Resiliency
Learning, adapting and growing from failure.

Features:
More In-Memory Magic
Oracle ACFS
High Availability NFS Services (HANFS)

Facing the Mobilegeddon with Oracle WebCenter Sites
DBA 201: SSH ...
Keys to the Kingdom

www.ioug.org
In the last two articles in this series, I’ve described how Oracle Database 12.1.0.2’s new In-Memory features offer the opportunity to dramatically improve query performance with minimal modifications to a database instance’s memory allocations and without having to change any application code. Wrapping up our in-depth review of Oracle Database 12.1.0.2’s in-memory features, in this issue’s article, I’ll demonstrate:

- How in-memory joins (IMJ) speed the application of bloom filters for star schema queries
- How in-memory aggregation (IMA) works in concert with IMJ, and how the query optimizer decides to choose between these methods
- How the new advanced index compression (AIC) features offer the ability to compress indexes for significant space savings and faster access to data when an index can still be leveraged for data retrieval

In-Memory Storage Indexes

Even though it’s possible that a large amount of a table’s data that is necessary to answer a query may already be present within the IMCS, for an extremely large table, it is also likely that only part of its data may be present. It’s therefore important for the optimizer to know unequivocally if the desired table’s data has not yet been populated when it’s constructing an execution plan.

Fortunately, Oracle 12.1.0.2 provides a solution — the In-Memory Storage Index — which retains metadata about all the data populated within the IMCS. Storage indexes were previously only available within the Exadata Storage Cell software; they make it possible to determine if an ASM AU already exists in a proper state within the Exadata Smart Flash Cache. Adopting this mechanism for the IMCS is a flash of genius because now it’s possible to scan only the required IMCS compression units (IMCUs) that have already been populated with the data to answer a query.

An astute Oracle DBA can either encourage or limit the query optimizer to leverage storage indexes within the IMCS and its IMCUs via the +INMEMORY and +NO_INMEMORY optimizer hints at the statement block level, respectively. Likewise, optimizer hints +INMEMORY_PRUNING(table_name) and +NO_INMEMORY_PRUNING(table_name) respectively instruct the optimizer to either leverage or ignore in-memory storage indexes to prune away missing values within IMCUs.
In-Memory Aggregation (IMA)

Once a table’s data segments are partially or completely resident within the IMCS, Oracle 12.1.0.2 offers another advantage: the ability to efficiently perform aggregation for a star schema query — the typical type of query employed within data warehousing application workloads — completely within the confines of the compute nodes’ CPUs. This new capability is called in-memory aggregation (IMA), and it uses a special type of optimizer method called vector transformation that changes a query’s execution plan so that it builds temporary tables in memory for the dimension tables in a star schema query. The optimizer can then use single instruction processing multiple data values (SIMD) vector processing to more quickly identify the matching members in the dimension and fact tables.

IMA Basics: Dense Keys, Key Vectors and Data Flow Operators

Let’s take a look at a candidate query that might be able to take advantage of in-memory aggregation. Listing 1 shows a typical star schema query against the SH.SALES fact table and two of its dimensions, SH.PROMOTIONS and SH.PRODUCTS:

```
SELECT R.promo_id, P.prod_id, SUM(S.quantity_sold), SUM(S.amount_sold)
FROM SH.sales S, SH.promotions R, SH.products P
WHERE S.prod_id = P.prod_id
AND S.promo_id = R.promo_id
AND P.prod_id BETWEEN 60 AND 120
AND R.promo_id > 99
GROUP BY R.promo_id, P.prod_id
ORDER BY R.promo_id, P.prod_id;
```

Listing 1: In-Memory Aggregation: A Candidate Query

When a star schema query like this one is hard-parsed and the optimizer determines that vector transformation is the best possible method for execution, the Oracle optimizer performs the following actions:

It evaluates the query to obtain the list of potential dense keys, a set of numeric key values that are stored as native integers and have a limited domain of values.

It then evaluates these dense keys to determine if they are either dense join keys (which represent every join key with joins columns residing within a single fact table or dimension) or dense grouping keys (which represent all grouping keys residing within a single fact table or dimension).

It will then construct one or more data flow operators (DFOs) — the smallest unit of work for IMA operations — for each dimension. Each DFO is a key vector structure, which is retained within a specially constructed temporary table for this purpose.

Next, it leverages these key vectors to translate each join key in the fact table of the star schema query to its corresponding dense grouping key(s).

Finally, it materializes these join keys to each dimensional temporary table using those dense grouping keys.

Figure 1 illustrates how vector transformation is utilized in this scenario for the query in Listing 1.

Figure 1: In-Memory Aggregation and Vector Transformation

The end result is that fact table data aggregation can be performed within memory much more efficiently because the optimizer can leverage vector transformation operations to identify matching fact and dimension table members. Vector transformation (VT) is more effective for two key reasons:

1. Vector transformation leverages SIMD vector processing to evaluate a common set of column values — for example, those that must be summed together, or counted or averaged — with a single CPU instruction. These operations will be performed at lightning speed within the CPU itself.

2. Vector transformation doesn’t require the use of a bloom filter (BF) to establish which members of each set are truly matching members. (Recall that bloom filters are extremely efficient at discarding nonmembers of both sets and guarantee that a false negative result never occurs; however, they also require the optimizer to perform one final hash join operation to eliminate any false positive matches.)

The Oracle 12.1.0.2 query optimizer is more likely to choose IMA’s VECTOR GROUP BY operation under the following conditions:

- All dimension tables must be populated within the IMCS and at least part of the fact table has been populated within the IMCS.
- The fact table is at least 10 times larger than the dimension tables.
- The fact table must have at least 10 million rows. However, note that this threshold is controlled by a hidden initialization
Controlling and Influencing IMA Behavior

While the query optimizer has free reign to choose IMA over other data aggregation techniques, an astute Oracle DBA can either influence or restrict the optimizer’s decision to leverage IMA via the optimizer hints shown in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Controls</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>VECTOR_TRANSFORM</td>
<td>Regardless of the optimizer’s calculated cost, these hints either enable or disable the vector transformation on the specified query block.</td>
<td></td>
</tr>
<tr>
<td>NO_VECTOR_TRANSFORM</td>
<td>Identifies or excludes the hinted table(s) as part of the fact table during in-memory aggregation processing.</td>
<td></td>
</tr>
<tr>
<td>VECTOR_TRANSFORM_FACT</td>
<td>Identifies or excludes the hinted table(s) as part of the dimension table(s) during in-memory aggregation processing.</td>
<td></td>
</tr>
<tr>
<td>NO_VECTOR_TRANSFORM_FACT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VECTOR_TRANSFORM_DIMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NO_VECTOR_TRANSFORM_DIMS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Optimizer Hints That Influence In-Memory Aggregation

Likewise, an Oracle DBA can also leverage the database initialization parameters shown in Table 2 to influence the optimizer’s choice of vector transformation methods.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Controls</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>_optimizer_vector_min_fact_rows</td>
<td>Specifies the minimum threshold of fact table rows for which IMA will be considered.</td>
<td>10M</td>
</tr>
<tr>
<td>_optimizer_vector_transformation</td>
<td>Enables (TRUE) or disables (FALSE) vector transformation.</td>
<td>TRUE</td>
</tr>
<tr>
<td>_always_vector_transformation</td>
<td>If set to TRUE, the optimizer will always favor vector transformation.</td>
<td>FALSE</td>
</tr>
<tr>
<td>_optimizer_vector_cost_adj</td>
<td>Like OPTIMIZER_INDEX_COST_ADJ, this factor influences the optimizer to consider vector transformation more strongly as the value approaches 10000 or disfavor it as it approaches a value of 1.</td>
<td>100</td>
</tr>
</tbody>
</table>

The resulting EXPLAIN PLAN output in Listing 2 shows an example of how the optimizer chose to leverage vector transformation over Bloom Filters to obtain a more efficient execution plan. Note that it was necessary to set the _optimizer_vector_min_fact_rows initialization parameter to a value of 15,000 at the session level to obtain this result because the default value of 10M rows would have been much too large for the SH.SALES fact table.

Listing 2: In-Memory Aggregation: EXPLAIN PLAN Showing Vector Transformation

Once the database object has been fully populated into the IMCS, the query optimizer can take immediate advantage of the speed at which data in the columnar store format is accessed and execute the statement as an in-memory query. An in-memory query takes advantage of the best features of IMCS to provide excellent query performance. While hybrid columnar compression is aimed at compressing data for efficient storage on spinning disks, IMCS compression is instead optimized for accessing data in memory for speediest retrieval. And because IMCS is able to store multiple values in a single vector structure, in-memory query can leverage SIMD processing to dramatically increase the its performance.

IMA: Vector Transformation Versus Star Transformation

Hopefully, these examples have explained exactly how IMA does what it does, but why IMA offers significant opportunities to improve data
For star transformation to be executed most effectively, however, a special type of index — a bitmap join index (BJI) — must have been created on each dimension table.

warehousing query performance may still be somewhat unclear. So let’s contrast IMA with one of the most common methods for improving query performance in data warehousing systems: star transformation.

Star transformation essentially converts a query’s execution plan into a series of joins between the dimension row sources to first find all possible matching values between those dimensions, and then applies the resulting matches via a hash join to the fact table itself. This technique abrogates the need to join each dimension row source back to the fact table serially so that filtering can be applied; rather, the filtering of all possible values is applied simultaneously.

For star transformation to be executed most effectively, however, a special type of index — a bitmap join index (BJI) — must have been created on each dimension table. Unlike other index structures, each BJI essentially contains a list of all valid join combinations for the columns that link a dimension table to its corresponding fact table. During star transformation, the dimensional BJs are first joined to filter only the matching fact table rows, and that resulting row set is hash-joined back to the fact table to create the final row set. Therefore, it only makes sense to create BJs when their underlying fact and dimension tables are populated infrequently and remain static until the next refresh of the tables’ data.

Vector transformation essentially reproduces these same steps, but with one key difference: the creation of the row set that’s finally used to filter against the fact table is performed completely in memory and — providing that the underlying computing node’s CPUs support SIMD, of course — most of the transformation operations can take advantage of its “software in silicon” capabilities. In addition, BJs are not a prerequisite for vector transformation, and because BJs are relatively expensive to maintain, vector transformation virtually eliminates the need for them.

Finally, it’s important to remember that IMA is essentially an optional execution path that the Oracle query optimizer chooses when all the prerequisites for vector transformation have been met and the resulting optimizer cost is lower than other access methods. As these various IMA examples have demonstrated, it is indeed possible to force the optimizer to use vector transformation via hints and initialization parameters; however, it makes sense to allow the optimizer to make its own decisions based on available cardinality statistics and platform characteristics.

Advanced Index Compression (AIC): Storing More Index Data in a Smaller Space

As we just explained in the case of star schema transformation, some types of indexes can still be quite useful. Of course, they are crucial for maintaining uniqueness and referential integrity within complex relational structures. However, except for these important and valid use cases, I also believe we should use the indexes like an expensive spice: only when needed, and even then sparingly.

Earlier database releases (9i) offered the ability to create a compressed index with the COMPRESS keyword. This type of compression is most valuable when the leading column(s) of a multiple column index contains many repeated values — for example, a leading identifier column of the child table in a parent-child relationship, as in Listing 3.

Listing 3: Creating a COMPRESSed Index (Pre-12.1.0.2)

In this case, the leading column’s value will be stored just once within the index block — much like a prefix — but store the remaining nonrepeating column values within the block normally. This results in a significantly higher number of index row pieces that can be stored within a single index block, and this translates into fewer physical I/Os to retrieve indexed data, especially during a full index scan or fast full index scan operation.

Advanced Index Compression: Benefits

Oracle Database 12.1.0.2 introduced a new method for index compression aptly titled advanced index compression (AIC). AIC essentially extends the same benefits of earlier advanced compression capabilities for segments containing data blocks to segments that contain index blocks. This translates to potentially higher compression ratios even when there is only one indexed column. In addition, indexes on values that are constantly changed during DML will still benefit from AIC because it extends the same improvements of the ADVANCED compression algorithms for row pieces in data segments to index segments, and this may alleviate the need to rebuild indexes on volatile data values over time.

Listing 4 shows how to apply the new 12.1.0.2 AIC features to three columns — ZIPCODE, CITY and STATE — in the SH.EST_SALES table. Note that the benefits of AIC don’t require dropping and recreating the index from scratch; AIC can be applied by simply rebuilding an index with the appropriate keyword:
CREATE INDEX ap.est_sales_zipcityst_idx
ON ap.est_sales(zipcode, city, state)
TABLESPACE ado_cold_idx
COMPRESS ADVANCED LOW;

ALTER INDEX ap.est_sales_zipcityst_idx
REBUILD ONLINE
COMPRESS ADVANCED LOW;

Listing 4: Creating an AIC Index (12.1.0.2),
or Rebuilding an Existing Index For AIC

The Oracle optimizer detects the positive impact of AIC when
performing a full table scan on the indexed values for the new index
that uses COMPRESS ADVANCED LOW. The query shown in Listing 5
can take immediate advantage of the SH.SALES_ZIPCITYST_IDX index
via a FAST FULL SCAN operation, but note the difference in optimizer
estimated performance with and without AIC. Listing 6 and Listing 7
show the EXPLAIN PLAN respectively before and after AIC was applied
to the index. Note that the estimated optimizer cost for this statement
has dropped significantly — from 768 to only 332 — resulting in a
potential improvement of over 131 percent as compared to the
pre-12.1.0.2 AIC version of the same indexed data.

Listing 5: Leveraging an AIC Index: Sample Query

AIC essentially extends the same benefits of earlier advanced
compression capabilities for
segments containing data blocks to
segments that contain index blocks.

---

AIC essentially extends the same
benefits of earlier advanced
compression capabilities for
segments containing data blocks to
segments that contain index blocks.

---

Listing 6: EXPLAIN PLAN Results: Before an AIC
Index Is Available

Listing 7: EXPLAIN PLAN Results: After an AIC Index
Is Available

What’s Next for 12.1.0.2?

It would be wonderful to perform SQL operations against data only
within memory, but sooner or later that data must be retrieved
from a physical device. My next article will discuss how to improve
the performance of physical I/O whenever it’s necessary to read
data from disk using two new features of Oracle Database
12.1.0.2 — attribute clustering and zone mapping — and how
these features work in concert with Oracle’s engineered systems
to provide excellent I/O throughput. 

---

Jim Czuprynski has accumulated over 30 years of experience during his career in
information technology. He has served diverse roles at several Fortune 1000 companies in
those three decades — mainframe programmer, applications developer, business analyst,
and project manager — before becoming an Oracle database administrator in 2001. He is an
Oracle ACE Director and he currently holds OCP certification for Oracle 9i, 10g and 11g.