executives and front-line employees often make decisions based on data that has multiple independent attributes or dimensions. Online analytical processing (OLAP) is the process of analyzing this type of dimensional data. Two primary methods of implementing OLAP are through multi-dimensional OLAP (MOLAP) and relational OLAP (ROLAP). Though the purpose of analyzing the data is the same for both methods, the architecture and processes differ: MOLAP stores the information in a cube, or a specialized pre-calculated data store, while ROLAP uses a standard relational data store.

MOLAP storage generally provides more rapid query response, but it might not be feasible to regularly move enormous amounts of data to populate the MOLAP store. As an alternative, you can define your cubes to use ROLAP mode, which enables a scalable solution. Another advantage to using ROLAP is that it can be built in a fraction of the time it takes to populate a MOLAP cube.

Implementing ROLAP cubes offers end users a simple solution that enables more dimensions, history, detail and faster deployments, while providing fast query responses. Through ROLAP, cubes can provide access to an enormous amount of data and perform queries that meet or exceed most business requirements.

The MOLAP challenge
Most of the time it takes to process a MOLAP cube is spent transferring data and populating the MOLAP cache (including aggregates). The data is transferred to a cube-building process that resides on a middle server or on a complex of servers. However, transferring large amounts of data can take a significant amount of time, and moving data from one server to another can introduce challenges. These challenges are applicable in any cube implementation as the MOLAP environment matures to deliver deeper and wider analytics.

Leveraging the Teradata aggregate join index (AJI) feature will optimize ROLAP performance. An AJI is an aggregated result set saved as an index in the database. It is transparent to end users and business intelligence (BI) administrators, and it is used automatically by the Teradata Optimizer.

By building AJIs on the Teradata Database, the data transfer and cube build is replaced with high-speed index builds. These indexes build in a fraction of the time it takes to build a MOLAP cube.
Building AJIs for ROLAP solutions

The cube uses a dimension map such as the one shown in table 1 to define the dimensions and levels accessible in the cube. The dimensional map is based on the schema/semantic layer database (see figure, page 59).

To deliver a timely, optimal and performant ROLAP solution, the following Teradata physical database design is recommended in a virtual or physical semantic layer database as shown in the figure. These physical table stipulations within the database should be considered:

- Snowflake or third normal form (3NF) models are recommended, but the solution can be implemented on a star schema.
- Primary and foreign keys are not compressible.
- Foreign keys are all defined as NOT NULL.
- Keep your AJI lean. Only place foreign key columns in the AJI. Name and/or description columns will result in a larger AJI that takes longer to build and maintain.
- Statistics should be collected on all primary key and foreign key relationship columns. This will assist in the AJI build and optimizer query plans.
- Dimension table primary key columns are defined as unique by either the UNIQUE constraint, unique primary index or unique secondary index.
- Implement referential integrity (RI). RI can be defined with no-check option given integrity exists within your data. Most data warehouses implement integrity checks within their low processes.

Table 1: Dimensional map

<table>
<thead>
<tr>
<th>Time</th>
<th>Products</th>
<th>Brands</th>
<th>Business</th>
<th>Channel</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>Product category</td>
<td>Brand category</td>
<td>Business type</td>
<td>Channel type</td>
<td>Business unit</td>
</tr>
<tr>
<td>Quarter</td>
<td>Product</td>
<td>Brand</td>
<td>Business</td>
<td>Channel</td>
<td>Organization</td>
</tr>
<tr>
<td>Month</td>
<td>Product</td>
<td>Brand</td>
<td>Business</td>
<td>Channel</td>
<td>Organization</td>
</tr>
<tr>
<td>Day</td>
<td>Product</td>
<td>Brand</td>
<td>Business</td>
<td>Channel</td>
<td>Organization</td>
</tr>
</tbody>
</table>

This is a map, based on the star/snowflake schema in the figure on page 59, of the dimensional model for a cube solution defining all of the dimensions that will be accessible in the cube and their hierarchies.

Table 2: Dimensional map with broad AJI for Teradata

<table>
<thead>
<tr>
<th>Time</th>
<th>Products</th>
<th>Brands</th>
<th>Business</th>
<th>Channel</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>Product category</td>
<td>Brand category</td>
<td>Business type</td>
<td>Channel type</td>
<td>Business unit</td>
</tr>
<tr>
<td>Quarter</td>
<td>Product</td>
<td>Brand</td>
<td>Business</td>
<td>Channel</td>
<td>Organization</td>
</tr>
<tr>
<td>Month</td>
<td>Product</td>
<td>Brand</td>
<td>Business</td>
<td>Channel</td>
<td>Organization</td>
</tr>
<tr>
<td>Day</td>
<td>Product</td>
<td>Brand</td>
<td>Business</td>
<td>Channel</td>
<td>Organization</td>
</tr>
</tbody>
</table>

The columns above the red line should be used in creating a broad AJI.

ALTER TABLE FACT_ADD foreign key
( SALES_CENTER_ID) references with no check option
SALE_CENTER ( SALES_CENTER_ID)

(Note: The above RI relationship is for a star design—i.e., FACT to one dimension table with lowest level. Otherwise, for snowflake design, RI(s) must be defined for each higher-level roll up where each level is in its own table.)

Address the lowest levels in your cube that are not in your AJI by utilizing primary index (PI), secondary index and partitioned primary index (PPI) to optimize detail data access.

Secondary indexes on columns that correspond to low level members in the dimensional model will provide fast access to rows in your transaction/FACT table. This will enable you to eliminate these columns/values from the AJI, thus making the AJI smaller while still providing access to this level. An example of this would be PRODUCT_ID. The PRODUCT_ID column is normally used in filtering and slicing cubes, and is often a high cardinality column, a good candidate for a secondary index.

PPIs allow a table to be partitioned on columns of interest while retaining the traditional use of the PI for data distribution and efficient access when PI values are specified in the query. A good candidate for PPI would be the day level within your time dimension. Most cubes do not provide access to day-level data since it is too costly to bring that level of detail into a cube. The same holds true with AJIs; it may be too costly to include day within the AJI, but fast access can be provided to day-level detail using PPI. This will enable larger cubes to be defined.

Another candidate for PPI may be a regional ID such as Branch_ID. If your geographic dimension has many members that can be expressed within a PPI clause, then it may be better to use it for the partitioning scheme. This geographic ID also fits in well
if you want to provide regional access to users, such as branch managers, from a single relational cube.

**AJI strategy**

Determining the columns to participate in the AJI is very important, but can be challenging. A good start would be to draw a red line across your dimensional model one level up from the lowest level of each dimension. (See table 2, page 60.) This is called a broad AJI. Single-level dimensions, such as Channel type in this example, are the exception. Since there is no higher level than Channel type in the dimension, it should be included in the broad AJI definition.

A good rule for selecting columns in the AJI definition is to include low cardinality columns in the AJI. High cardinality columns are good candidates for secondary indexes on the FACT table and should be excluded from the AJI. High cardinality columns that are defined within the AJI will increase the size of the AJI, thus affecting performance.

For larger cubes that contain more than 40 dimensions, it may be necessary to eliminate seldom-used dimensions from the AJIs. This will ensure that the highest performance is given to navigations that are most often used, as seldom-used navigations will run more slowly. Most business users are willing to accept this trade-off given that they are most likely getting more detail, more dimensions and timelier data with a ROLAP solution.

This is a good initial approach to creating an AJI. As DBAs gain more experience and better understanding of the types of analyses end users are requesting, they can determine more appropriate AJIs to create—whether to build an AJI on a specific subset of dimension, for instance, or whether to build an AJI across all dimensions and at what dimensional level.

**Defining AJI**

As an AJI is created, the semantic layer database should be referenced (see figure, page 59) and the following considerations understood:

- Once the SQL is determined, wrap the CREATE JOIN INDEX and PRIMARY INDEX syntax and execute the DML statement via Teradata Queryman or Winddi. Creation time for an AJI will depend on size of the tables and system usage.
- Notice that the ak.Area_Id referenced in the data definition language (DDL) is from the foreign key value from the SALES_CENTER table (the lowest dimension), not from the AREA table. Hence, unlike star dimensions, in the semantic layer database the higher levels in the AJI definition do not need to be included for the Optimizer to rewrite/use the AJI.
- Whether star, snowflake or 3NF is being used, it is recommended to only place foreign key values (IDs) in the AJI. Placing other values, such as descriptions and attributes, may speed up query performance, but it will increase the size and time to build your AJI. The Teradata Optimizer will take care of SQL requests with Desc or Name by joining to the dimension table on the foreign key value and then aggregating on the attribute or description field.
- The Channel and Business dimensions are star single tables with a one-level hierarchy. Single-level dimensions such as these are the exception. Since there is no higher level in the dimension, it is included in the DDL previously mentioned. Notice that ad.Channel_Id is from the foreign key value from the FACT table.

**What is an AJI?**

An aggregate join index (AJI) is an aggregated result set saved as an index in the database. The AJI will be used automatically by the Teradata Optimizer when like columns and aggregates are made frequently within a query plan.

```sql
CREATE JOIN INDEX AJI_Example ,NO FALLBACK ,CHECKSUM = DEFAULT AS
SELECT COUNT(*) (FLOAT, NAMED CountStar ) ,
    ae.Brand_Category_Id ,
    ac.Product_Category_Id ,
    ad.Business_Type_Id ,
    ad.Channel_Id ,
    ak.Area_Id ,
    al.Year ,
    al.Quarter ,
    al.Month ,
    SUM(ad.Sales ) (FLOAT, NAMED SALES )
FROM
    Product ac ,
    Fact ad ,
    Brand ae ,
    Sales_Center ak ,
    Time al
WHERE
    (((ad.product_id = ac.product_id ) AND
    (ad.brand_id = ae.brand_id )) AND
    (ad.sale_center_id = ak.sale_center_id ))
```
As DBAs gain more experience and better understanding of the types of analyses end users are requesting, they can determine the appropriate types of AJIs to create.

Verifying relational queries

It's always a good idea to check your relational queries against your defined AJI. To do so, capture the SQL request via Teradata Database Query Log access logs. Then check the request using the Teradata Explain command to ensure the AJI is called in the query plan. (See the Explain plan, right.)

If the AJI does not provide the desired performance, other AJIs can be built to provide faster performance. This can be accomplished by creating AJIs at higher levels than the first AJI, or by removing less-often used dimensions from broad AJIs. Note: Higher-level AJIs or subset of dimensions of a broad AJI, will benefit from the first AJI and build in a fraction of the time it took to build the first one.

After all indexes are created, structures will provide fast query performances for a variety of OLAP queries:

- **The broad AJI** for most frequently used access paths
- **The secondary indexes** on high cardinality FACT columns
- **The PPI** on the DATE column in the FACT table

The only types of queries that are not accounted for in these structures are those that select low-level dimension members across multiple dimensions without qualifying values. An example is, “Give me the SUM of sales by DAY, by PRODUCT, by SALE CENTERS with no qualifications (WHERE criteria).” This request would result in many rows being returned to the client and would not be considered an OLAP query. The client would most likely be transferring a bulk amount of data to a PC for analysis using another tool.

Explain

1) First, we lock a distinct EXAMPLE "pseudo table" for read on a RowHash to prevent global deadlock for EXAMPLE.AJI_EXAMPLE.
2) Next, we lock EXAMPLE.AJI_EXAMPLE for read.
3) We do an all-amps SUM step to aggregate from EXAMPLE.AJI_EXAMPLE by way of an all-rows scan with no residual conditions, and the grouping identifier in field 1. Aggregate Intermediate Results are computed globally, then placed in Spool 3. The size of Spool 3 is estimated with no confidence to be 1,040 rows. The estimated time for this step is 0.18 seconds.
4) We do an all-amps RETRIEVE step from Spool 3 (Last Use) by way of an all-rows scan into Spool 1 (group_amps), which is built locally on the AMPS. The size of Spool 1 is estimated with no confidence to be 1,040 rows. The estimated time for this step is 0.01 seconds.
5) Finally, we send out an END TRANSACTION step to all AMPS involved in processing the request.

Many opportunities with AJI

This is one approach to using the Teradata Database AJI feature. While MOLAP cubes present some challenges, leveraging AJI in ROLAP enables users to create deeper and wider analytics. By expanding on the possibilities presented through AJI, including the use of secondary indexes and PPIs, the many combinations and various AJI constructs can greatly improve your OLAP experience.