Improve your OLAP Environment with Hyperion and Teradata

Rupal Shah

Teradata Corporation

November 2004

Scope

This paper describes how you can use Hyperion and Teradata technologies to improve your analytical OLAP application environment, specifically around Hybrid (HOLAP) and Relational (ROLAP) type solutions. This paper is targeted to BI administrators, system integrators and database developers. Readers are expected to have a basic understanding of the features of the Teradata Database Aggregate Join Index and Hyperion Essbase and Integration Services products.

Table of Contents

Introduction

Hyperion and Teradata understand the challenges facing customers today in delivering and accessing important analytic solutions to their end users with access to detail data. These challenges can range from maintenance and deployment to performance and availability associated with an analytic OLAP application. This paper is really for those folks interested in moving from a MOLAP environment to a HOLAP/ROLAP type solution which can address some of the current challenges when trying to access current or deeper analytics without being overly penalized in performance. However, knowing that 'cut-point', will be imperative to deliver the best analytical OLAP environment to your end-users. We are not making any claims that the techniques identified in this paper will produce 'exact' performance characteristics as in a MOLAP solution, but rather an approach to provide HOLAP/ROLAP type solutions with 'better' and sometimes 'close too' performance one would experience with a MOLAP solution, with the benefits of addressing some of your current challenges.

Business Case

At a high-level, the business case example below will identify some of these challenges in delivering your current and/or deeper analytics and the proposed solution to meet those challenges.

Challenges

Though, these challenges are broad in nature, they are applicable on any OLAP implementation as your environment matures over time.

- **Deeper Analytics** – Providing deeper analytics in a MOLAP design, generally means building larger and possibly more cubes to support such a request. This still is without assurance, based on data volumes, whether or not users will be able to access down to SKU and Account type levels. *Question*: Will my hardware support larger and more cubes?

- **Cube Build Times** – With deeper MOLAP analytics comes longer and more cube builds within the given batch window to perform this task. *Question*: Will my cube(s) build(s) finish in my batch window? What about reducing my current build times?

- **Cube Maintenance** – More cubes more maintenance, larger cube more hardware resources. *Question*: How can I reduce my cube maintenance?

- **Network Saturation with Data Transfer** – Unless your environment is on a private/dedicated network and/or cubes built during off-peak hours, data transfer will impact LAN access. *Question*: How can I reduce network saturation?

- **Analytic Application to reflect more 'real-time' data** – With longer cube builds to meet deeper MOLAP analytics comes less frequent updates to your analytics. *Question*: How current is your analytics? Do you want more 'real-time' analytics?

Solution

To meet the above challenges our customer is interested in implementing a new solution with more direct Teradata access to detail data in the warehouse. The solution approach is as follows:

- **HOLAP/ROLAP type solution** – Customer is interested in extending their analytics with a relational solution to reduce cube build times, reduce cube maintenance, reduce network saturation data transfers and get to 'real-time' detail data in the warehouse.

- **Essbase OLAP Environment** – Customer wants to use their current investment in what they have designed in Essbase. Hence, theoretically no change in current Essbase environment, just what and where the analysis is being done. Any additional work would be in the form of extending their current OLAP model and metaoutline to meet their deeper analytic request. We will assume for this paper readers are familiar with Hyperion products.

- **Teradata Database Feature** – Customer is aware with this approach there is a performance penalty to access those relational OLAP levels. Hence, to assist in attaining 'better' performance in this area we will use a Teradata Database feature called Aggregate Join Index (AJI). The simple definition is, an AJI is nothing more than aggregated result set saved as an object in the database. It is transparent to an end-user and BI Administrators and will be used automatically by the Teradata optimizer when a query plan contains frequently made like columns and aggregates. We will assume for this paper readers are familiar with this feature. For more information, refer to the Teradata Database SQL Reference – Data Definition Statements.

Note: This paper will not focus on any size or hardware configurations with the Teradata Database or Hyperion products. Since, this will vary from customer to customer and the intention here is for the reader to walk away with an approach to 'better' their OLAP environment.

Intentions

To some degree changing just your OLAP design to HOLAP/ROLAP by itself (i.e. without any further Teradata work) will address the reduction of cube maintenance, long build times and network saturation by default. Though, you could stop at this point, two challenges still require your attention to improve your OLAP environment. The first, "better build time" when building Essbase OLAP part/content (i.e. Hyperion Essbase requires top/first-level of analytics to exist in Essbase). And second, "better Hybrid OLAP response" when accessing and navigating your Hybrid OLAP content when it resides in the relational database. Hence, the idea here is to incorporate the Teradata Database AJI feature to improve these two specific areas of your OLAP environment.

Better build time

When building any OLAP content there are 3 component measurements to take into consideration when talking about "Total Build Time". They are: Query, Data Transport and Cube 'build' times, which will all differ greatly depending on the OLAP design (i.e. MOLAP, HOLAP and ROLAP).

- **Query Time** – The time it takes for query, in this case the dataload query, to complete on the database.

- **Data Transport Time** – The time it take to transport the data after the dataload query has completed and populated the Essbase part of the cube/OLAP content.

- **Essbase/Cube 'build' time** – The time it takes to construct the Essbase part of the cube/OLAP content from the data it received and perform whatever additional rollup calculations required to accurately display the defined the top-level OLAP content.

This paper will only focus on improving the first bullet "Query Time" with the Teradata Database AJI feature. To improve second and third bullet items, refer to your network administrator and Essbase documentation for more information and parameter tuning.

Better relational OLAP response

There are 3 types of OLAP analysis environments available to Essbase Analytic Services. They are MOLAP, HOLAP and ROLAP type environments.

- **MOLAP** – refers to all dimensions and levels of your analysis in the Essbase cube.

- **HOLAP** – refers to some dimensions and levels of your analysis in the Essbase cube and in the rest in the relational database

- **ROLAP type** – traditional ROLAP refers to all of your analysis is in the database. However, at this time, Essbase requires at least the top/first-level to reside in Essbase. Hence, we'll call it ROLAP type for a lack of a better term. Though, one could argue this could also be called HOLAP.

Though, this paper we will focus on the third bullet. The principles and the Teradata Database feature example described in this document can apply to any OLAP design that falls between second and third bullets.

Basically, we are going to use an OLAP design that reflects all dimensions and levels are in the database, except for the very top-level of each dimension which will reside in Essbase. Since this is a requirement at this time for Essbase. Hence, it requires some data loading. We will show how using Teradata AJI feature will improve build and relational analysis access times. Though the Teradata Database AJI feature will require DBA involvement, the result AJI object is transparent to an end-user and BI Admin. The Teradata optimizer will determine automatically during its query plan whether or not to use this object. Hence, there is no need to rewrite or recreate your OLAP designs or database access to take advantage of this Teradata Database feature. Which we stated earlier, "theoretically, no change in current Essbase environment, just what and where the analysis is being done."

Environment Considerations

For the purposes this paper and the examples used, the following environment considerations are disclosed and defined here:

OLAP Design

In our case example, we have an Essbase OLAP model that we are interested in addressing the points made in the previous sections. We are interested in shorter cube build time, deeper analysis, less cube maintenance and closer to 'real-time' data in our data warehouse, which can be difficult to achieve in our existing MOLAP model and will exceed our batch 'operational' window timeframe for this type of deliverable analytic. Hence, we will consider a HOLAP/ROLAP type approach to delivering this analytic environment. The Essbase Integration Service OLAP metaoutline will be defined as having all dimensions and levels are in the database, except the first or top-level which will be in Essbase.

Data Warehouse Refresh

Since defining and building an AJI will against base tables will introduce concerns regarding the accuracy of the data being analyzed. We are going to assume the cube part build and AJI creation occurs in a window where the data warehouse is not being updated and is considered current. Though, regardless if you take this approach or not, DBA and BI administrators would always need to re-executed their build whenever the data warehouse has been refreshed.

Proposed Aggregate Join Index

Though there are many combinations one can come up with in regards to what type of AJI should be created. We are proposing in this paper, to create a broad AJI that references all dimensions and levels except the very lowest level. Since, creating an AJI at the lowest level would result in size, time to build and access, no different then different than going against the base transaction tables directly. And the intent here is to provide an AJI with levels of aggregation for those levels of analysis 'most used' in the OLAP environment.

Physical Database Design

Physical database design of any Enterprise Data Warehouse by definition should reflect the customers business independent of any tools requirement or approach to delivering BI content. The Teradata EDW should be designed to adhere to the practices and methodologies to best support a enterprise data warehousing environment. Hence, it will be assumed in this document that this Teradata EDW foundation exists. Meaning, the physical design, whatever it may be, should be agnostic to any tool, and should be ultimately driven by to the customer's business requirements.

In this paper the physical database design is a snowflake. Not to say the information within this paper could not apply to any other physical designs.

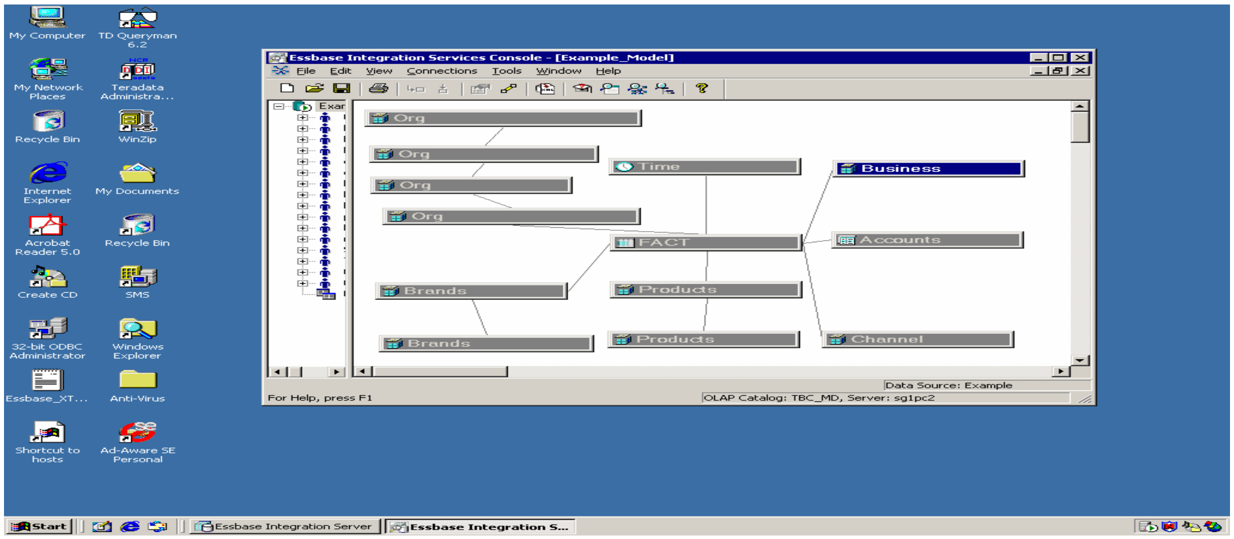
Semantic Layer / View Methodology
Independent of the physical database design mentioned above, the recommended method to access relational content is to create a view/semantic layer (i.e. database/user). The database/user should contain appropriate objects/views pointing to base/production tables that are required to support the customers reporting/analytic requirements and any tool dependencies to delivery BI content. This approach fits in well with Essbase Integration Services approach to designing OLAP models, which is based on a 'logical' star and/or snowflake approach. Not to mention, creating a view layer for users/BI administrators, is generally considered 'good practice' in a Teradata environment. A DBA should work closely with the BI Administrator in determining the appropriate views required to meet the analytic reporting needs.

Note: In this paper the EIS logical OLAP model design is a snowflake and there is a one to one view definition to the base tables.

Better Build Time using AJI

To illustrate the first point above, this section will describe our example EIS OLAP model, steps for using EIS to help create an AJI for our case example and demonstrate its effectiveness for "Better Build Time" using Hyperion Essbase Integration Services product.

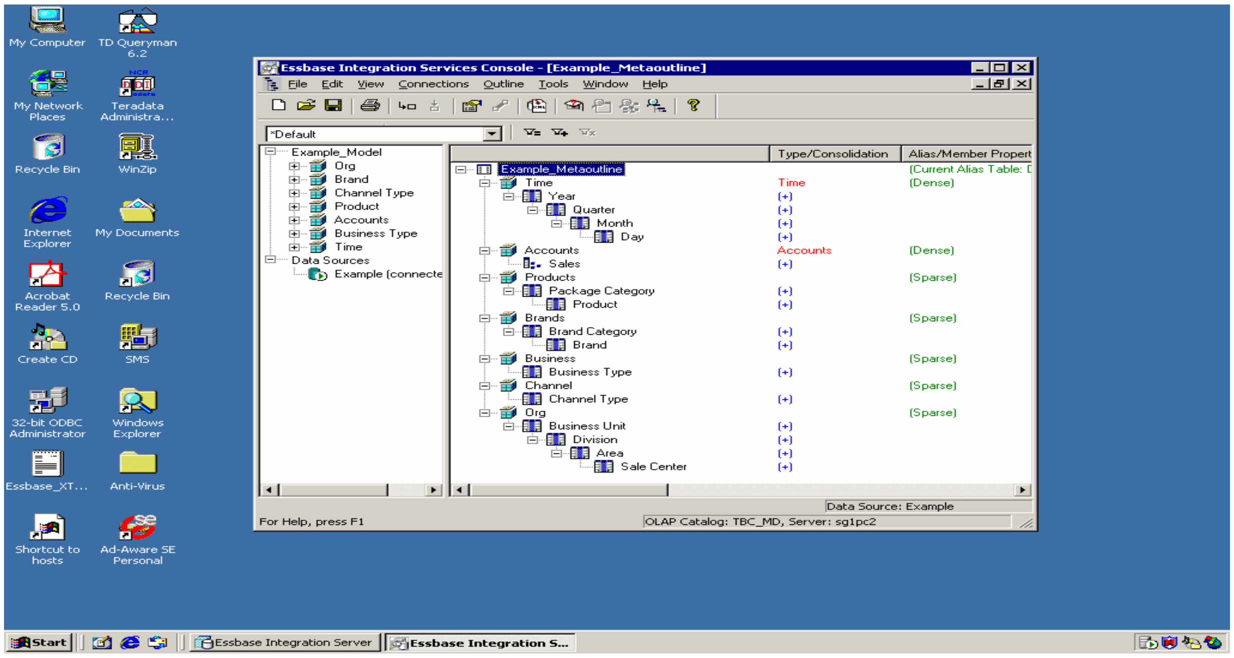
EIS OLAP Model – Figure 1, shows Fact object with (6) dimensions and (1) account measure defined for this OLAP Model.



(Figure 1)

EIS Metaoutline – Figure 2, shows our *current* MOLAP Metaoutline and Figure 2.1 shows our ‘new’ Hybrid HOLAP/ROLAP type model design for our new solution approach. We are assuming readers are familiar with Hyperion products and know how to set dimension levels for relational/hybrid access, which are identified with grey icons and the top-level in Essbase in blue icons.

Note, we are only populating the very top/first-level in Essbase.



Essbase Integration Services Console - [Example_Metaoutline]

*Default

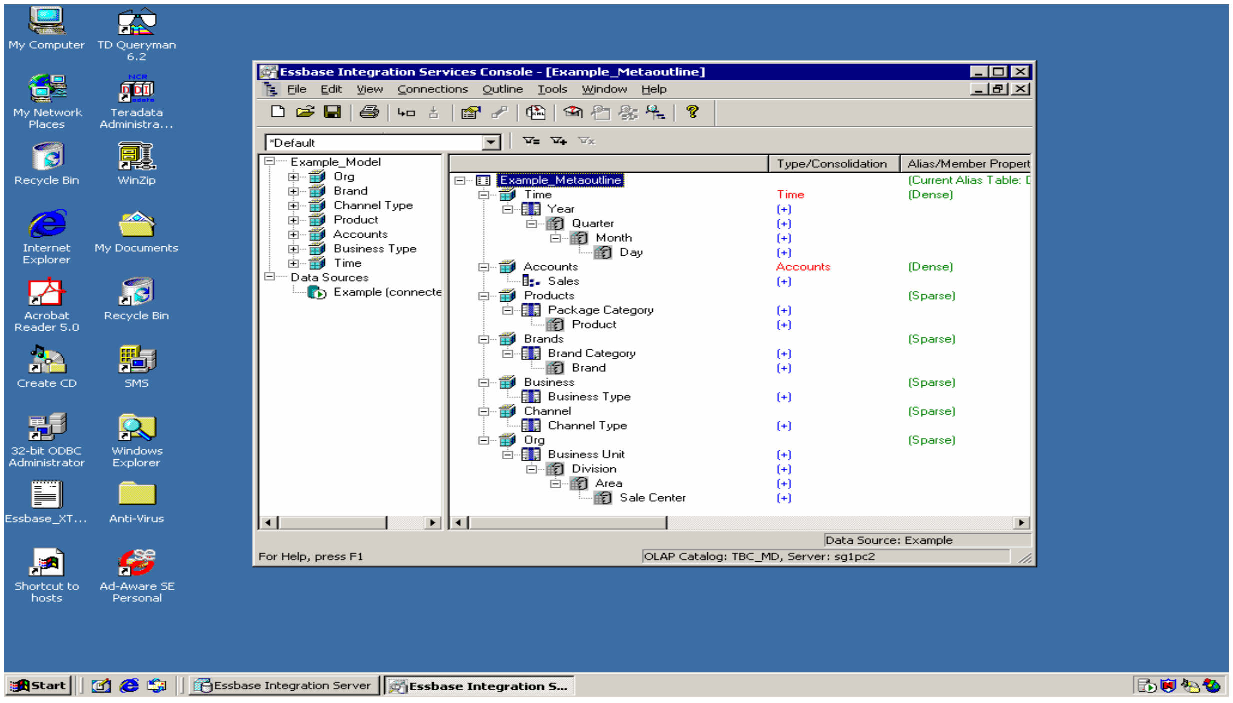
```
Example_Model
  Org
  Brand
  Channel Type
  Product
  Accounts
  Business Type
  Time
Data Sources
  Example (connecte
```

```
                                    Type/Consolidation   Alias/Member Propert
Example_Metaoutline                                      (Current Alias Table: D
  Time                              Time                 (Dense)
    Year                            (+)
      Quarter                       (+)
        Month                       (+)
          Day                       (+)
  Accounts                          Accounts             (Dense)
    Sales                           (+)
  Products                                               (Sparse)
    Package Category                (+)
      Product                       (+)
  Brands                                                 (Sparse)
    Brand Category                  (+)
      Brand                         (+)
  Business                                               (Sparse)
    Business Type                   (+)
  Channel                                                (Sparse)
    Channel Type                    (+)
  Org                                                    (Sparse)
    Business Unit                   (+)
      Division                      (+)
        Area                        (+)
          Sale Center               (+)
```

Data Source: Example

For Help, press F1 | OLAP Catalog: TBC_MD, Server: sg1pc2

(Figure 2)



Essbase Integration Services Console - [Example_Metaoutline]

*Default

```
Example_Model
  Org
  Brand
  Channel Type
  Product
  Accounts
  Business Type
  Time
Data Sources
  Example (connecte
```

```
                                    Type/Consolidation   Alias/Member Propert
Example_Metaoutline                                      (Current Alias Table: D
  Time                              Time                 (Dense)
    Year                            (+)
      Quarter                       (+)
        Month                       (+)
          Day                       (+)
  Accounts                          Accounts             (Dense)
    Sales                           (+)
  Products                                               (Sparse)
    Package Category                (+)
      Product                       (+)
  Brands                                                 (Sparse)
    Brand Category                  (+)
      Brand                         (+)
  Business                                               (Sparse)
    Business Type                   (+)
  Channel                                                (Sparse)
    Channel Type                    (+)
  Org                                                    (Sparse)
    Business Unit                   (+)
      Division                      (+)
        Area                        (+)
          Sale Center               (+)
```

Data Source: Example

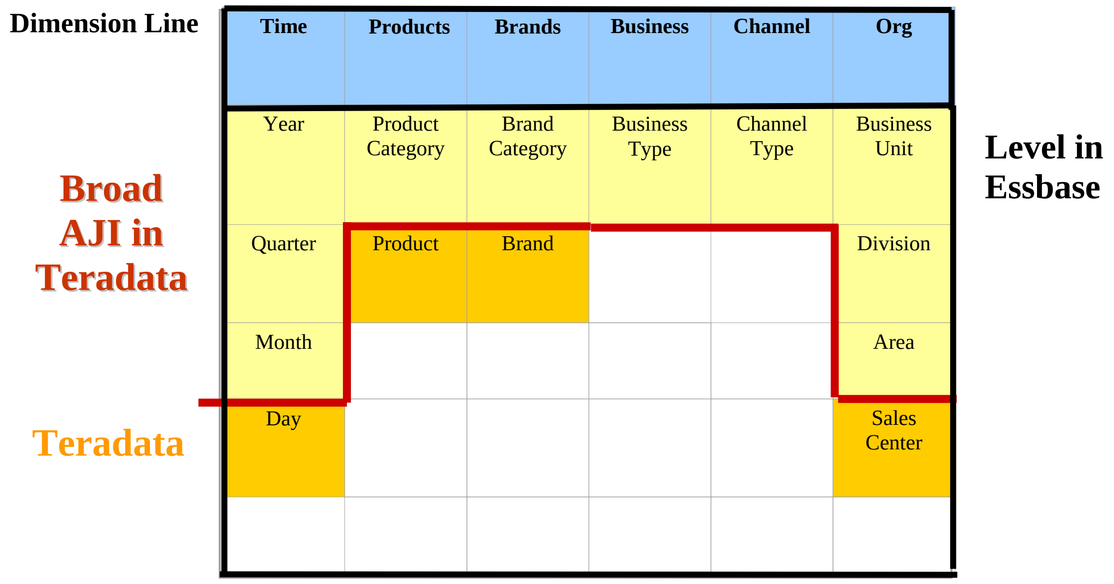
For Help, press F1 | OLAP Catalog: TBC_MD, Server: sg1pc2

(Figure 2.1)

At this point you can execute this build, without implementing our proposed solution with Teradata AJI feature. This build will execute 2 types of queries against the database. First, a set of structural queries to build your dimension and level member labels for Essbase. The second query is your (1) dataload query to bring data into the lowest level in the Essbase cube/part of the OLAP design, which in this case is the top-level only.

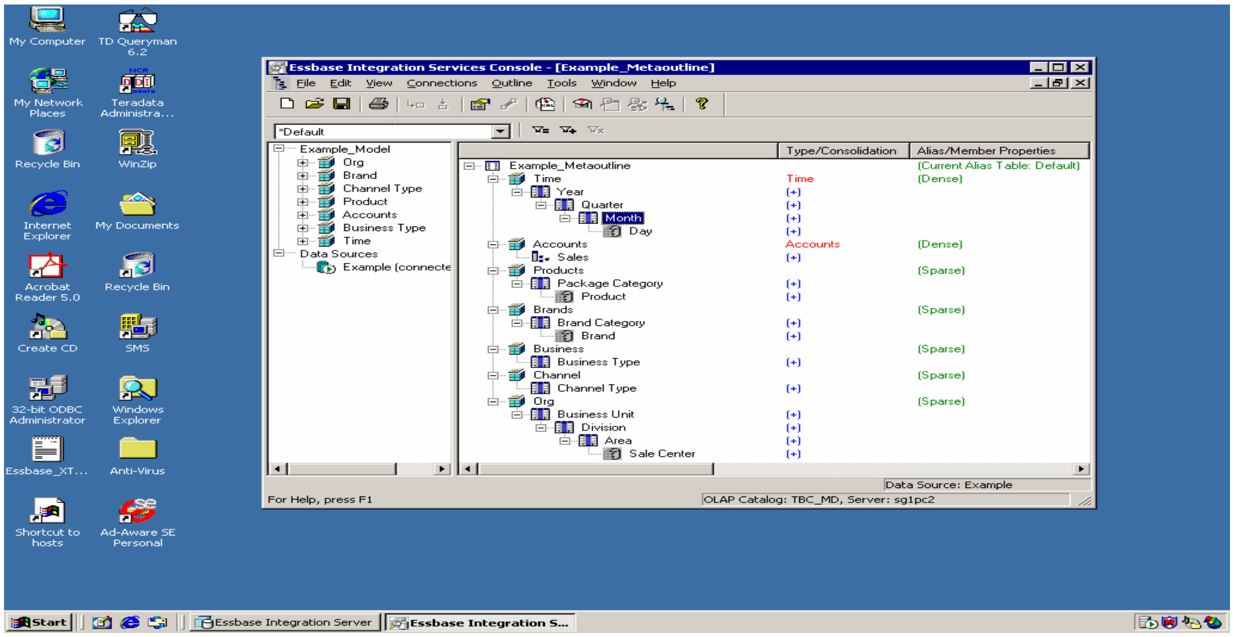
What this paper is proposing is the 'broad' AJI we are suggesting will benefit both and will improve those relational levels and your Essbase cube/part build as well.

Aggregate Join Indexing Strategy – As we mentioned earlier, there are many combinations in regards to what type of AJI should be created. What we are proposing, as mentioned earlier in this paper, is to create an AJI that references all dimensions and levels except the very lowest level. See Figure A below, area in Yellow represents the 'broad' AJI that we will create in Teradata that will contain aggregates on all levels and all columns and measures defined for this model. Why, because this is where we feel end-user will spend most of their analysis time and will require *better* response from Hybrid OLAP type requests. Note this is not the only type of AJI one can create. More experience and better understanding of your end-user requests to the database will determine the types of AJI that can be created to improve beyond this initial suggested approach (i.e. an AJI on a specific dimension or AJI at each relational level). Since we don't have any specific requirement regarding what dimensions and levels are most used at this time. We are proposing (1) 'broad' AJI as a start, to improve end-user response to relational access and to minimize DBA maintenance. Since, there will be DBA maintenance required to support an AJI when refreshing the warehouse. In addition, the AJI we will create will also help build our required top-level BI content in the Essbase cube because the query that will build this top-level will be a subset of the AJI we create. Hence, the optimizer will the AJI rather than accessing the base tables, which will result in a faster query response and will satisfy our first objective of "better build time" of your Essbase OLAP environment.

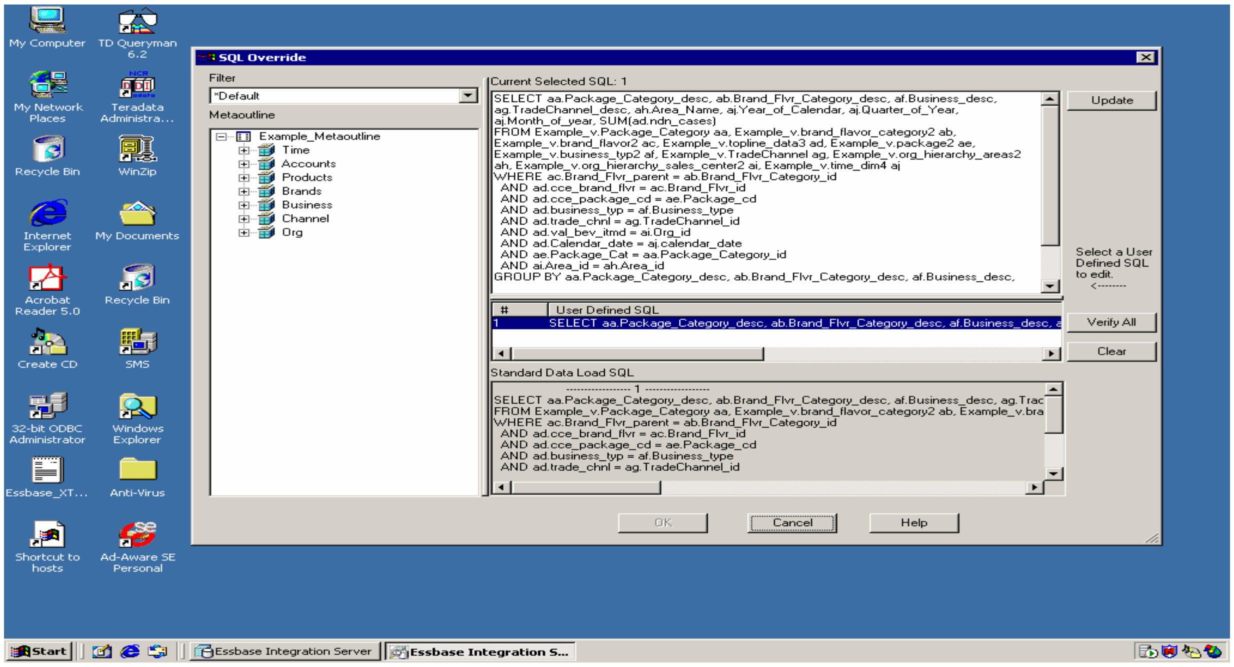


(Figure A)

Using EIS Metaoutline to help define an AJI – Tip, since an AJI is nothing more than an aggregated SELECT statement wrapped in CREATE JOIN INDEX syntax where the result set is saved in the database. We can use EIS to generate the SELECT statement via OLAP Metaoutline interface. This is done when we tag all levels except the lowest (see Figure 3) and view and copy the SQL from the Edit SQL dialog box (see Figure 4). Then wrap the CREATE JOIN INDEX and PRIMARY INDEX syntax (see Figure 5) and execute the DML statement via Queryman or Winddi.



(Figure 3)



Current Selected SQL: 1

```
SELECT aa.Package_Category_desc, ab.Brand_Flvr_Category_desc, af.Business_desc,
ag.TradeChannel_desc, ah.Area_Name, aj.Year_of_Calendar, aj.Quarter_of_Year,
aj.Month_of_year, SUM(ad.ndn_cases)
FROM Example_v.Package_Category aa, Example_v.brand_flavor_category2 ab,
Example_v.brand_flavor2 ac, Example_v.topline_data3 ad, Example_v.package2 ae,
Example_v.business_typ2 af, Example_v.TradeChannel ag, Example_v.org_hierarchy_areas2
ah, Example_v.org_hierarchy_sales_center2 ai, Example_v.time_dim4 aj
WHERE ac.Brand_Flvr_parent = ab.Brand_Flvr_Category_id
 AND ad.cce_brand_flvr = ac.Brand_Flvr_id
 AND ad.cce_package_cd = ae.Package_cd
 AND ad.business_typ = af.Business_type
 AND ad.trade_chnl = ag.TradeChannel_id
 AND ad.val_bev_itmd = ai.Org_id
 AND ad.Calendar_date = aj.calendar_date
 AND ae.Package_Cat = aa.Package_Category_id
 AND ai.Area_id = ah.Area_id
GROUP BY aa.Package_Category_desc, ab.Brand_Flvr_Category_desc, af.Business_desc,
```

#	User Defined SQL
1	SELECT aa.Package_Category_desc, ab.Brand_Flvr_Category_desc, af.Business_desc, a

Standard Data Load SQL

```
                ------------------ 1 ------------------
SELECT aa.Package_Category_desc, ab.Brand_Flvr_Category_desc, af.Business_desc, ag.Trac
FROM Example_v.Package_Category aa, Example_v.brand_flavor_category2 ab, Example_v.bra
WHERE ac.Brand_Flvr_parent = ab.Brand_Flvr_Category_id
 AND ad.cce_brand_flvr = ac.Brand_Flvr_id
 AND ad.cce_package_cd = ae.Package_cd
 AND ad.business_typ = af.Business_type
 AND ad.trade_chnl = ag.TradeChannel_id
```

(Figure 4)

Create the AJI – Creation time for an AJI will depend on size of the tables and system usage.

Note: AJI definition (SQL) syntax can not be agasint view objects, only against actual tables. Though, to keep things organized and within this analytic's semantic layer. We are creating this AJI in the semantic layer (view database) Example_v.

```
CREATE JOIN INDEX Example_v.AJI_Example ,NO FALLBACK ,CHECKSUM = DEFAULT AS
SELECT COUNT(*)(FLOAT, NAMED CountStar), aa.Product_Category, ab.Brand_Category,
af.Business, ag.Channel, ah.Area, al.Division, ak.Business_Unit, aj.Year, aj.Quarter, aj.Month,
SUM(ad.Sales) (FLOAT, NAMED NDN_CASES )
FROM Example.Product_Category aa, Example.Brand_Category ab, Example.Brand ac, Example.FACT
ad, Example.Product ae, Example.Business af, Example.Channel ag, Example.Area ah,
Example.Sale_Center ai, Example.Time aj, Example.Business_Unit ak, Example.Division al
WHERE ac.Brand_Category_Id = ab.Brand_Category_Id
  AND ad.Brand_Id = ac.Brand_Id
  AND ad.Product_Id = ae.Product_Id
  AND ad.Business_Id = af.Business_Id
  AND ad.Channel_Id = ag.Channel_Id
  AND ad.Sale_Center_Id= ai.Sale_Center_Id
  AND ad.Calendar_Date = aj.Calendar_Date
  AND ae.Product_Category_Id = aa.Package_Category_Id
  AND ai.Area_Id = ah.Area_Id
  AND ah.Division_Id = al.Division_Id
  AND al.Business_Unit_Id = ak.Business_Unit_Id
GROUP BY aa.Product_Category, ab.Brand_Category, af.Business, ag.Channel, ah.Area, al.Division,
ak.Business_Unit, aj.Year, aj.Quarter, aj.Month
PRIMARY INDEX ( Product_Category, Brand_Category, Business ,Channel, Area, Division,
Business_Unit, Year ,Quarter, Month);
```

(Figure 5)

Check EIS dataload query against an AJI – After AJI has been created, check via Teradata Explain command (Figure 6) if the AJI will be used by the EIS dataload query for building the top-level in Essbase OLAP (ROLAP type) design (Figure 2.1). To do this reset your Metaoutline to ROLAP type design and copy SQL from Edit SQL dialog box (see Figure 4) and paste in Queryman or Winddi. As Figure 6 shows, the AJI is called in the query plan.

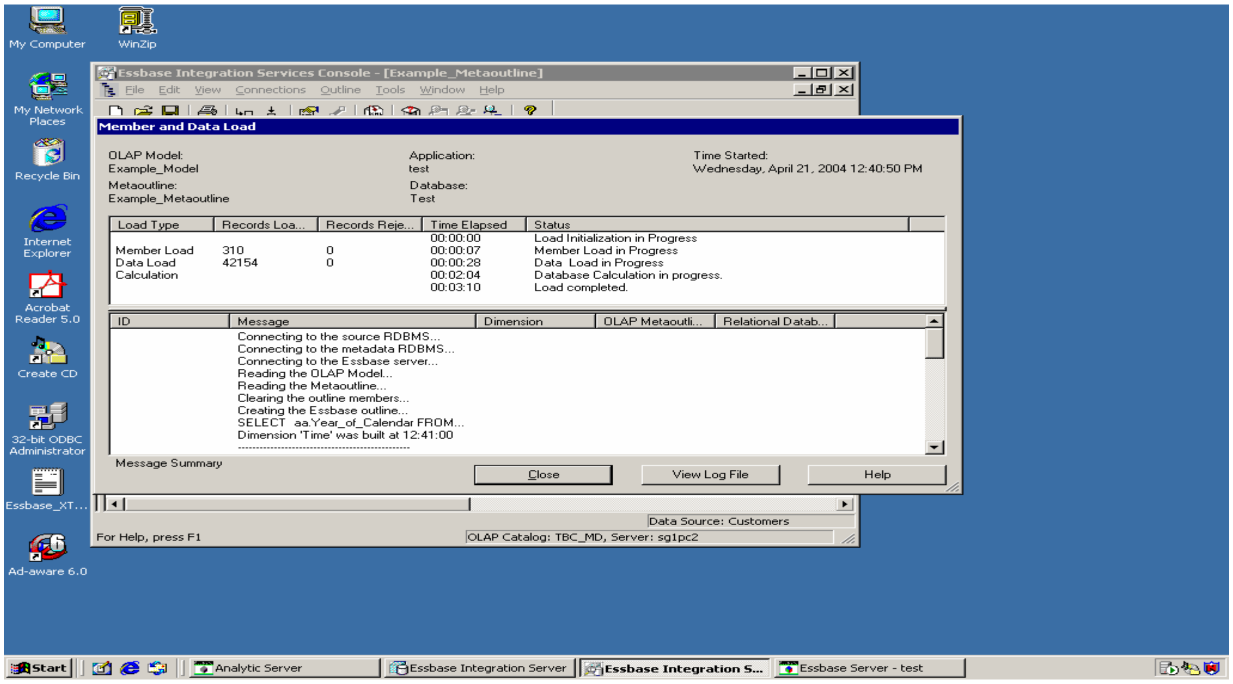
```
Explain
SELECT  aa.Business, ab.Channel, ad.Product_Category, af.Brand_Category, ah.Business_Unit,
al.Year, SUM(ac.Sales)
FROM    Example_v.Business aa, Example_v.Channel ab, Example_v.FACT ac,
Example_v.Product_Category ad, Example_v.Product ae, Example_v.Brand_Category af,
Example_v.Brand ag, Example_v.Business_Unit ah, Example_v.Division ai, Example_v.Area aj,
Example_v.Sale_Center ak, Example_v.Time al
WHERE  aa.Business_Id = ac.Business_Id
  AND   ac.Channel_Id = ab.Channel_Id
  AND   ac.Product_Id = ae.Product_Id
  AND   ac.Brand_Id = ag.Brand_Id
  AND   ae.Product_Category_Id = ad.Product_Category_Id
  AND   ag.Brand_Category_Id = af.Brand_Category_Id
  AND   ah.Business_Unit_id = ai.Business_Unit_Id
  AND   ai.Division_Id = aj.Division_Id
  AND   aj.Area_Id = ak.Area_Id
  AND   ak.Sale_Center_Id = ac.Sale_Center_Id
  AND   al.Calendar_Date = ac.Calendar_Date
GROUP  BY aa.Business, ab.Channel, ad.Product_Category, af.Brand_Category, ah.Business_Unit,
al.Year
ORDER BY     1  ASC ,  2  ASC ,  3  ASC ,  4  ASC ,  5  ASC ,  6  ASC
```

```
1) First, we lock a distinct Example."pseudo table" for read on a RowHash
   to prevent global deadlock for AJI_EXAMPLE.
2) Next, we lock Example_v.AJI_Example for read.
3) We do an all-AMPs SUM step to aggregate from Example_v.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 aggregate spool
   file will not be cached in memory.  The size of Spool 3 is
   estimated with low confidence to be 2,912,040 rows.  The estimated
   time for this step is 8 minutes and 38 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.  Then we do a SORT to order Spool 1 by the sort key
   in spool field1.  The result spool file will not be cached in
   memory.  The size of Spool 1 is estimated with low confidence to
   be 2,912,040 rows.  The estimated time for this step is 15.92
   seconds.
5) Finally, we send out an END TRANSACTION step to all AMPs involved
   in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of
   statement 1.
```

(Figure 6)

Note: This dataload query contains a subset of columns that make up our AJI. Hence, our optimizer will use it instead of going against our base tables.

Hence, as Figure 7 shows below, the EIS build for our ROLAP type design took 3mins to build (i.e. query, data transfer for top level only). Where as, without the AJI the build would have completed in 42mins.



OLAP Model:
Example_Model
Metaoutline:
Example_Metaoutline
Application:
test
Database:
Test
Time Started:
Wednesday, April 21, 2004 12:40:50 PM

Load Type	Records Loa...	Records Reje...	Time Elapsed	Status
			00:00:00	Load Initialization in Progress
Member Load	310	0	00:00:07	Member Load in Progress
Data Load	42154	0	00:00:28	Data Load in Progress
Calculation			00:02:04	Database Calculation in progress.
			00:03:10	Load completed.

ID	Message	Dimension	OLAP Metaoutli...	Relational Datab...
	Connecting to the source RDBMS...			
	Connecting to the metadata RDBMS...			
	Connecting to the Essbase server...			
	Reading the OLAP Model...			
	Reading the Metaoutline...			
	Clearing the outline members...			
	Creating the Essbase outline...			
	SELECT aa.Year_of_Calendar FROM...			
	Dimension 'Time' was built at 12:41:00			
	--			

Message Summary

Data Source: Customers
For Help, press F1
OLAP Catalog: TBC_MD, Server: sg1pc2

(Figure 7)

Granted it took 2 hours to build the AJI, but as we will demonstrate in the next section below “for better OLAP relational response” the same AJI would have been required by a majority of the queries sent to the database by Essbase when accessing relational/hybrid levels.

Better relational OLAP response using AJI

Now to illustrate second point above, we will show the AJI above being referenced by an Essbase Hybrid query when navigating relational levels for analysis. As you can see, below the query contains columns and joins that were referenced in our broad AJI. Hence, there is no additional work required for relational OLAP queries from Essbase to take advantage of our newly created AJI in our example. This will occur automatically via the Teradata optimizer without any intervention from the end-user, BI or DBA administrator. Result in relational query now takes 3 seconds instead of 95 seconds in our example.

```
SELECT DISTINCT
        aa.Year ,
        aa.Quarter ,
        ab.Business_Type ,
        ac.Channel_Type ,
        ad.Product_Category ,
        ae.Brand_Category,
        af.Business_Unit ,
        SUM ( ag.sales )
FROM
        time aa , business_type ab , channel_type ac , product_category ad ,
        brand_category ae , org_business_units af , fact ag , product ah , brand ai ,
        org_sales_center aj , org_areas ak , org_divisions al
WHERE
        aa.calendar_date = ag.calendar_date AND
        ab.business_type_id = ag.business_type_id AND
        af.business_unit_id = al.business_unit_id AND
        ag.channel_id = ac.channel_id AND
        ag.product_id = ah.product_id AND
        ag.brand_id = ai.brand_id AND
        ah.product_category_id = ad.product_category_id AND
        ai.brand_category_id = ae.brand_category_id AND
        aj.org_id = ag.org_id AND
        ak.area_id = aj.area_id AND
        al.division_id = ak.division_id
        AND ( ( ( aa.year = '2000' ) ) ) AND ( ( ( ab.business_type = ‘RETAILER' ) ) )
        AND ( ( ( ac.channel_type = 'WHOLESALE' ) ) ) AND ( ( (
        ad.product_category = ‘TENTS' ) ) ) AND ( ( ( ae.brand_category = ‘APEX' ) )
        ) AND ( ( ( af.business_unit = ‘MIDWEST' ) ) )
GROUP BY
        aa.Year , aa.Quarter, ab.Business_Type , ac.Channel_Type ,
        ad.Product_Category , ae.Brand_Category , af.Business_Unit
ORDER BY
```

w/ AJI	**: 3 secs**
w/o AJI	**: 95 secs**

Summary

We hope this paper has shown the reader an approach on how to use Teradata Database Aggregate Join Index feature to help ‘better’ your OLAP environment. In our business case example we created 1 ‘broad’ AJI to address some of the challenges in delivering your current and deeper OLAP analytics. Many combinations and various AJI constructs can greatly improve your OLAP experience. This business case is only one example. Understanding the cost (time to build and maintain AJI), trade-offs (not quite ‘speed of thought’) and appropriate ‘cut-point’ (Essbase and relational OLAP parts) will be ‘key’ to implementing and deploying the right OLAP environment for your end-users. Some of the highlighted conclusions, benefits and disadvantages are below:

Conclusions

- Aggregate Join indexes can greatly improve query performance – for any Essbase cube part build and relational OLAP access

- Easy to define – fairly straight forward to create with the help of using EIS to create the SQL statement

- Network traffic reduced – since, HOLAP/ROLAP keeps majority of data in the data warehouse, less data is transferred

- Cube build times minimized – less data transfers means faster cube part build time.

- Built in parallel – using Teradata for heavy lifting when building AJI

- Better response times with relational levels – using AJI for faster response

- Requires Teradata Database V2R5.1

Benefits

- Indexes created are can be relatively small structures - dependent upon number of demographics rather than number of rows in Fact/base table(s)

- More in line with an Active data warehouse as opposed to MOLAP – access to ‘real-time’ data and more frequent Essbase cube build with a HOLAP solution

- AJIs can be shared by multiple cube definitions – transparent to any tool or user

- Can create broader and deeper cubes - more dimensions, more categories, more member

- Maintenance on number of cubes reduced – no need to build other cubes to address deeper or wider analytics

Disadvantages

With advantages come disadvantages when using the mentioned above solution with AJI. Though, this paper did not get into the maintenance trade-offs and requirements to ensure your relational queries reference the AJI. Readers should reference

Teradata Database documentation for requirements to ensure Teradata optimizer will use the AJI you create. A few of these points are made below:

- Updates to base table are slower – AJI updates are tied to base table updates. In MOLAP this is not the case
- Referential Integrity – RI is required to ensure optimizer will use AJI based on incoming requests referencing join relationships between tables.
- Response times – Never claimed our solution will perform 'exact' to a MOLAP solution. Especially when cube analysis goes deeper and wider at the same time.
- Today's optimizer – need to be 'SQL' aware. RI is one requirement to ensure this.
- No desktop cube – no mobile or local cube option available.

Next Steps

To address lowest (relational) levels in your OLAP designs, consider the following:

Secondary Indexes

Secondary indexes provide faster set selection. Secondary indexes are frequently selected by the Optimizer when a search condition cannot be satisfied with a primary index retrieval. The Optimizer also selects secondary indexes for query plans when they completely or partially cover a query.

Partitioned Primary Index

Are designed to optimize range queries while also providing efficient primary index join strategies. Analyze your range query optimization needs carefully because there are performance tradeoffs between specific range query enhancements and possible decrements for primary index accesses and joins and aggregations on the primary index that occur as a function of the number of active partitions.