Use in-memory for hot data, flash for intermediate data, disk for cold data
Hardware and Software
Engineered to Work Together