Renewal

Complete your goals, plan for challenges, reinvent yourself

Features:

Precisely My Point: Leverage Attribute Clustering and Zone Mapping in Oracle Database 12.1.0.2

Securing an Oracle Database Environment with SELinux

DBA 101: Becoming an Oracle DBA

IOUG Press Corner: Boost Your Career Through Writing
The new features of 12.1.02 grant an Oracle DBA the power to:

- Leverage attribute clustering to improve query performance when data is retrieved from HDD devices by storing a table’s data in a specific order — and therefore in closer proximity on disk — for faster access times.

- Implement zone mapping to improve the performance of physical reads when data must be retrieved from HDDs by capture metadata about exactly where data is stored on disk, thus enabling pruning of physical I/O activity.

- Leverage both these feature sets in concert with engineered systems features as an alternative to creating multiple indexes on a table’s columnar data, thereby potentially also improving DML throughput during batch processing as well as OLTP transaction activity.

Data Warehousing, Star Schemas and the Lack of Ordered Data

When I speak with my Oracle DBA colleagues about their most vexing issues, with rare exceptions — say, application workloads that demand extremely quick response time like banking/financial/trading systems — I’ve found that my colleagues aren’t usually worried about the response time of their OLTP applications or even the tables loaded via batch processing. On the contrary, it’s their IT organization’s decision support systems (DSS) and data warehousing (DW) application workloads that continue to demand ever-faster response time.

Ongoing IT industry trends are increasing the demand for ever-faster queries as well:

- As big data and the Internet of Things (IOT) continue to penetrate almost every line of business, analytical queries are required to return meaningful results in almost real time about semi-structured or completely unstructured data.
DSS/DW queries continue to demand more computing resources as the volume, complexity and multidimensionality of data expands. (Intel announced in September 2014 that 90 percent of all data that exists today was created in just the last two years.)

The typical knee-jerk reaction to decreasing query response time — adding indexes to tables in search of faster searches, joins and ordered retrievals — has become untenable as the sheer volume of data grows.

To add to the complexity of these issues, the number of dimensions that need to be analyzed, as well as the detailed data present in those dimensions, continues to expand.

Consider a typical star schema query from Listing 1, which attempts to identify all sales detail for customers located in a specific town in Wisconsin who also purchased products within a specific product category.

Click here for the DDL to create the underlying tables for this query.

Note that several attributes of the Products and Customers dimensions — not the join keys that link each dimension to its fact table — are used as selection criteria:

```
SELECT
  P.prod_category,
  P.prod_subcategory,
  C.cust_state_province,
  C.cust_city,
  SUM(S.quantity_sold),
  SUM(S.amount_sold)
FROM
  sh.zm_sales S,
  sh.zm_products P,
  sh.zm_customers C
WHERE
  S.prod_id = P.prod_id
  AND S.cust_id = C.cust_id
  ... = 'Bay City'
  AND C.cust_state_province = 'WI'
  AND P.prod_category = 'Software/Other'
GROUP BY
  P.prod_category,
  P.prod_subcategory,
  C.cust_state_province,
  C.cust_city
ORDER BY
  P.prod_category,
  P.prod_subcategory,
  C.cust_state_province,
  C.cust_city;
```

Listing 1: Typical Sales History Star Schema Query

In this case, the attributes of the Product dimension (SH.ZM_PRODUCTS) and Customer dimension (SH.ZM_CUSTOMERS) that are required for filtering only the required data are completely unavailable in the Sales fact table (SH.ZM_SALES). An unfortunate fact of most data stored within fact tables is that if the columnar data for the same Product and Customer are not stored closely together (or clustered) within the fact table — not an uncommon situation! — then the query is quite likely to consume considerably more physical I/O when these data need to be retrieved from the fact table.

Of course, it’s quite unreasonable to expect the fact table’s data to be stored in the right order all the time for every query; the data could have been loaded in any order from any number of sources, and even if it’s loaded from a batch process or static data source, there’s no way to guarantee that it’s sorted the best way to satisfy this particular query. An Oracle DBA would probably respond to complaints of poor response time by creating a non-unique index on the column combinations used for this particular query. However, this solution cannot address a crucial fact: the index cannot capture the necessary data because it is only present within the attributes of the Product and Customer dimensions.

Attribute Clustering

Oracle Database 12.1.0.2 offers a potential solution to this performance issue through its new attribute clustering features. Simply put, attribute clustering tends to improve physical I/O performed against data segments by grouping a table’s row pieces into a more appropriate order on physical disk based on how these data are most likely to be queried:

- Searches for data within tables that employ attribute clustering will tend to be significantly faster because that data will be clustered in closer proximity on disk, so this means considerably less physical I/O will be needed to retrieve the same number of row pieces.
- If an index has been defined on the columns most likely to be used for selection criteria, that index’s clustering factor will be significantly lower than if the data was not ordered appropriately.
- Attribute clustering also permits ordering the row pieces within the fact table based on how the data is grouped together in corresponding dimension tables, or even based on the order in which rows will be joined between fact and dimension table(s).
Data that’s been ordered via attribute clustering also tends to improve the data’s compressibility because similar groups of data values will be located closer to each other. — perhaps even within the same database block or extent — and this usually has a positive impact on data that is typically accessed via full table scans. Because of their proximity, clustered data also tends to require fewer CPU cycles to compress.

Attribute clustering can take advantage of advanced Oracle Enterprise System features such as Exadata storage indexes. Finally, data that’s been grouped via attribute clustering can take advantage of in-memory min/max pruning as well as advanced Oracle Enterprise System features such as Exadata storage indexes and zone maps, which we will explore in the second section of this article.

### Linear Versus Interleaved Attribute Clustering

Attribute Clustering offers two different methods of data clustering. The default method — LINEAR — insures that data will be ordered within a table based on the order of the specified column(s) in the CLUSTERING clause for a table or table partition. Listing 2 illustrates the simplest form of linear attribute clustering for table SH.ZM_SALES — clustering on the column values for CUST_ID and PROD_ID:

```sql
ALTER TABLE sh.zm_sales DROP CLUSTERING;
ALTER TABLE sh.zm_sales
ADD CLUSTERING
BY LINEAR ORDER (cust_id, prod_id);
ALTER TABLE sh.zm_sales MOVE ALLOW CLUSTERING;
EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','*ZM_SALES*');
```

Listing 2: Applying LINEAR Attribute Clustering

It’s important to realize that until the ALTER TABLE … MOVE ALLOW CLUSTERING; command is issued, the data in SH.ZM_SALES will remain in its originally loaded order. Once the data has been clustered, however, a simple query retrieving the first 100 rows proves that the data in this table has indeed been physically ordered based on the specified clustering clause. Note the ordering of the rows is evident based on the progression of values for each row’s ROWID, as Listing 3 shows:

```sql
SQL> SELECT ROWID, cust_id, prod_id, quantity_sold, amount_sold
  FROM sh.sales
  WHERE rownum < 101;
```

### Listing 3: Proving LINEAR Attribute Clustering Implementation

The second type of clustering, INTERVAL, applies a special algorithm known as a Z-Order Curve (also known as Morton Order) to cluster related data into discrete groups based on the columns specified. Listing 4 shows how the INTERVAL clustering method is implemented for table SH.ZM_SALES, and Listing 5 shows the resulting row order.

```sql
ALTER TABLE sh.zm_sales DROP CLUSTERING;
ALTER TABLE sh.zm_sales
ADD CLUSTERING
BY INTERVAL ORDER (cust_id, prod_id)
YES ON LOAD
YES ON DATA MOVEMENT
WITHOUT MATERIALIZED ZONEMAP;
ALTER TABLE sh.zm_sales MOVE ALLOW CLUSTERING;
EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','*SALES*');
```

Listing 4: Applying INTERVAL Attribute Clustering
### SQL Query

```sql
SQL> SELECT ROWID, cust_id, prod_id, quantity_sold, amount_sold
    FROM sh.sales
    WHERE rownum < 101;
```

<table>
<thead>
<tr>
<th>ROWID</th>
<th>CUST_ID</th>
<th>PROD_ID</th>
<th>QUANTITY_SOLD</th>
<th>AMOUNT_SOLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAWeZAAKAAABiSAAA</td>
<td>987</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAAB</td>
<td>1660</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAAC</td>
<td>1762</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSABA</td>
<td>1843</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAAE</td>
<td>1948</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAXX</td>
<td>659</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAYY</td>
<td>840</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAZ</td>
<td>949</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAA</td>
<td>1242</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAB</td>
<td>1291</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAC</td>
<td>1422</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAD</td>
<td>1485</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAE</td>
<td>1580</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAZ</td>
<td>14457</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAA</td>
<td>17011</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAA</td>
<td>17566</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAA</td>
<td>17633</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAA</td>
<td>2</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAB</td>
<td>11453</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAB</td>
<td>12783</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAB</td>
<td>15826</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAB</td>
<td>26631</td>
<td>13</td>
<td>1</td>
<td>1232.16</td>
</tr>
<tr>
<td>AAAWeZAAKAAABiSAB</td>
<td>349</td>
<td>13</td>
<td>1</td>
<td>1237.31</td>
</tr>
</tbody>
</table>
```

### Attribute Clustering and Dimensional Joins

While the order of these data’s rows using INTERLEAVED clustering may appear to be distributed in an almost desperately random fashion when compared to LINEAR clustering, these rows are actually arranged according to an ingenious algorithm that allows a series of values to be located quickly and efficiently within several smaller “boxed sets” of rows. Even though the algorithm is almost 50 years old (it was first proposed by G. M. Morton in 1966), it has been proven extremely effective for searches, especially when a B-Tree index structure is used to access these smaller box sets of rows.

Oracle 12.1.0.2 implements INTERLEAVED clustering using what is typically called the Z-Order Curve method, as illustrated in Figure 1. The Z-Order Curve method essentially passes over each “boxed set” of clustered values in a backward “Z” pattern using a series of bits to record the minimum and maximum values within each group of values. This “Z” pattern can be repeated virtually infinitely, but it requires a very small number of bits to record the clustered values.

An interesting analogy for the way that interleaved clustering works is the puzzle of the nine dots, in which the challenge is to draw a line through nine dots arranged in three rows of three dots each, but using only four vertical or horizontal lines without ever lifting pencil off paper. Interleaved clustering essentially attempts to do the same thing, but with (in theory) an infinite number of multiple data points in n-dimensional space instead of just a few dots on a piece of paper.

### Attribute Clustering and Dimensional Joins

Another interesting feature of attribute clustering is the ability to arrange data within a fact table or fact table partition based not on how the data is sorted within the fact table, but instead how it is ordered within one or more corresponding dimensions. Consider Listing 6, which shows an implementation of a dimensional join between the ZM_SALES fact table and one of its corresponding dimensions, ZM_PRODUCTS:

```sql
ALTER TABLE sh.zm_sales DROP CLUSTERING;
ALTER TABLE sh.zm_sales
    ADD CLUSTERING
    sh.zm_sales JOIN sh.zm_products
    ON (sh.zm_sales.prod_id = sh.zm_products.prod_id)
    BY LINEAR ORDER (
        sh.zm_products.prod_category,
        sh.zm_products.prod_subcategory)
    YES ON LOAD
    YES ON DATA MOVEMENT
    WITHOUT MATERIALIZED ZONEMAP;
ALTER TABLE sh.zm_sales MOVE ALLOW CLUSTERING;
EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','ZM_SALES');
```

### Listing 5: INTERVAL Attribute Clustering: Ordered Results

While the order of these data’s rows using INTERLEAVED clustering may appear to be distributed in an almost desperately random fashion when compared to LINEAR clustering, these rows are actually arranged according to an ingenious algorithm that allows a series of values to be located quickly and efficiently within several smaller “boxed sets” of rows. Even though the algorithm is almost 50 years old (it was first proposed by G. M. Morton in 1966), it has been proven extremely effective for searches, especially when a B-Tree index structure is used to access these smaller box sets of rows.

Oracle 12.1.0.2 implements INTERLEAVED clustering using what is typically called the Z-Order Curve method, as illustrated in Figure 1. The Z-Order Curve method essentially passes over each “boxed set” of clustered values in a backward “Z” pattern using a series of bits to record the minimum and maximum values within each group of values. This “Z” pattern can be repeated virtually infinitely, but it requires a very small number of bits to record the clustered values.

An interesting analogy for the way that interleaved clustering works is the puzzle of the nine dots, in which the challenge is to draw a line through nine dots arranged in three rows of three dots each, but using only four vertical or horizontal lines without ever lifting pencil off paper. Interleaved clustering essentially attempts to do the same thing, but with (in theory) an infinite number of multiple data points in n-dimensional space instead of just a few dots on a piece of paper.

### Attribute Clustering and Dimensional Joins

Another interesting feature of attribute clustering is the ability to arrange data within a fact table or fact table partition based not on how the data is sorted within the fact table, but instead how it is ordered within one or more corresponding dimensions. Consider Listing 6, which shows an implementation of a dimensional join between the ZM_SALES fact table and one of its corresponding dimensions, ZM_PRODUCTS:

```sql
ALTER TABLE sh.zm_sales DROP CLUSTERING;
ALTER TABLE sh.zm_sales
    ADD CLUSTERING
    sh.zm_sales JOIN sh.zm_products
    ON (sh.zm_sales.prod_id = sh.zm_products.prod_id)
    BY LINEAR ORDER (
        sh.zm_products.prod_category,
        sh.zm_products.prod_subcategory)
    YES ON LOAD
    YES ON DATA MOVEMENT
    WITHOUT MATERIALIZED ZONEMAP;
ALTER TABLE sh.zm_sales MOVE ALLOW CLUSTERING;
EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','ZM_SALES');
```

### Listing 6: Single Dimensional Join

In this scenario, attribute clustering will use the order of the Product Category and Product Subcategory codes as they are stored in the SH.ZM_PRODUCTS dimension table. The key advantage of this clustering method is that it can essentially pre-group and pre-order rows in the fact table in the same way that one or more queries against the fact and dimension is likely to retrieve them.
The previous scenario only used one dimension for attribute clustering, but it is possible to use more than one dimension as well, as Listing 7 shows. In this scenario, the fact table’s rows will be ordered using the INTERLEAVED (Z-Order Curve) algorithm instead of the default LINEAR method.

```
ALTER TABLE sh.zm_sales DROP CLUSTERING;
ALTER TABLE sh.zm_sales
ADD CLUSTERING sh.zm_sales
JOIN sh.zm_products
ON (sh.zm_sales.prod_id = sh.zm_products.prod_id)
JOIN sh.zm_customers
ON (sh.zm_sales.cust_id = sh.zm_customers.cust_id)
BY INTERLEAVED ORDER (sh.zm_sales.cust_id, sh.zm_sales.prod_id, sh.zm_customers.cust_id, sh.zm_customers.state_province, sh.zm_customers.city, sh.zm_products.prod_category, sh.zm_products.prod_subcategory)
YES ON LOAD
YES ON DATA MOVEMENT
WITHOUT MATERIALIZED ZONEMAP;
ALTER TABLE sh.zm_sales MOVE ALLOW CLUSTERING;
EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','ZM_SALES');
```

Listing 7: Multi-Dimensional Join: Simple

Finally, Listing 8 shows one more attribute clustering scenario: the use of both the Customer and Product dimensions to order the fact table in LINEAR order based on four columns not present in the fact table at all.

```
ALTER TABLE sh.zm_sales DROP CLUSTERING;
ALTER TABLE sh.zm_sales
ADD CLUSTERING sh.zm_sales
JOIN sh.zm_products
ON (sh.zm_sales.prod_id = sh.zm_products.prod_id)
JOIN sh.zm_customers
ON (sh.zm_sales.cust_id = sh.zm_customers.cust_id)
BY LINEAR ORDER (sh.zm_customers.cust_state_province, sh.zm_customers.cust_city, sh.zm_products.prod_category, sh.zm_products.prod_subcategory)
YES ON LOAD
YES ON DATA MOVEMENT
WITHOUT MATERIALIZED ZONEMAP;
ALTER TABLE sh.zm_sales MOVE ALLOW CLUSTERING;
EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','ZM_SALES');
```

Listing 8: Multi-Dimensional Join: Complex

This attribute clustering method is particularly useful when queries against the fact table and these two dimensions typically need to group and order data within a hierarchy of values; in this case, both the State and City attributes of the Customer dimension as well as the major and minor Categories of the Product dimension.

Attribute Clustering Metadata

Oracle Database 12.1.0.2 also provides four new data dictionary views summarized in Table 1 that contain the metadata about which attribute clustering methods have been applied to database objects.

<table>
<thead>
<tr>
<th>Data Dictionary View</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA_CLUSTERING_TABLES</td>
<td>Shows which tables have at least one attribute cluster</td>
</tr>
<tr>
<td>DBA_CLUSTERING.Keys</td>
<td>Displays keys and values which control attribute clustering</td>
</tr>
<tr>
<td>DBA_CLUSTERING_DIMENSIONS</td>
<td>Lists which dimensions are used to control sorting and ordering within fact tables</td>
</tr>
<tr>
<td>DBA_CLUSTERING_JOINS</td>
<td>When joins between dimension and fact tables are employed, identifies the dimensions and columns used to define attribute clustering</td>
</tr>
</tbody>
</table>

Table 1: Attribute Clustering Metadata: Data Dictionary Views

The current state of attribute clustering for the sample tables we’re using for these demonstrations are illustrated in the following queries. Listing 9 lists a query against DBA_CLUSTERING_TABLES and the resulting output that shows which tables are using attribute clustering.

```
SET LINESIZE 100
SET PAGESIZE 20000
COL owner FORMAT A12          HEADING "Table|Owner"
COL table_name  FORMAT A30          HEADING "Table Name"
COL cls_type  FORMAT A12          HEADING "Cluster Type"
COL on_data_formats  FORMAT A08          HEADING "On Data Forms"
COL on_load cls_onld  FORMAT A08          HEADING "On Load Formats"
COL on_datamovement cls_ondm  FORMAT A08          HEADING "On Data Movement Formats"
COL wzmap  FORMAT A08          HEADING "With Zone Map?"
TTITLE "Clustering Attribute Metadata (from DBA_CLUSTERING_TABLES)"
SELECT
  owner, table_name, cls_type, on_data_formats, on_load, on_datamovement, wzmap
FROM dba_cluster
WHERE owner = 'SH'
ORDER BY 1,2;
TTITLE OFF
```

Listing 9: Tables Using Attribute Clustering

For example, Listing 10 displays a query against DBA_CLUSTERING_ KEYS and the resulting output that shows which columns implement the ordering of data within fact tables using attribute clustering:
Listing 10: Attribute Clustering Keys

The query against DBA_CLUSTERING_DIMENSIONS and its resulting output in Listing 11 shows which dimension tables are used to enforce attribute clustering on their corresponding fact tables.

Listing 11: Attribute Clustering Dimensions

Finally, Listing 12 lists a query against DBA_CLUSTERING_JOINS and its resulting output that shows which joins between fact and dimension tables to implement attribute clustering.

Listing 12: Attribute Clustering Joins

Zone Maps: Hit 'Em Where They Ain’t

While attribute clustering actually stores related data in an appropriate pattern based on the desired column ordering of the values, the zone mapping features of Oracle Database 12.1.0.2 actually does the inverse: It accurately ascertains exactly where data is stored on I/O devices and thus eliminates unnecessary physical I/O processing through Oracle 12c Database’s proprietary in-memory min/max pruning features.

Zone maps are database objects specifically designed to locate table and table partition data with minimal physical I/O. Zone maps also pair nicely with Exadata storage indexes and are therefore only useful on selected Oracle Enterprise hardware.

Creating Zone Maps

Zone maps can be created either automatically during the specification of attribute clustering for a table or partition, or they can be created after attribute clustering has been implemented. In fact, attribute clustering is not even required to implement a zone map, but as we’ll see a bit later, it makes sense to leverage both features simultaneously. Listing 13 shows how a zone map can be created during attribute clustering creation:
Listing 13: Creating a Zone Map Automatically During Attribute Clustering Implementation

However, only one zone map can exist per table at any time, so any existing zone map must be dropped before a new one can be created, as Listing 14 illustrates:

```
SQL> CREATE MATERIALIZED ZONEMAP sh.mzm_sales
       TABLESPACE ado_cold_data
       REFRESH ON LOAD DATA MOVEMENT
       AS
       SELECT
             SYS_OP_ZONE_ID(S.ROWID)
               ,MIN(cust_id)
               ,MAX(cust_id)
               ,MIN(prod_id)
               ,MAX(prod_id)
       FROM sh.zm_sales S
       GROUP BY sys_op_zone_id(S.ROWID);
ERROR at line 1:
ORA-31958: fact table "SH"."ZM_SALES" already has a zonemap "SH"."MZM_SALES" on it
SQL> DROP MATERIALIZED ZONEMAP sh.mzm_sales;
Zone map dropped.
SQL> DROP CREATE MATERIALIZED ZONEMAP sh.mzm_sales
       TABLESPACE ado_cold_data
       REFRESH ON LOAD DATA MOVEMENT
       AS . . .
Zone map created.
```

Listing 14: Creating a Zone Map Manually When One Already Exists

Just as with attribute clustering, metadata about zone maps is retained within the database’s data dictionary. Table 2 summarizes the two new data dictionary views in Oracle Database 12.1.0.2 that contain the metadata for zone map objects.

<table>
<thead>
<tr>
<th>Data Dictionary View</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA_ZONEMAPS</td>
<td>Describes Zone Map metadata, including when a Zone Map was refreshed</td>
</tr>
<tr>
<td>DBA_ZONEMAP_MEASURES</td>
<td>Describes functional methods the Zone Map has utilized to map out data (e.g. MIN/MAX to enable pruning)</td>
</tr>
</tbody>
</table>

Table 2: Zone Maps Metadata: Data Dictionary Views

Listing 15 illustrates how to query the DBA_ZONEMAPS data dictionary view to see information about which zone maps exist, and Listing 16 shows the results of a query against the DBA_ZONEMAP_MEASURES view to identify which measures — in other words, the functions that were used to obtain the boundaries of each zone map — and which columns of underlying database tables were used to construct those boundaries.
Zone Mapping Metadata
(from DBA_ZONEMAPS)

<table>
<thead>
<tr>
<th>Zone Map</th>
<th>Zone Map</th>
<th>Fact Table</th>
<th>Fact Table</th>
<th>Zone Map</th>
<th>Scale</th>
<th>Clst?</th>
<th>Enabled?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH</td>
<td>MZM_SALES</td>
<td>SH</td>
<td>ZM_SALES</td>
<td>10</td>
<td>NO</td>
<td>NO</td>
<td>ENABLED</td>
</tr>
</tbody>
</table>

Invalid? Stale? Unusable State
----------- --------- ----------- -------
NO         NO        NO          VALID

Zone Mapping Metadata
(from DBA_ZONEMAP_MEASURES)

<table>
<thead>
<tr>
<th>Pos</th>
<th>Zone Map</th>
<th>In Aggregate</th>
<th>Column Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH</td>
<td>MZM_SALES</td>
<td>2 MIN</td>
<td>MIN_1_CUST_ID</td>
</tr>
<tr>
<td>SH</td>
<td>MZM_SALES</td>
<td>3 MAX</td>
<td>MAX_1_CUST_ID</td>
</tr>
<tr>
<td>SH</td>
<td>MZM_SALES</td>
<td>4 MIN</td>
<td>MIN_2_PROD_ID</td>
</tr>
</tbody>
</table>

Building Non-Default Zone Maps

Interestingly, Oracle 12.1.0.2 also provides the capability to build your own zone maps using functions of your choice. Depending on your perspective, this may either be a thrilling experiment or a daunting task; however, it appears that the primary reason for this flexibility is to allow the construction of more complex zone mapping structures in later releases of the database. The example in Listing 18 illustrates how to reconstruct the SH.MZM_SALES zone map using appropriate minimum and maximum values for four columns in the SH.PRODUCTS and SH.CUSTOMERS dimension tables. Note the call to the SYS_OP_ZONE_ID function to obtain an appropriate value for that mandatory column of the Zone Map:

Oracle 12.1.0.2 also provides the capability to build your own zone maps using functions of your choice.
Listing 18: Building a Zone Map with Aggregate Functions

**Putting It All Together: Attribute Clustering Plus Zone Mapping**

At this point in our discussion, it shouldn’t come as a great surprise that attribute clustering and zone maps work quite nicely together. They are especially effective at reducing the requirement to create alternative B-Tree or bitmap indexes on one or more columns in a table in a possibly futile attempt to improve the performance of just a few queries at the cost of DML performance.

For example, Listing 19 shows the resulting execution plan for our original query in Listing 1. Note that the query was able to take advantage of the SH.MZM_SALES zone map that was created via the code in Listing 18, the plan shows the application of the zone map to filtering operations against the SH.ZM_SALES table through the SYS_ZMAP_FILTER function:

**Listing 19: Execution Plan for Query Leveraging Zone Mapping**

**Leveraging In-Memory Min-Max Partition Pruning**

Pairing attribute clustering with zone maps also offer another intriguing performance improvement opportunity: the ability to leverage in-memory partition pruning based on the range of minimum and maximum values that zone maps can track to minimize physical I/O as storage is searched for the appropriate values. To illustrate the power of combining attribute clustering and zone maps, consider table AP.RANDOMIZED_PARTED, which was created and loaded via the code in Listing 20:

**Listing 20: Creating a Partitioned Table to Demonstrate In-Memory Min-Max Partition Pruning**
The query in Listing 21 results in the optimizer choosing quite a different approach to accessing these data, as the execution plan in Listing 22 shows:

```
EXPLAIN PLAN FOR
SELECT
    MIN(key_id), MAX(key_id), COUNT(*)
FROM ap.randomized_parted
WHERE key_date BETWEEN TO_DATE('2014-01-01','YYYY-MM-DD')
    AND TO_DATE('2014-09-30','YYYY-MM-DD')
    AND key_sts < 50;
SELECT plan_table_output
FROM TABLE(DBMS_XPLAN.DISPLAY(FORMAT => 'BASIC PREDICATE PARTITION'));
```

Listing 21: Leveraging Zone Mapping and Attribute Clustering to Prune Partitions

```
Plan hash value: 4272865020
------------------------------------------------------------------------
----------------- |
Id  | Operation                  | Name              | Pstart| Pstop |
------------------------------------------------------------------------
----------------- |
   0 | SELECT STATEMENT           |                   |       |       |
   1 |  SORT AGGREGATE            |                   |       |       |
   2 |   PX COORDINATOR           |                   |       |       |
   3 |    PX SEND QC (RANDOM)     | :TQ10000          |       |       |
   4 |     SORT AGGREGATE         |                   |       |       |
   5 |      PX BLOCK ITERATOR     |                   |KEY(AP)|KEY(AP)|
*  6 |       TABLE ACCESS STORAGE | RANDOMIZED_PARTED |KEY(AP)|KEY(AP)|
     |                         | FULL WITH ZONEMAP |
------------------------------------------------------------------------
```

Predicate Information (identified by operation id):
---------------------------------------------------
6 - storage("KEY_STS"<50 AND "KEY_DATE"<=TO_DATE(' 2014-09-30
00:00:00','yyyy-mm-dd hh24:mi:ss'))
   filter(SYS_ZMAP_FILTER('/* ZM_PRUNING */ SELECT "ZONE_ID$", CASE
   WHEN BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE CASE WHEN
   (zm."MIN_1_KEY_STS" >= :1) THEN 3 ELSE 2 END END FROM
   "AP"."ZM_RANDOMIZED_PARTED" zm WHERE zm."ZONE_LEVEL$"=0
   ORDER BY zm."ZONE_ID$",SYS_OP_ZONE_ID(ROWNID),50)<3 AND
   "KEY_STS"<50 AND "KEY_DATE"<=TO_DATE(' 2014-09-30
00:00:00','yyyy-mm-dd hh24:mi:ss'))

Listing 22: Resulting Execution Plan Leveraging Partition Pruning

Oracle Database Release 12.1.0.2: Concluding the Series

The last four articles in this series have delved deeply into many of the new feature sets of Oracle Database Release 12.1.0.2. Though many IT organizations are doubtlessly waiting for Oracle Database Release 12.2 until beginning their foray into Oracle Database 12c, hopefully these articles have made a convincing case that there’s no time like the present to begin exploring 12.1.0.2’s key features, especially the potentially dramatic increase in database application performance through In-Memory Column Store (IMCS).

References


Oracle Database In-Memory. (2014, October). Retrieved February 15, 2015, from Oracle Database 12c Online Documentation Library.


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.