Oracle In-Memory & Data Warehouse: The Perfect Combination?

UKOUG Tech17, 6 December 2017
Dani Schnider, Trivadis AG

@dani_schnider danischnider.wordpress.com
Dani Schnider

- Working for Trivadis in Glattbrugg/Zurich
  - Senior Principal Consultant
  - Data Warehouse Lead Architect
  - Trainer of several Courses
- Co-Author of the books
  - Data Warehousing mit Oracle
  - Data Warehouse Blueprints
- Certified Data Vault Data Modeler

@dani_schnider
danischnider.wordpress.com
Oracle Database In-Memory Architecture

- SGA
- Buffer Cache
- DBWn
- User
- IM Column Store
- C1, C2, C3, C4, C5
- TX Journal
- IMCO
- Wnnn
- Row Format

SELECT
UPDATE
Advantages of Oracle Database In-Memory

- Support of mixed workloads (OLTP / DWH)
- Optimized for analytical queries and real-time reporting
- Fast queries and aggregations on large data sets
- Column-wise compression of data in memory
- Only required columns are read
- Less effort needed for performance optimization
- No application changes necessary
Do we still need a Data Warehouse?

No.
- Analytical queries can be executed directly on operational systems
- No query impact for OLTP systems due to In-Memory Column Store
- Allows real-time reporting on current operational data

Yes.
- Integration of data from different source system
- Historization/versioning of data (traceability)
- Provision of information for different subject areas
Data Warehouse Architecture
Typical Data Warehouse Architecture

- Source Systems
- Staging Area
- Cleansing Area
- Core
- Marts
- BI Platform
- ETL
- Metadata

Example diagram showing the typical architecture of a data warehouse.
What should be loaded into In-Memory Column Store?

<table>
<thead>
<tr>
<th>Data Warehouse</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Staging Area</strong></td>
</tr>
<tr>
<td>![Staging Area Diagram]</td>
</tr>
</tbody>
</table>

Metadata
What should be loaded into In-Memory Column Store?

Typical Use Case for In-Memory in Data Warehouse:

- **Data Marts**

Reasons:

- Query optimization of frequently used queries
- Star Schema is an ideal candidate for In-Memory Column Store
- Only required columns must be read and aggregated
What is different with In-Memory in Data Warehouse?

Nearly nothing.

- Purpose of the data warehouse remains the same
- Architecture (DWH layers) remains the same
- Data models remain the same
- Queries, reports and BI applications remain the same

What changes?

- Physical design of data marts
In-Memory & Physical Design
Physical Design of Data Marts (classically)

- **Dimension Tables:**
  - Primary key constraint
  - ev. Bitmap indexes on filter columns

- **Fact Tables:**
  - Partitioning (usually by date)
  - Foreign key constraints to dimension tables
  - Bitmap index for each dimension key
  - ev. Additional bitmap join indexes

- **Optimizer Statistics**
  - Refresh of statistics after each ETL run

- **Materialized Views**
  - Materialized Views for frequently used aggregation levels
  - Dimension objects for hierarchies on dimension tables
  - ev. Materialized View Logs on Core tables
  - ev. Indexes on Materialized Views
  - ev. Partitioning of Materialized Views
Physical Design of Data Marts (with In-Memory)

- Dimension Tables:
  - Primary key constraint

- Fact Tables:
  - Partitioning (usually by date)
  - Foreign key constraints to dimension tables
  - ev. Partial bitmap indexes

- Optimizer Statistics
  - Refresh of statistics after each ETL run

- In-Memory Column Store
  - All dimension tables
  - Small fact tables (complete)
  - For large fact tables ev. only frequently used measures / dimension keys
  - For partitioned fact tables ev. only current (or frequently used) partitions

- In-Memory Compression Type
  - Typically MEMCOMPRESS FOR QUERY (LOW or HIGH)
## In-Memory Compression Type

<table>
<thead>
<tr>
<th>Compression Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO MEMCOMPRESS</td>
<td>Data is populated without any compression</td>
</tr>
<tr>
<td>MEMCOMPRESS FOR DML</td>
<td>Minimal compression optimized for DML performance</td>
</tr>
<tr>
<td>MEMCOMPRESS FOR QUERY LOW</td>
<td>Optimized for query performance (default)</td>
</tr>
<tr>
<td>MEMCOMPRESS FOR QUERY HIGH</td>
<td>Optimized for query performance as well as space saving</td>
</tr>
<tr>
<td>MEMCOMPRESS FOR CAPACITY LOW</td>
<td>Balanced with a greater bias towards space saving</td>
</tr>
<tr>
<td>MEMCOMPRESS FOR CAPACITY HIGH</td>
<td>Optimized for space saving</td>
</tr>
</tbody>
</table>
Demo 1

a) Load In-Memory Column Store
b) In-Memory Compression Type
Star Query Optimization
Query Optimization on Star Schema

Typical Queries on Star Schema:
- Filter criteria on dimensions
- Facts determined by join with dimensions

Star Schema Problem:
- Tables with filter criteria should be evaluated first
- Join only on two tables at a time
- No relationships between dimensions
Query Optimization on Star Schema

- **Star Join** (Oracle 7)
  - Cartesian product of dimension tables

- **Star Transformation** (Oracle 8.0.4/8.1/11.2)
  - „Join“ between dimensions and bitmap indexes
  - Bitmap indexes can be combined

- **Join Filter Pruning** (Oracle 11.1)
  - Creation of bloom filter for dimensions
  - Access on fact table via bloom filter

- **Vector Transformation** (Oracle 12.1.0.2, In-Memory Option)
  - Key vector for each dimension
  - Determination of facts with combination of key vectors
In-Memory Performance Features

- **In-Memory Scan**
  - Column-wise reading of data from IM Column Store
  - Aggregation to required granularity

- **In-Memory Join** (= Join Filter Pruning)
  - Efficient method to join tables
  - Use of „Bloom Filters“

- **In-Memory Aggregation** (= Vector Transformation)
  - Comparable with Star Transformation
  - Use of „Key Vectors“ instead of bitmap indexes
In-Memory Scans

- Only required columns are scanned
  - Internal In-Memory Storage Index

- In-Memory Expressions
  - Automatical detected expressions via Expression Statistics Store (ESS)

- In-Memory Virtual Columns
  - Must explicitly be loaded in IMCS
In-Memory Joins

- Based on Hash Join
- Bloom Filter
  - Bit vector on join column
  - Result is false positive
- Create Bloom Filter for build input table
- Use Bloom Filter as additional filter on probe input
In-Memory Aggregation (Vector Transformation)

**Phase 1** (for each dimension with filter criteria)
1. Scan on dimension table (including data filtering)
2. Build key vector
3. Aggregation of data (In-Memory Accumulator)
4. Create temporary table

**Phase 2**
5. Full table scan on fact table, filtering using key vectors
6. Aggregation using HASH GROUP BY / VECTOR GROUP BY
7. Join with temporary tables (Join Back)
8. Ev. join with additional dimensions (without filter criteria)
In-Memory Aggregation (Vector Transformation)

**FACTS**

<table>
<thead>
<tr>
<th>DIM1</th>
<th>DIM2</th>
<th>FACTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 Alpha</td>
<td>21 X green</td>
<td>11 22 1000</td>
</tr>
<tr>
<td>12 Alpha</td>
<td>22 X blue</td>
<td>11 24 1200</td>
</tr>
<tr>
<td>13 Beta</td>
<td>23 Y green</td>
<td>12 21 300</td>
</tr>
<tr>
<td>14 Beta</td>
<td>24 Y blue</td>
<td>12 22 3200</td>
</tr>
<tr>
<td>15 Beta</td>
<td>25 Y red</td>
<td>12 24 700</td>
</tr>
<tr>
<td>16 Gamma</td>
<td>26 Z red</td>
<td>13 22 1100</td>
</tr>
<tr>
<td>17 Delta</td>
<td></td>
<td>14 21 2000</td>
</tr>
<tr>
<td>18 Delta</td>
<td></td>
<td>14 24 800</td>
</tr>
</tbody>
</table>

**DIM1**

<table>
<thead>
<tr>
<th>DIM1</th>
<th>DIM2</th>
<th>FACTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 Alpha</td>
<td>21 X green</td>
<td>11 22 1000</td>
</tr>
<tr>
<td>12 Alpha</td>
<td>22 X blue</td>
<td>11 24 1200</td>
</tr>
<tr>
<td>13 Beta</td>
<td>23 Y green</td>
<td>12 21 300</td>
</tr>
<tr>
<td>14 Beta</td>
<td>24 Y blue</td>
<td>12 22 3200</td>
</tr>
<tr>
<td>15 Beta</td>
<td>25 Y red</td>
<td>12 24 700</td>
</tr>
<tr>
<td>16 Gamma</td>
<td>26 Z red</td>
<td>13 22 1100</td>
</tr>
<tr>
<td>17 Delta</td>
<td></td>
<td>14 21 2000</td>
</tr>
<tr>
<td>18 Delta</td>
<td></td>
<td>14 24 800</td>
</tr>
</tbody>
</table>

**SELECT D1, D21, D22, SUM(FACTS.F), FROM FACTS, JOIN DIM1 ON (....), JOIN DIM2 ON (....), WHERE D1 IN ('Beta', 'Gamma'), AND D21 = 'Y', GROUP BY D1, D21, D22
In-Memory Aggregation (Vector Transformation)

<table>
<thead>
<tr>
<th>DIM1</th>
<th>KV1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Alpha</td>
</tr>
<tr>
<td>12</td>
<td>Alpha</td>
</tr>
<tr>
<td>13</td>
<td>Beta</td>
</tr>
<tr>
<td>14</td>
<td>Beta</td>
</tr>
<tr>
<td>15</td>
<td>Beta</td>
</tr>
<tr>
<td>16</td>
<td>Gamma</td>
</tr>
<tr>
<td>17</td>
<td>Delta</td>
</tr>
<tr>
<td>18</td>
<td>Delta</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIM2</th>
<th>KV2</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>X green</td>
</tr>
<tr>
<td>22</td>
<td>X blue</td>
</tr>
<tr>
<td>23</td>
<td>Y green</td>
</tr>
<tr>
<td>24</td>
<td>Y blue</td>
</tr>
<tr>
<td>25</td>
<td>Y red</td>
</tr>
<tr>
<td>26</td>
<td>Z red</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FACTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 22 1000</td>
</tr>
<tr>
<td>11 24 1200</td>
</tr>
<tr>
<td>12 21 300</td>
</tr>
<tr>
<td>12 22 3200</td>
</tr>
<tr>
<td>12 24 700</td>
</tr>
<tr>
<td>13 22 1100</td>
</tr>
<tr>
<td>14 21 2000</td>
</tr>
<tr>
<td>14 24 800</td>
</tr>
<tr>
<td>14 25 1600</td>
</tr>
<tr>
<td>14 26 700</td>
</tr>
<tr>
<td>15 23 1100</td>
</tr>
<tr>
<td>15 24 1200</td>
</tr>
<tr>
<td>15 26 500</td>
</tr>
<tr>
<td>16 22 2400</td>
</tr>
<tr>
<td>16 23 800</td>
</tr>
<tr>
<td>17 22 1300</td>
</tr>
<tr>
<td>17 25 1100</td>
</tr>
<tr>
<td>18 21 900</td>
</tr>
<tr>
<td>18 24 2100</td>
</tr>
<tr>
<td>18 26 600</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Beta</td>
</tr>
<tr>
<td>2 Gamma</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Y green</td>
</tr>
<tr>
<td>2 Y blue</td>
</tr>
<tr>
<td>3 Y red</td>
</tr>
</tbody>
</table>

---

Beta Y green 1100
Beta Y blue 2000
Beta Y red 1600
Gamma Y green 800
In-Memory Configuration
In-Memory Configuration for Individual Tables

- **Complete Table**
  
  ```sql
  ALTER TABLE sales INMEMORY
  ```

- **Individual Columns**
  
  ```sql
  ALTER TABLE sales INMEMORY NO INMEMORY (cust_id, time_id)
  ```

- **Individual Partitions**
  
  ```sql
  ALTER TABLE sales MODIFY PARTITION p_2016 INMEMORY
  ```
In-Memory Configuration for a Star Schema

Typical Configuration:

- **Dimension Tables**
  - Completely loaded in In-Memory Column Store

- **Fact Table**
  - Partitioned by date column
  - Current (or frequently used) partitions in In-Memory Column Store
  - History Partitioned not in In-Memory Column Store
In-Memory & Partial Local Indexes

INDEXING ON

INDEXING OFF

Table Partition P1
Table Partition P2
Table Partition P3
Table Partition P4

Table Partition P5
Table Partition P6

INDEXING ON

INDEXING OFF
Demo 2

a) In-Memory & Star Schema
b) In-Memory & Partial Local Indexes
Oracle In-Memory & Data Warehouse: The Perfect Combination?
Oracle Database In-Memory & Data Warehouse

Oracle Database In-Memory:
- Powerful Performance Features for Query Performance
- Perfect for Analytical Queries on Star Schemas
- No Impact on Architecture of Data Warehouse

To Be Considered:
- Physical Database Design must be Adapted
- Important to Understand how In-Memory Works
- Corrections and Enhancements in Oracle 12.2
Oracle In-Memory & Data Warehouse: (Almost) The Perfect Combination!
Further Information

- Oracle Database In-Memory and Hash Keys

- Syntax Issues with INMEMORY Column Clause

- Derived Measures and Virtual Columns