ORACLE
DATABASE

Oracle Database In-Memory: In-Memory Aggregation

ORACLE WHITE PAPER | JANUARY 2015

ORACLE

Table of Contents

.

Introduction

In-Memory Aggregation, a feature of the Oracle Database In-Memory, is the new best practice for executing star queries in the Oracle Database. In-Memory Aggregation provides new SQL execution operations that accelerate the performance of wide range of analytic queries against star and similar schemas. In-Memory Aggregation optimizes joins between dimension and fact tables, and computes aggregations using CPU efficient algorithms. The relative advantage of In-Memory Aggregation increases as queries become more difficult, joining more dimension tables to the fact table and aggregating over more grouping columns and more fact rows. In-Memory Aggregation provides the greatest benefit when used with together with tables populated in the in-memory column store, though In-Memory Aggregation also delivers big benefits when used with conventional tables stored on disk.

In-Memory Aggregation will be automatically chosen by the SQL optimizer based on execution cost. Applications do not need to be modified to use In-Memory Aggregation.

Benefits of In-Memory Aggregation

In-Memory Aggregation (IMA) is designed to provide improved query performance while utilizing fewer CPU resources. In-Memory Aggregation provides fast query performance with fully dynamic aggregation of data without the need for indexes, summary tables or materialized views. In-Memory Aggregation typically provides a 3-8 times improvement in query performance over non-IMA plans, with more consistent performance and fewer longer running queries.

Without the need to support objects such as indexes and summary tables, IMA can provide fast and consistent query performance across more dimension tables (more joins) and more attributes (more grouping columns), and across dimension and fact tables that are being updated in real-time. As compared to alternative SQL execution plans (for example, hash joins and Bloom filters) IMA plans use less CPU, leaving the Database with additional capacity to support more concurrent users.

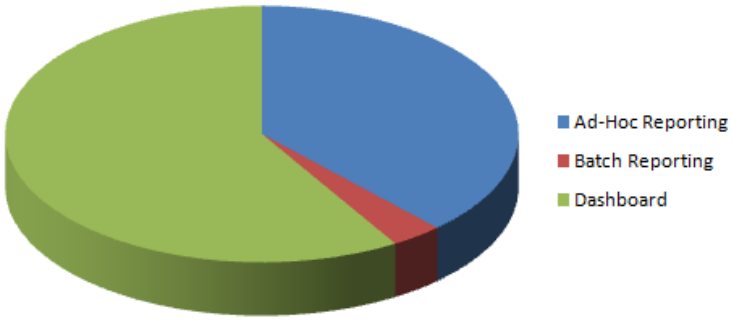
By providing fast and consistent query performance In-Memory Aggregation, along with objects in the in-memory column store, simplifies management of the schema through the elimination of indexes and summary tables.

To illustrate the benefits of In-Memory Aggregation and the in-memory column store, this paper presents a case study representing a varied business intelligence workload. This workload, with a wide range of queries, will be easily recognizable by data warehouse architects and business intelligence application developers as a reasonable representation of a business intelligence query workload.

In all, there were 2,760 queries representing 3 business intelligence workloads (more details on the workload can be found in the appendix):

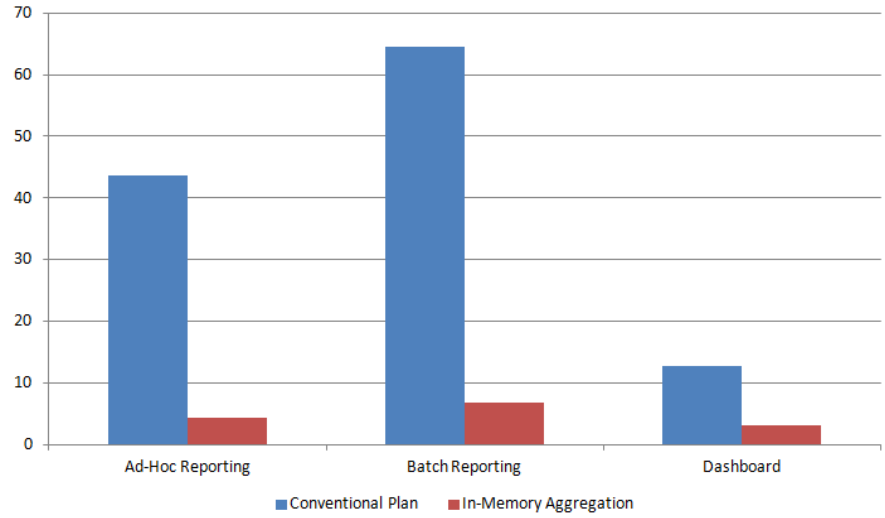
- Dashboard reports selecting data at high and mid-level aggregate levels with moderate selectively (filtering) and a mix of 4 and 9 dimensions. These reports answer question such as *"How is the business performing, world-wide, this year or quarter?"* and *"How are my departments, products sales district and stores performing this quarter or this month?"*.
- Batch reporting of detailed information with less selective filters, returning many rows. *"How are specific products performing on a store by store basis?",* with reports distributed to all buyers, merchandisers and store managers via PDF files.

» Ad-hoc reporting, representing wide range of queries generated by business users who are slicing and dicing data at different levels of aggregation. *Everything else – a highly unpredictable query workload, generated by sales and marking analysts*.



Workload distribution for the In-Memory Aggregation case study.

In-Memory Aggregation (IMA) was on average 7.0 times faster than conventional query processing techniques, provided the most consistent query performance, and scaled the best in terms of concurrent users. In-Memory Aggregation improved the most difficult queries the most –those that aggregated the most fact rows, joined the largest number of dimension tables to the fact table, and returned the largest numbers of rows. Not every query on a star schema will be improved by IMA (a decision which the query optimizer will determine); however, for this workload, In-Memory Aggregation was faster with 94.8 percent of queries, by an average of 24.7 seconds. Conventional plans were faster with only 5.2% of queries and then only by an average of 1.3 seconds. Thus, for star schemas, the query optimizer will be expected to choose to use IMA frequently, but not for every query.



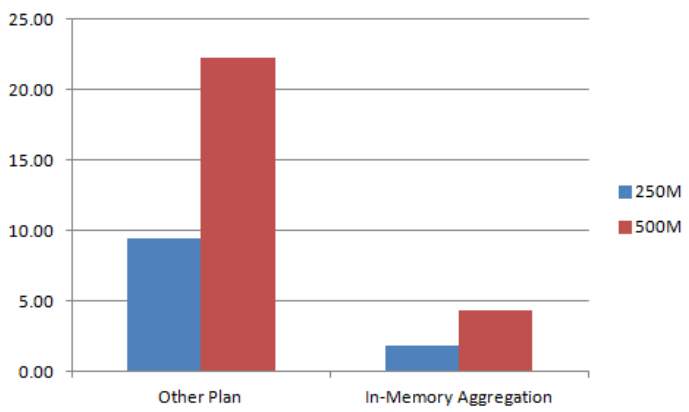
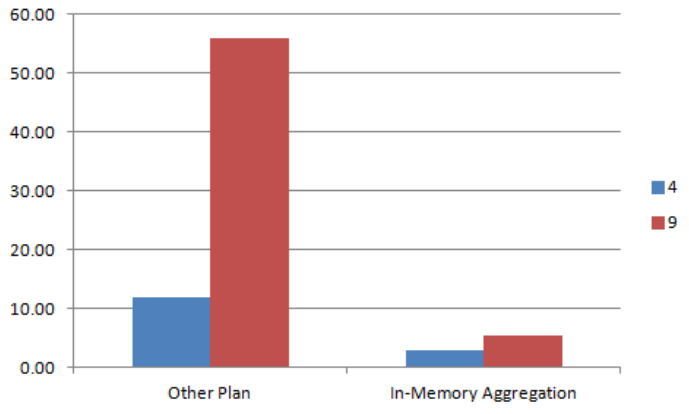
Average query performance (seconds per query) of objects in the in-memory column store with conventional execution plans and In-Memory Aggregation for various business intelligence query workloads

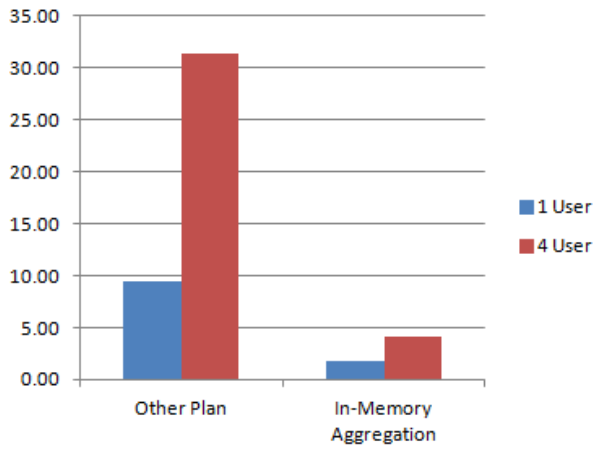
Data Set Size, Concurrent Users and Query Complexity

The size of the data set, query complexity and number of concurrent users (as well as other factors such as hardware, server configuration, etc.) all affect query performance. All other things being equal, IMA plans will scale better than non-IMA plans for query complexity and number of concurrent users and about the same for data set size.

The following table breaks out all queries in the workload by fact table, single and 4 user runs and numbers of dimensions.

FACTORS AFFECTING SCALABILIY

Comparison	**Query Performance (Average Elapsed Times, Seconds per Query)**
Data Set Size . Compares two star schema: » *250 million row fact table with 600,000 row customer table.* » *500 million row fact table with 6 million row customer table.* IMA and non-IMA plans scale at similar rates, with IMA plans having lower average times for each.	
Query Complexity Compares 4 and 9 dimensional queries. The 9 dimensional queries require more effort for joins and tend to return more rows. IMA plans are on average 4 times faster for the 4 dimensional queries and 10.4 times faster for the 9 dimensional queries. This highlights the efficiency of transforming joins to KEY VECTOR filters on the fact table and aggregating data in a single pass.	

Comparison	Query Performance (Average Elapsed Times, Seconds per Query)
Concurrent Users Compares a query runs of 1 and 4 concurrent users (no waits between queries). IMA plans are on average 5.2 times faster in the 1 user run and 7.7 times faster in the 4 concurrent users run. This highlights the advantage of lower CPU use by In-Memory Aggregation.	

Understanding In-Memory Aggregation Processing

This section will help you identify SQL execution plans that use In-Memory Aggregation and explains how the IMA plans process queries.

In-Memory Aggregation uses a *vector transformation* plan to minimize the amount of data that must flow through the execution plan (that is, from one operation to the next), transform joins into fast filters on the fact table and use array structures to accumulate aggregate data. This strategy uses less CPU as compared to alternative plans.

Sample Query and SQL Execution Plan

The following example will identify the key operations in the vector transformation execution plan. Consider the following query.

```
SELECT d1.calendar_year_name,
  d1.calendar_quarter_name,
  d3.region_name,
  SUM(f.sales),
  SUM(f.units)
FROM time_dim d1,
  customer_dim d3,
  units_fact f
WHERE d1.day_id           = f.day_id
AND d3.customer_id        = f.customer_id
AND d1.calendar_year_name = 'CY2012'
GROUP BY d1.calendar_year_name,
  d1.calendar_quarter_name,
  d3.region_name;
```

This query will result in the following SQL execution plan:

```
-------------------------------------------------------------------------------
| Id  | Operation                               | Name                        |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                        |                             |
|   1 |  TEMP TABLE TRANSFORMATION              |                             |
|   2 |   LOAD AS SELECT                        | SYS_TEMP_0FD9DADAD_9873DD   |
|   3 |    VECTOR GROUP BY                      |                             |
|   4 |     KEY VECTOR CREATE BUFFERED          | :KV0000                     |
|   5 |      PARTITION RANGE ALL                |                             |
|   6 |       TABLE ACCESS INMEMORY FULL        | TIME_DIM                    |
|   7 |   LOAD AS SELECT                        | SYS_TEMP_0FD9DADAE_9873DD   |
|   8 |    VECTOR GROUP BY                      |                             |
|   9 |     KEY VECTOR CREATE BUFFERED          | :KV0001                     |
|  10 |      TABLE ACCESS INMEMORY FULL         | CUSTOMER_DIM                |
|  11 |   HASH GROUP BY                         |                             |
|  12 |    HASH JOIN                            |                             |
|  13 |     HASH JOIN                           |                             |
|  14 |      TABLE ACCESS FULL                  | SYS_TEMP_0FD9DADAE_9873DD   |
|  15 |      VIEW                               | VW_VT_AF278325              |
|  16 |       VECTOR GROUP BY                   |                             |
|  17 |        HASH GROUP BY                    |                             |
|  18 |         KEY VECTOR USE                  | :KV0001                     |
|  19 |          KEY VECTOR USE                 | :KV0000                     |
|  20 |           PARTITION RANGE SUBQUERY      |                             |
|  21 |            TABLE ACCESS INMEMORY FULL   | UNITS_FACT                  |
|  22 |     TABLE ACCESS FULL                   | SYS_TEMP_0FD9DADAD_9873DD   |
-------------------------------------------------------------------------------
```

The KEY VECTOR USE and VECTOR GROUP BY operations indicate that this is a vector transformation plan. The steps in the plan are outlined below.

1. The Database creates a *key vector* object (plan IDs 4 and 9) for each dimension table. Key vector objects are stored in-memory (in the SGA) and contain a mapping between a *dense grouping key* (a surrogate key) and the join keys (for example, DAY_ID). While the in-memory format is different (and can vary by the data type of the join keys) the following table illustrates the concept of the key vector object:

DAY_ID	DENSE_GROUPING KEY
1-Jan-12	62
2-Jan-12	62
3-Jan-12	62
4-Jan-12	62
	...
28-Dec-12	47
29-Dec-12	47
30-Dec-12	47
31-Dec-12	47

   Each value of the dense grouping key maps to a single aggregate value. In this example, the time dimension is group by CALENDAR_YEAR_NAME, CALENDAR_QUARTER_NAME. So the dense grouping key value of 62 corresponds to the first quarter of 2012 and the individual DAY_ID corresponding to that quarter are mapped to this grouping key value.

2. For each dimension table, the database also creates a temporary table with selected columns and filtered rows from that dimension table, plus the dense grouping key (plan IDs 2-6 for the TIME_DIM table and plan IDs 7-10 for the CUSTOMER_DIM table). The VECTOR GROUP BY operations (IDs 3 and 8) groups by an internal function to produce unique rows for the dense grouping key. Predicates on the dimension tables are applied during the table scans (IDs 6 and 10).

   The purpose of the temporary table is to save this data for late materialization of the final row set. From this point forward, until the final row set is materialized, only the dense grouping key needs to be carried

through the execution plan. This significantly reduces the amount of data that needs to be processed during the joins and aggregation of data from the fact table, saving CPU and memory. This becomes a greater advantage as the number of columns selected from the dimension table increases.

The temporary table containing the selected columns and filtered rows from the TIME_DIM table and the dense grouping key is illustrated below.

CALENDAR_YEAR_NAME	CALENDAR_QUARTER_NAME	DENSE_GROUPING_KEY
CY2012	Q1-CY2012	62
CY2012	Q2-CY2012	78
CY2012	Q3-CY2012	36
CY2012	Q4-CY2012	47

3. The fact table is scanned using the key vectors (plan IDs 18 and 19). These operations transform joins between the dimension table and fact tables into a scan (filter) of the fact table. This can be observed in the Predicate Information section of the execution plan as in the following example.

```
22 - inmemory(SYS_OP_KEY_VECTOR_FILTER("F"."DAY_ID",:KV0000) AND
     SYS_OP_KEY_VECTOR_FILTER("F"."CUSTOMER_ID",:KV0001))
     filter(SYS_OP_KEY_VECTOR_FILTER("F"."DAY_ID",:KV0000) AND
     SYS_OP_KEY_VECTOR_FILTER("F"."CUSTOMER_ID",:KV0001))
```

This approach has 2 advantages:

- Fast lookups into the key vector object replace more expensive hash join processing.
- In most cases rows in the fact table are scanned only once from all joins, eliminating the need to redistribute rows from one hash join to the next hash join.

4. As rows begin to flow from KEY VECTOR USE to VECTOR GROUP BY (plan ID 16), data are simultaneously aggregated into an in-memory array indexed by the dense grouping keys (this array is referred to as an *aggregate accumulator*).

The following illustrates the concept of the VECTOR GROUP BY aggregate accumulator. The dense grouping keys for time_dim are on the left side, and the dense grouping keys for customer_dim are across the top.

	39	17	91	84	26
62					
78					
36					
47					

As the fact table is scanned and rows meet the conditions of the key vector filters, values are stored in the aggregate accumulator.

	39	17	91	84	26
62					
78	653				
36				2574	
47		368			

As additional fact rows scanned using the key vector filter, a running total is updated in the aggregate accumulator.

	39	17	91	84	26
62					930822
78	841113		640322		
36				920321	
47		900707			950201

So, for example, 841,113 is the running total for the time period Q2-CY2012 and the region corresponding to the dense grouping key of 39.

5. The vector group by accumulator is very efficient, providing that it does not grow too large. If the database determines that it will grow too large, aggregation fails over to the HASH GROUP BY operation (plan ID 17) which, for a larger set of output rows, is more memory efficient.
6. It is possible that some rows will be aggregated using VECTOR GROUP BY and the remaining rows be aggregated by HASH GROUP BY. The final HASH GROUP BY (plan ID 11) aggregates data from each into a single row set.
7. The aggregated fact rows are joined to the temporary tables using the dense grouping key (IDs 12, 13, 14 and 22), producing the final row set. Note that joins occur between what is usually a subset of rows from the dimension tables and aggregate fact rows, often reducing the number of rows joined by factors of millions.

CALENDAR_YEAR_NAME	CALENDAR_QUARTER_NAME	REGION_NAME	SUM(F.SALES)	SUM(F.UNITS)
CY2012	Q1-CY2012	Africa	3888852	14035
CY2012	Q1-CY2012	Asia	39166100	141780
CY2012	Q1-CY2012	Europe	13317259	48186
CY2012	Q1-CY2012	North America	22446217	81160
...	...	...	...	...
CY2012	Q4-CY2012	Europe	13484616	48707
CY2012	Q4-CY2012	North America	22687904	82124
CY2012	Q4-CY2012	Oceania	65602	250
CY2012	Q4-CY2012	South America	9881804	35779

In-Memory Aggregation and Conventional Tables

In-Memory Aggregation accelerates joins and aggregation, while objects populated in the in-memory column store accelerate scan and filter operations. Queries that access data from conventional row store tables can use and will typically benefit from the vector transformation plan, providing that the query is appropriate for that plan. This can occur with tables that are entirely conventional row store or with tables where some partitions are loaded into memory and other partitions are not. The vector transformation plan can also be used when some tables are conventional and other tables are in-memory.

In-Memory Aggregation and Exadata

In many cases not all tables will fit into memory; however In-Memory Aggregation can be used with conventional tables stored on disk with similar performance benefits (less the scan speed of the in-memory column store). When In-Memory Aggregation is used with Exadata and conventional tables, performance of In-Memory Aggregation is enhanced by the ability to offload the KEY VECTOR USE operation to Exadata storage servers. The offload capability distributes key vector processing across Exadata storage servers and minimizes the volume of data that must be returned to the database nodes.

When an entire table is loaded into an in-memory table on the database node(s), key vector processing will always occur on the in-memory table. In cases where the entire table is not loaded into memory (for example, only the most recent partitions are loaded into in-memory and other partitions are on disk), the key vector operation can be off loaded to the storage server for the partitions that are on disk.

Schema and Query Compatibility

In general, the vector transformation plan is designed for queries that join one or more relatively small dimension tables to one or more relatively large fact tables and aggregate measures in the fact table. More specifically:

- The following aggregation operators are supported: SUM, AVERAGE, MIN, MAX, STD and VARIANCE. COUNT DISTINCT operations are not supported by VECTOR GROUP BY, but the query block can use the vector transformation plan if at least one measure from the fact table is aggregated using a supported aggregation operator. When this occurs, aggregation of COUNT DISTINCT will fail over to the HASH GROUP BY operator.
- The grouping syntax used in the query is GROUP BY. GROUPING SETS, GROUP BY ROLLUP and GROUP BY are not currently supported by the vector transformation plan.
- All joins must be equijoins. For the best performance, columns on both sides of the join should be the same data type and not be wrapped in a function.
- Multiple columns from a dimension table may join the fact table, however they all must be the same join type.

SQL Optimizer and In-Memory Aggregation

The SQL optimizer chooses whether to use In-Memory Aggregation based on the cost of the vector transformation plan as compared to other plans.

The vector transformation plan trades off a higher cost of creating the key vector object (as compared to, for example, a hash table) for more efficient scans of the fact table and late joins and materialization. The time used to create the key vector object is proportional to the size of the dimension table while the benefit of the key vector is proportional to the size of the fact table. The VECTOR GROUP BY operation is more efficient than hash group by when aggregating many fact rows into relatively few output rows. The advantage of late joins and materialization is proportional to the number of columns and rows that are returned from the dimension tables and the number of aggregate rows returned from the fact table. These comparisons suggest the SQL optimizer will choose a vector transformation plan when:

- Relatively small dimension tables are joined to relatively large fact tables, with the threshold of dimension table size being about 1/10th the size of the fact table.
- More dimension tables are joined to the fact table.
- Relatively larger numbers of fact rows will be aggregated.
- A relatively small number of rows will be produced by the aggregation.

The relatively high cost of the KEY VECTOR CREATE operation also suggests that alternative plans might have an advantage with queries that join few dimension tables to fact tables and are highly selective. Those queries would tend to cost higher due to KEY VECTOR CREATE and have less of a cost advantage for KEY VECTOR USE and VECTOR GROUP BY.

Conclusion

In-Memory Aggregation, particularly when used with the in-memory column store, provides several interrelated benefits:

- Faster query response.
- The ability to support a more flexible reporting environment.
- Support for real time analytics.
- Fewer demands on system resources.
- Simplifies management of the schema.

The primary driver behind these benefits is the ability to provide fast query performance with fully dynamic calculation of aggregate data, without the need to for indexes, summary tables or materialized views. In-Memory

Aggregation (IMA) typically provides a 3-8 times improvement in query performance as compared to non-IMA plans. Without the need for supporting objects such as summary tables, it is possible to provide consistently fast query performance with more attributes, more measures and against tables being updated in real-time. As compared with other SQL execution plans IMA uses less CP, leaving the system available to support more concurrent users or different workloads.

Appendix – Case Study Technical Details

The case study runs 2,760 queries run against two 9 dimensional star schemas on an Intel blade-type server with 12 CPU cores, 96 GB of RAM and local disk storage. In order to isolate the benefit of In-Memory Aggregation, all tables were in-memory. Running the workload on commodity hardware highlights the processing efficiency of In-Memory Aggregation and the in-memory column store rather than raw processing power of larger hardware.

The query workload was designed to represent a varied business intelligence workload ranging from high-level dashboard reports to ad-hoc analytic queries. The workload selects a variety of attributes and measures from the following tables:

- A common set of 8 dimension tables (time, product, channel tables and 5 tables with demographic attributes).
- A 250 million row fact table with 10 measures, paired with a 601,000 row customer dimension table.
- A 500 million row fact table with 10 measures, paired with a 6.1 million row customer dimension table.

Queries were randomly generated using rules that produce queries the following characteristics:

- Groups of 4 and 9 dimensions, some with fixed sets of dimensions and others with randomly selected dimensions.
- Some include queries at level high level aggregates and others drilled down to more detail level data.
- Two different styles of filtering, resulting in sets of queries that are more and less selective (that is, returning more or fewer rows).
- Fixed and random selections of 3 of 10 measures aggregated using SUM.

At runtime:

- All queries accessed in-memory tables.
- Some queries are run with single user and others with 4 concurrent users, all with no wait time between queries.
- Default parallelism was used.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

Author : William Endress