

Implementing AJs for ROLAP

By: Carlos Bouloy,
Senior Consultant,
Teradata

Implementing AJIs for ROLAP

Table of Contents

<i>Executive Summary</i>	2
<i>Introduction</i>	3
<i>Steps for Building Aggregate Join Indexes for ROLAP Cubes</i>	3
<i>Implementing a ROLAP Cube from a Normalized Model</i>	7
<i>Conclusion</i>	8
<i>About the Author</i>	8

Executive Summary

Building, updating, maintaining, and enhancing MOLAP cubes can be difficult. Things such as the time it takes to build a cube, limited dimensionality, limited history, and limited detail all are major issues associated with MOLAP cubes. Teradata has an alternative to alleviate these issues. Defining your cubes as ROLAP will enable a scalable cube solution that will result in cubes that are larger in dimensionality, detail, and history that are built in a fraction of the time it took to build a MOLAP cube. This white paper describes how to build and implement ROLAP cubes.

Implementing AJIs for ROLAP

Introduction

The time it takes to build a MOLAP cube is spent mostly in transferring data. The data are transferred to a cube building process that resides on a middle server or complex of servers. In order to reduce the cube build times, it is often recommended to reduce the amount of data being transferred. Hence, fewer dimensions, less detail, and less history. However, this results in less information for the business user to analyze.

Building aggregate join indexes (AJI) on Teradata® Database eliminates the data transfer and replaces it with high-speed index builds. These indexes build in a fraction of the time it takes to build a cube. Implementing ROLAP cubes with Teradata Database offers a simple ROLAP solution that enables more dimensions, more history, greater detail, and much faster cube builds while still providing MOLAP-like query responses.

Time	Product	Vendor	Location	Customer
Year	Dept	Parent Company	Region	Customer Type
Quarter	Class	Company	Division	
Month	SKU		Area	
Week			Branch	
Day			Store	

Figure 2. Dimensional Model

- B. To date, the customers that have implemented the AJI ROLAP solution have built their AJIs from a star schema data mart that they built on Teradata Database. This is implemented as insert selects into OLAP star tables after the customer's batch updates were complete. The recommended model for the star schema is a snowflake.

The AJI solution could have also been built from the customer's 3NF tables, but this would require taking the load processes into account, most likely resulting in a change of load procedures at the customer site. The last section of this paper discusses the issues of building AJIs from 3NF tables.

- C. Teradata Database V2R5.1 or greater is highly recommended for implementing AJI ROLAP solutions. V2R5.1 will enable an AJI ROLAP solution to be built with few AJIs. This is because enhancements made to the Teradata Optimizer in V2R5.1 enable it to recognize referential integrity in selecting AJIs. There is a good chance that only one AJI would be necessary for a cube. There are many factors that will affect cube performance; therefore, more AJIs may be necessary. These factors are:

- System size
- System utilization
- Dimension cardinalities
- Amount of aggregation from leaf level data to next higher dimensional levels
- AJI size
- Partitioning on leaf level data

MOLAP – ROLAP Comparison		
	MOLAP	ROLAP
Dimensions	38	38/39
Measures	24	24
History	5 years	5 years
Detail	Month	Day
Build time	19 hours to cube build	2 minutes AJI build
Cube size	3 GB	200 MB
Query Response Times	5 Seconds	90% of queries were faster than MOLAP

Figure 1 shows a comparison of a MOLAP solution and a ROLAP solution.

Steps for Building Aggregate Join Indexes for ROLAP Cubes

This section identifies some of the steps and suggestions for building AJIs for ROLAP cubes.

- A. Build a dimensional model for the cube solution. The dimensional model would look similar to Figure 2. It defines all the dimensions that will be accessible in the cube and their hierarchies.

Implementing AJIs for ROLAP

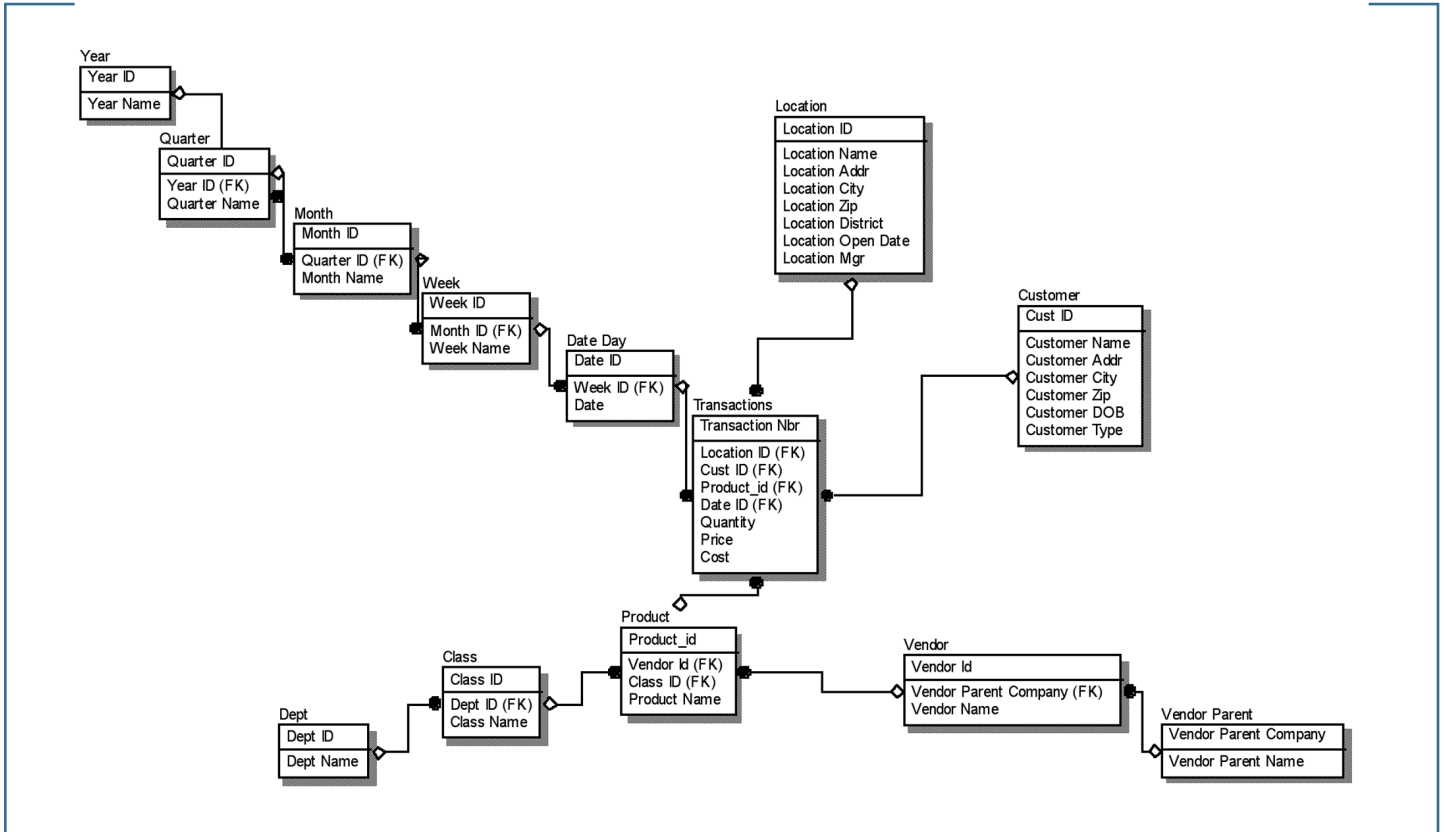


Figure 3. Logical Star Schema for the dimensional model in Figure 2.

D. The physical table stipulations for Figure 3 are:

- A snowflake model is recommended over a star model.
- All foreign keys must be defined as not null.
- All primary and foreign keys are not compressable.
- All dimension table primary keys are defined as unique utilizing the UNIQUE constraint or the primary key is defined as a UNIQUE PRIMARY INDEX.
- Collecting statistics on all primary key/foreign key relationship columns will help to optimize the aggregate join index

build. The statistics will also help to optimize queries that execute against the base tables.

- Referential integrity must be implemented on the primary key/foreign key columns. The referential integrity can be defined with the check (a.k.a., hard RI) or no check option (a.k.a., soft RI).

Example 1.

```
ALTER TABLE PRODUCT
ADD foreign key (CLASS_ID) references
with no check option CLASS (CLASS_ID);
```

Implementing AJIs for ROLAP

E. Take advantage of partitioning for transaction table accesses.
 A good candidate for OLAP cubes is the DATE column, since it is present in most cubes. MOLAP cubes are challenged in enabling a user to access day level data from within the cube due to the size the cube would be if day were included. Day level data are usually supported via drill-through queries. Partitioned primary index on the date column will enable fast access to day level queries in ROLAP.

F. Determine the columns that will participate in the broad AJI.
 A good start would be to draw a red line across your dimensional model one level up from the lowest level of each dimension. Single level dimensions, such as Customer in this example, are the exception. There is no higher level than Customer Type in the dimension so it should be included in the broad AJI definition. See Figure 4.

- a. A good rule of thumb for including 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. Exclude them from the AJI. High cardinality columns that are defined within the AJI will increase the size of the AJI, thus affecting performance. See Figure 5.
- b. For larger cubes that contain many dimensions, greater than 15, it may be necessary to eliminate seldom used dimensions from the AJIs. This will ensure that the highest performance is given to most often used navigations. Seldom used navigations will run slower. Most business users are willing to accept this trade-off given that they are most likely getting more detail, more dimensions, and more timely data with this ROLAP solution.

Time	Product	Vendor	Location	Customer
Year	Dept	Parent Company	Region	Customer Type
Quarter	Class	Company	Division	
Month	SKU		Area	
Week			Branch	
Day			Store	

Figure 4. Broad AJI Definition

Time	Product	Vendor	Location	Customer
Year	Dept	Parent Company	Region	Customer Type
Quarter	Class	Company	Division	
Month	SKU		Area	
Week			Branch	
Day			Store	

Figure 5. Broad AJI (Improved)

Implementing AJIs for ROLAP

G. The DDL for the aggregate join index in Figure 5 is:

```
CREATE JOIN INDEX olap_aji1 as
```

```
SELECT
```

```
  b.Year,  
  b.Quarter,  
  c.Month,  
  d.Week,  
  i.Dept,  
  i.Class,  
  f.Parent Company,  
  f.Region,  
  f.Division,  
  f.Area,  
  f.Branch,  
  a.Customer_Type,  
  SUM(Quantity)  
  SUM(Price)  
  SUM(Cost)
```

```
FROM
```

```
  Transactions a,  
  Quarter b,  
  Month c,  
  Week d,  
  Date_Day e,  
  Location f,  
  Product h,  
  Class i,  
  Vendor j
```

```
WHERE
```

```
  a.date_id = e.date_id and  
  e.week_id = c.week_id and  
  c.month_id = b.month_id and  
  a.store_no = f.store_no and  
  a.product_id = h.product_id and  
  h.class_id = i.class_id and  
  h.vendor_id = j.vendor_id
```

```
GROUP BY
```

```
  b.Year,  
  b.Quarter,  
  c.Month,  
  d.Week,  
  i.Dept,  
  i.Class,  
  f.Parent Company,  
  f.Region,  
  f.Division,  
  f.Area,  
  f.Branch,  
  a.Customer_Type
```

```
PRIMARY INDEX (Week, Class, Branch, Customer_Type);
```

Note: the above DDL includes the higher level dimension members. This is done to provide optimal performance in the pure star model. Once a snowflake model is implemented and partial group-bys are supported with AJIs, it will not be necessary to include the higher level dimension members in the AJI definition.

H. After all indexes are created, fast query performance for a variety of OLAP queries is provided by structures in place including:

- a. The broad AJI.
- b. The secondary indexes.
- c. The partitioned primary index on the date column.

The only types of queries that are not accounted for in these structures are queries that select low level dimension members across multiple dimensions without qualifying values. An example of this would be “Give me the SUM of sales by DATE, by STORE, by SKU with no qualifications (WHERE criteria).” This business query would result in MANY rows being returned to the client and would not be considered an OLAP query. The user would most likely be transferring a bulk amount of data back to his PC for analysis using another tool.

Implementing AJIs for ROLAP

- I. As of Teradata V2R6.2, aggregate join indexes will have the capability of being defined having a partitioned primary index. PPI for AJIs will assist in query performance and AJI maintenance. Prior to V2R6.2, an AJI can be created as a value ordered AJI by adding an ORDER BY clause to the end of the SQL statement within your AJI DDL. A value ordered AJI will provide good query benefits much like PPI does for base tables except without the maintenance benefits. A good candidate column for the ORDER BY clause would be a member level from the Time dimension such as WEEK_ID, MONTH_ID, or YEAR_ID.
- J. Executing an EXPLAIN on a query will show if the aggregate join index is being used. Unless the AJI that was built is very large, it is usually evident from a slow responding query that the AJI is not being used. If the first broad AJI is too large, then another higher level AJI can be built to support higher level queries. Compare Figure 6 to Figure 5. Note, the next higher level broad AJI will benefit from the first AJI that was built and will build in a fraction of the time it took to build the first one.

Time	Product	Vendor	Location	Customer
Year	Dept	Parent Company	Region	Customer Type
Quarter	Class	Company	Division	
Month	SKU		Area	
Week			Branch	
Day			Store	

Figure 6. Higher level AJI

- K. Priority scheduling will need to be implemented for the users who are accessing the AJI and OLAP star data structures within Teradata Database. We recommend that you implement this using milestones in Teradata Priority Scheduler. The queries that hit the AJI, secondary index, or PPI, will perform quickly and not use much resource. Therefore, give these queries a high priority. It is still possible for the user to select a large

volume of detail, and these queries may use a lot of resources. This cannot be controlled. The priority scheduling will handle this by giving the query a high priority initially, and then reduce the priority the longer it is running in the database.

Implementing a ROLAP Cube from a Normalized Model

Many of the steps will remain the same for building a hollow cube from a normalized model. However, building the AJIs from your normalized tables will affect your load processes due to issues with AJIs. Teradata Multiload not supporting aggregate join indexes will most likely be the biggest issue as most customers load their data on a nightly basis using Multiload. Following are the different load scenarios to consider:

- A. Multiload into normalized tables, and then build AJIs from the normalized tables. This will add time to your overall load processes. This may be alright as long as the mload plus the aggregate build fits within your batch window. We recommend that statistics are collected on the JOIN columns and PRIMARY INDEXES of the aggregate join index DDL to get an efficient EXPLAIN plan. This will help to minimize the amount of time it takes to create the aggregates. Initial implementations of aggregate builds on production data have resulted in AJIs that build in less than one hour.
- B. Teradata T pump into normalized tables. This will avoid the dropping and recreating of the AJIs. As updates are executed against the base table, the Teradata Database will automatically update any AJI that has the base table defined within its CREATE JOIN INDEX DDL. A thorough performance analysis will need to be executed to determine if the extra updates to the AJI can be supported in a timely manner by T pump.
- C. Teradata Fastload into staging tables, and then insert select into base tables. This should be the most optimal solution as far as updating all tables and indexes involved, but this will require a rewrite of existing load processes.

Implementing AJIs for ROLAP

Given the updating of data can be supported for the aggregate data, the presentation of data will need to be determined. Most OLAP tools work well with star schemas. Creating a star representation of data using views of the normalized tables is easily do-able. The same SQL that is used for the AJI DDL can be used for the SELECT portion of the CREATE VIEW DDL. Or tools that support normalized data model queries can simply work off of the normalized tables.

Conclusion

Currently, data warehouses are predominantly built using RDBMSs. If you have a warehouse built on a relational database, and you want to perform OLAP analysis against it, ROLAP is a natural fit. This isn't to say that MOLAP can't be a part of your data warehouse solution. It's just that MOLAP isn't currently well-suited for large volumes of data (10-50GB is fine, but anything over 50GB is stretching their capabilities).

Teradata offers methods to define your cubes as ROLAP. This will enable a scalable cube solution that will result in cubes that are larger in dimensionality, detail, and history that are built in a fraction of the time it took to build a MOLAP cube.

About the Author

Carlos Bouloy, a senior consultant, has been with Teradata for 16 years. He specializes in optimizing applications for Teradata. His most recent activities have focused on building ROLAP solutions on Teradata.