Multi-Pass SQL

Techniques for BOE XIr2

Multi-Pass SQL

BOE Release Dependency

All the information and examples contained within this presentation are based upon Business Objects Enterprise XIr2. The service pack (SP) level is at SP2 or higher. At this time Business Objects Enterprise 3.0 has just been released. BOE 3.0 has new features that impact multi-pass SQL implementation. A revision of this presentation is currently planned. None of the features in BOE 3.0 render the techniques reviewed obsolete. The additional features ease multi-pass SQL implementation and provide new techniques that can simplify the process.

Multi-Pass Definition

- Creation of Multiple Queries Against a Database
  - Create a number of simple queries to be processed separately by the DBMS
- Combine Query Results to Achieve Desired Outcome
- Traditional Approach
  - Use temporary tables to store intermediate results of queries
  - Combine temp tables with each other or with permanent tables for final result
- Query Tool Approach
  - Combine query results in an intelligent way for final result
  - Utilize reporting tier

Multi-Pass Competitive Landscape

- All BI Vendors Claim Capability
  - Truth is all do
  - But to various degrees
- Microstrategy
  - Ones who are more likely to stress capability
  - Claim all other vendors are limited to single pass
- Implementing Multi-Pass within Business Objects
  - How is it invoked?
  - Where is burden placed?
    - Semantic layer
    - Report builder
    - End user
  - How much effort at each level?

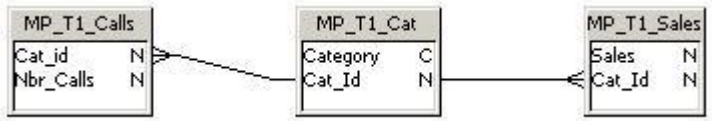
Multi-Pass Scenarios

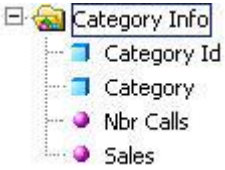
- Sharing Dimensions Across Fact Tables
- Calculations Which Require End Results
  - Ratios: Pct of product sales to total sales
- Blending Grains of Measurements
  - Time: Year-To-Date, Month-To-Date, Last-Year-To-Date, etc
- Need for Semi-Additive Measures
  - Inventory as of: end of quarter, end of month, end of week
- Analyzing a Subset of Data
  - Last transaction for each account, latest status record for account

Sharing Dimensions

- One Dimension Table
  - Product Category
- Two Fact Tables
  - Sales Calls
  - Sales Amounts
- Report requires the total number of Sales Calls, total Sales Amount by Product Category

Cat_Id	Category
1	Electronics
2	Food
3	Gifts
4	Health & Beauty
5	Household
6	Kid's Korner
7	Travel





Sharing Dimensions

- Proposed SQL leads to incorrect results
- Nature of SQL
  - Not specific to Business Objects
  - Not specific to any RDBMS
- Nbr Calls duplicated by number of times that a category has a sales amount recorded
- Sales duplicated by number of times that a category has a sales call recorded

```
SELECT
MP_T1_CAT.Category, MP_T1_CAT.Cat_Id,
sum(MP_T1_Calls.Nbr_Calls),
sum(MP_T1_Sales.Sales)
FROM
MP_T1_Cat, MP_T1_Calls, MP_T1_Sales
WHERE
MP_T1_Cat.Cat_Id = MP_T1_Calls.Cat_id
AND MP_T1_Cat.Cat_Id = MP_T1_Sales.Cat_Id
GROUP BY MP_T1_CAT.Category,
MP_T1_CAT.Cat_Id
```

Category Id	Category	Nbr Calls	Sales
1	Electronics	120	119,745
2	Food	10	21,876
3	Gifts	180	181,810
4	Health & Beauty	30	46,828
5	Household	165	887,740
6	Kid's Korner	120	506
7	Travel	21	18,578

Sharing Dimensions

- Traditionalist approach
  - Create a temp table with columns of Cat_ID, Nbr_Sales_Calls, and Sales

```
CREATE TABLE Sales_and_Calls
   (Cat_Id integer, Nbr_Calls integer,
    Sales decimal(10,2))
```

  - Insert number of calls by category in table

```
INSERT INTO Sales_and_Calls (Cat_Id,
   Nbr_Calls, Sales)
   SELECT MP_T1_Calls.Cat_ID,
   sum(MP_T1_Calls.Nbr_Calls), 0
   FROM MP_T1_Calls
   GROUP BY  MP_T1_Calls.Cat_Id
```

  - Insert total sales amount by category in table

```
INSERT INTO Sales_and_Calls (Cat_Id,
   Nbr_Calls, Sales)
   SELECT MP_T1_Sales.Cat_ID, 0,
   sum(MP_T1_Sales.Sales)
   FROM MP_T1_Sales
   GROUP BY  MP_T1_Sales.Cat_Id
```

  - Select results for report

```
SELECT MP_T1_Cat.Cat_Id, Category,
   sum(Nbr_Calls), sum(Sales)
FROM Sales_and_Calls JOIN MP_T1_Cat
ON MP_T1_Cat.Cat_Id = Sales_and_Calls.Cat_Id
GROUP BY MP_T1_Cat.Cat_Id, Category
ORDER BY MP_T1_Cat.Cat_Id
```

  - Drop table

```
DROP TABLE Sales_and_Calls
```

Sharing Dimensions

- Data inserted into the temporary table contains separate rows for Nbr_Calls and Sales
- Once pulled together for report, results are correct
- Process is very database centric

	Cat_Id	Nbr_Calls	Sales
1	1	30	0.00
2	1	0	39915.00
3	2	10	0.00
4	2	0	10938.00
5	3	60	0.00
6	3	0	36362.00
7	4	0	11707.00
8	4	30	0.00
9	5	0	88774.00
10	5	55	0.00
11	6	0	5502.00
12	6	60	0.00
13	7	0	9289.00
14	7	21	0.00

	CAT_ID	CATEGORY	Sum(Nbr_Calls)	Sum(Sales)
1	1	Electronics	30	39915.00
2	2	Food	10	10938.00
3	3	Gifts	60	36362.00
4	4	Health & Beauty	30	11707.00
5	5	Household	55	88774.00
6	6	Kid's Korner	60	5502.00
7	7	Travel	21	9289.00

Sharing Dimensions

- How does Business Objects handle this?
  - Business Objects does not create work tables within the database
  - But it does create multiple SELECT statements
  - Results are combined on the Business Objects tier
  - Not on the database tier

```
SELECT
MP_T1_CAT.Category, MP_T1_CAT.Cat_Id,
sum(MP_T1_Sales.Sales)
FROM
MP_T1_Cat, MP_T1_Sales
WHERE
MP_T1_Cat.Cat_Id = MP_T1_Sales.Cat_Id
GROUP BY MP_T1_CAT.Category,
MP_T1_CAT.Cat_Id
```

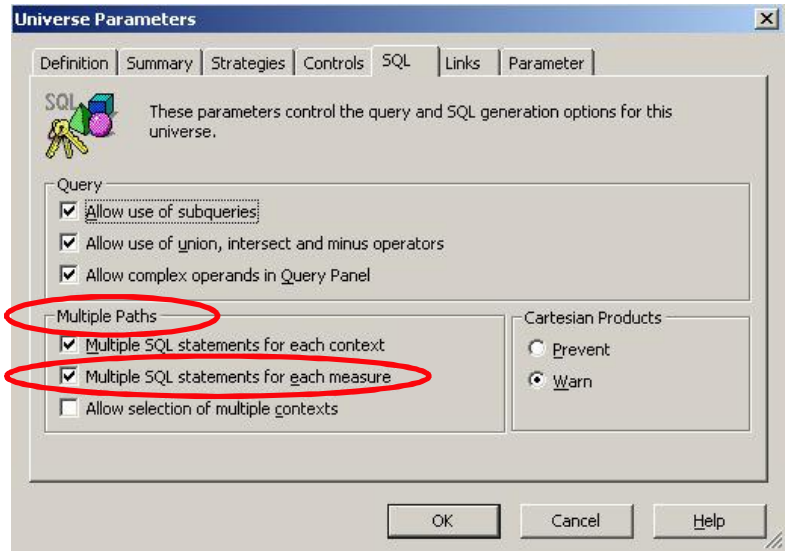
```
SELECT
MP_T1_CAT.Category, MP_T1_CAT.Cat_Id
sum(MP_T1_Calls.Nbr_Calls)
FROM
MP_T1_Cat, MP_T1_Calls
WHERE
MP_T1_Cat.Cat_Id = MP_T1_Calls.Cat_id
GROUP BY MP_T1_CAT.Category,
MP_T1_CAT.Cat_Id
```

Sharing Dimensions

- Within the query panel select the desired objects
- The two SELECT statements are generated
- Result sets from the two SELECTs are combined on the Business Objects tier and then displayed
- Results are identical to the *traditional* hand coding method

Category Id	Category	Nbr Calls	Sales
1	Electronics	30	39,915
2	Food	10	10,938
3	Gifts	60	36,362
4	Health & Beauty	30	11,707
5	Household	55	88,774
6	Kid's Korner	60	5,502
7	Travel	21	9,289

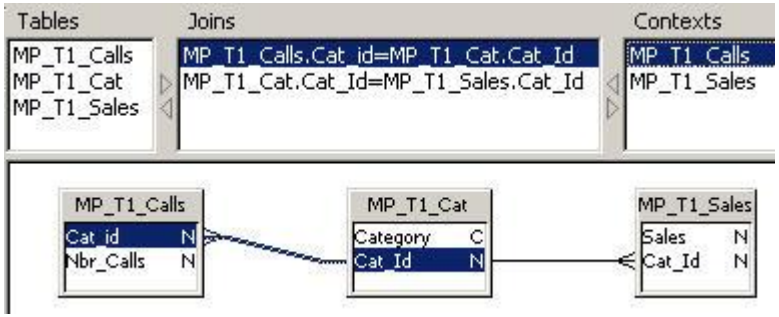
Sharing Dimensions

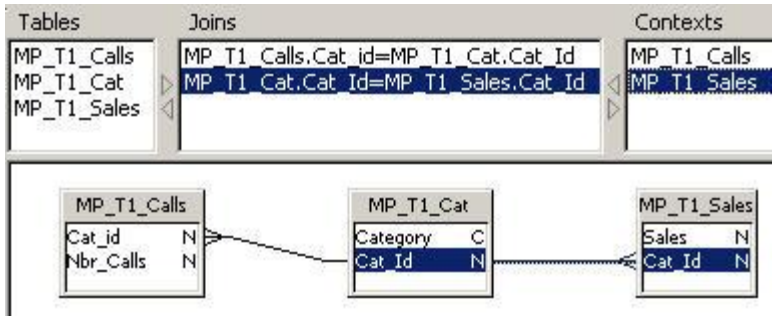


- Universe parameter controls creation of multiple SELECT
- Enabled by default
- Actually the label, *Multiple SQL statements for each measure*, is a little misleading
  - Notice section name for parameter, *Multiple Paths*
  - Only applies when measures being retrieved are from different tables

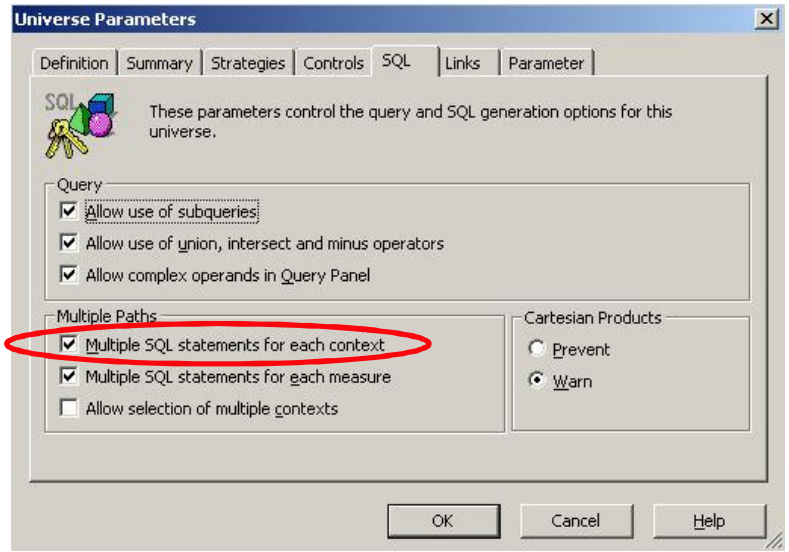
Sharing Dimensions

- Contexts can also be used to solve this multi-pass scenario
- Contexts separate multiple paths between tables
  - From MP_T1_Cat to MP_T1_Calls
  - From MP_T1_Cat to MP_T1_Sales
- Proper Universe design dictates that contexts should always be identified within a Universe
  - If cardinalities have been properly identified, contexts can automatically detected by Universe Designer



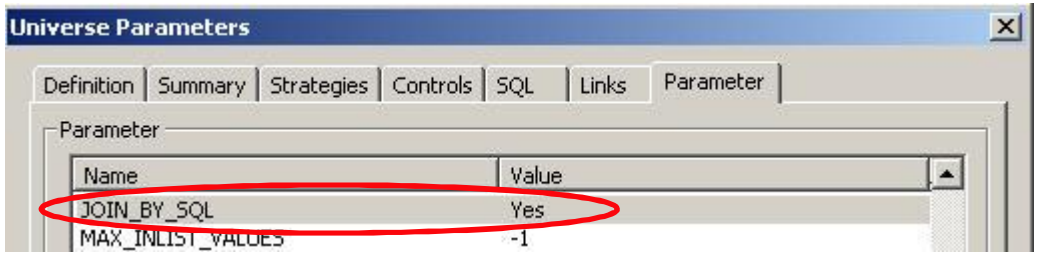


Sharing Dimensions



- Universe parameter, *Multiple SQL statements for each context,* controls creation of multiple SELECTs due to contexts
- Enabled by default
- Extremely useful when *Multiple SQL statements for each measure* is disabled

Sharing Dimensions



- Must Business Objects combine the query result sets?
  - *JOIN_BY_SQL*, Universe parameter was introduced in BOE XIr2 SP1
  - Creation of multiple SELECT statements controlled by previously reviewed parameters
- *JOIN_BY_SQL* affects SQL generation only
  - Each SELECT statement generates a derived table
  - Results sets of derived table combined by the database
  - Business Objects receives only one result set from the database

Sharing Dimensions

- Same objects used in query panel as before

```
SELECT
  COALESCE( F__1.Axis__1,F__2.Axis__1  ),
  COALESCE( F__1.Axis__2,F__2.Axis__2  ),
  F__1.M__3, F__2.M__4
FROM
   ( SELECT
       MP_T1_Cat.Cat_Id AS Axis__1,
       MP_T1_Cat.Category AS Axis__2,
       sum(MP_T1_Calls.Nbr_Calls) AS M__3
     FROM MP_T1_Cat, MP_T1_Calls
     WHERE
       ( MP_T1_Calls.Cat_id=MP_T1_Cat.Cat_Id
   )
     GROUP BY
       MP_T1_Cat.Cat_Id,
       MP_T1_Cat.Category ) F__1
   FULL OUTER JOIN
   ( SELECT
       MP_T1_Cat.Cat_Id AS Axis__1,
       MP_T1_Cat.Category AS Axis__2,
       sum(MP_T1_Sales.Sales) AS M__4
     FROM MP_T1_Cat, MP_T1_Sales
     WHERE
       ( MP_T1_Cat.Cat_Id=MP_T1_Sales.Cat_Id
   )
     GROUP BY
       MP_T1_Cat.Cat_Id,
       MP_T1_Cat.Category ) F__2
   ON ( F__1.Axis__1=F__2.Axis__1 AND
   F__1.Axis__2=F__2.Axis__2  )
```

Sharing Dimensions

- Same objects used in query panel as before
- Derived table F__1 retrieves the Number of Calls along with the two dimensions, Category and Category ID

```
SELECT
  COALESCE( F__1.Axis__1,F__2.Axis__1  ),
  COALESCE( F__1.Axis__2,F__2.Axis__2  ),
  F__1.M__3, F__2.M__4
FROM
   ( SELECT
       MP_T1_Cat.Cat_Id AS Axis__1,
       MP_T1_Cat.Category AS Axis__2,
       sum(MP_T1_Calls.Nbr_Calls) AS M__3
     FROM MP_T1_Cat, MP_T1_Calls
     WHERE
       ( MP_T1_Calls.Cat_id=MP_T1_Cat.Cat_Id
   )
     GROUP BY
       MP_T1_Cat.Cat_Id,
       MP_T1_Cat.Category ) F__1
   FULL OUTER JOIN
   ( SELECT
       MP_T1_Cat.Cat_Id AS Axis__1,
       MP_T1_Cat.Category AS Axis__2,
       sum(MP_T1_Sales.Sales) AS M__4
     FROM MP_T1_Cat, MP_T1_Sales
     WHERE
       ( MP_T1_Cat.Cat_Id=MP_T1_Sales.Cat_Id
   )
     GROUP BY
       MP_T1_Cat.Cat_Id,
       MP_T1_Cat.Category ) F__2
   ON ( F__1.Axis__1=F__2.Axis__1 AND
   F__1.Axis__2=F__2.Axis__2  )
```

Sharing Dimensions

- Same objects used in query panel as before
- Derived table F__1 retrieves the Number of Calls along with the two dimensions, Category and Category ID
- Derived table F__2 retrieves the Total Sales along with the two dimensions, Category and Category ID

```
SELECT
  COALESCE( F__1.Axis__1,F__2.Axis__1  ),
  COALESCE( F__1.Axis__2,F__2.Axis__2  ),
  F__1.M__3, F__2.M__4
FROM
   ( SELECT
       MP_T1_Cat.Cat_Id AS Axis__1,
       MP_T1_Cat.Category AS Axis__2,
       sum(MP_T1_Calls.Nbr_Calls) AS M__3
     FROM MP_T1_Cat, MP_T1_Calls
     WHERE
       ( MP_T1_Calls.Cat_id=MP_T1_Cat.Cat_Id
   )
     GROUP BY
       MP_T1_Cat.Cat_Id,
       MP_T1_Cat.Category ) F__1
  FULL OUTER JOIN
  ( SELECT
      MP_T1_Cat.Cat_Id AS Axis__1,
      MP_T1_Cat.Category AS Axis__2,
      sum(MP_T1_Sales.Sales) AS M__4
    FROM MP_T1_Cat, MP_T1_Sales
    WHERE
      ( MP_T1_Cat.Cat_Id=MP_T1_Sales.Cat_Id
  )
    GROUP BY
      MP_T1_Cat.Cat_Id,
      MP_T1_Cat.Category ) F__2
  ON ( F__1.Axis__1=F__2.Axis__1 AND
  F__1.Axis__2=F__2.Axis__2  )
```

Sharing Dimensions

- Same objects used in query panel as before
- Derived table F__1 retrieves the Number of Calls along with the two dimensions, Category and Category ID
- Derived table F__2 retrieves the Total Sales along with the two dimensions, Category and Category ID
- Primary SELECT combines the results from the derived tables using the COALESCE function to return non-null values of the dimensions
  - AXIS__1 is Category ID
  - AXIS__2 is Category
  - F__1.M__3 is Number of Calls
  - F__2.M__4 is Sales Amount

```
SELECT
  COALESCE( F__1.Axis__1,F__2.Axis__1  ),
  COALESCE( F__1.Axis__2,F__2.Axis__2  ),
  F__1.M__3, F__2.M__4
FROM
    ( SELECT
        MP_T1_Cat.Cat_Id AS Axis__1,
        MP_T1_Cat.Category AS Axis__2,
        sum(MP_T1_Calls.Nbr_Calls) AS M__3
      FROM MP_T1_Cat, MP_T1_Calls
      WHERE
        ( MP_T1_Calls.Cat_id=MP_T1_Cat.Cat_Id  )
      GROUP BY
        MP_T1_Cat.Cat_Id,
        MP_T1_Cat.Category ) F__1
    FULL OUTER JOIN
    ( SELECT
        MP_T1_Cat.Cat_Id AS Axis__1,
        MP_T1_Cat.Category AS Axis__2,
        sum(MP_T1_Sales.Sales) AS M__4
      FROM MP_T1_Cat, MP_T1_Sales
      WHERE
        ( MP_T1_Cat.Cat_Id=MP_T1_Sales.Cat_Id  )
      GROUP BY
        MP_T1_Cat.Cat_Id,
        MP_T1_Cat.Category ) F__2
    ON ( F__1.Axis__1=F__2.Axis__1 AND
    F__1.Axis__2=F__2.Axis__2  )
```

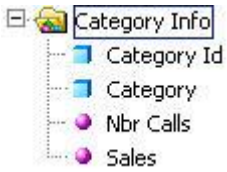
Sharing Dimensions - Recap

- Two Universe Parameters
  - *Multiple SQL statements for each measure*
  - *Multiple SQL statements for each context*
  - Both enabled by default
  - Results multiple SQL statements
- Additional Universe Parameter
  - *JOIN_BY_SQL*
  - Converts multiple SELECT statements into derived tables
  - Primary *SELECT* then uses derived tables as query source
  - Method is more database centric
- More Innovative Approach Versus Traditional Method
  - Better performance due to less database activity
  - Partial shift of load from database to reporting tier

Advanced Calculations

- Require end result as part of the calculation
  - Percentage of sales by category
- Requires
  - Total overall sales
  - Total sales by category

Cat_Id	Category
1	Electronics
2	Food
3	Gifts
4	Health & Beauty
5	Household
6	Kid's Korner
7	Travel



Advanced Calculations

- Traditionalist approach
  - Create a temp table with one columns, Total Sales
  - Insert total overall sales
  - Select results for report
    - Total sales used from the temporary table to calculate ratio
  - Drop table

```
CREATE TABLE Total_Sales
   (All_Sales decimal(10,2))
```

```
INSERT INTO Total_Sales (All_Sales)
   SELECT sum(MP_T1_Sales.Sales)
   FROM MP_T1_Sales
```

```
SELECT Category,
   sum(MP_T1_Sales.Sales),
   (cast(sum(MP_T1_Sales.Sales) as
   decimal(10,4))/(All_Sales) ) * 100
   as Pct_of_Sales
   FROM MP_T1_Sales, Total_Sales,
   MP_T1_Cat
   WHERE MP_T1_Cat.Cat_Id =
   MP_T1_Sales.Cat_Id
   GROUP BY Category, All_Sales
   ORDER BY Category
```

```
DROP TABLE Total_Sales
```

Advanced Calculations

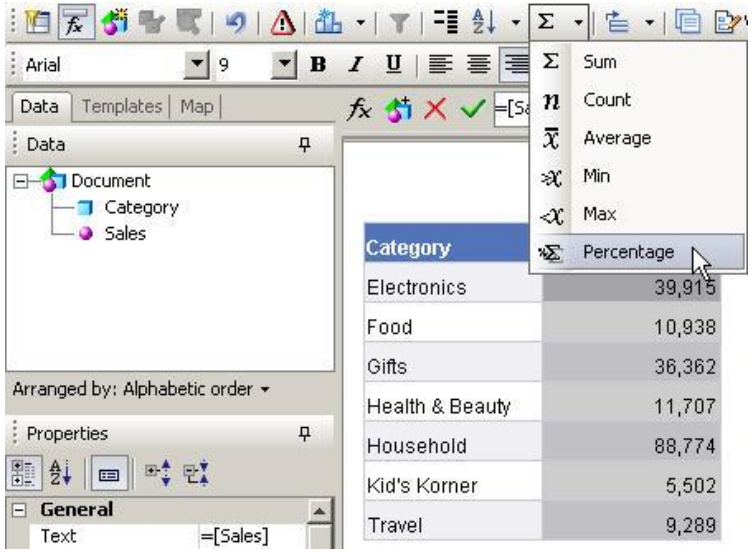
- One row of data inserted into the temporary table which contains the overall total of Sales
- Result can then be used to perform calculation in primary SELECT
  - SELECT between the SALES fact table and the temporary table does produce Cartesian product
  - Result is still accurate since only one row exists in the temporary table

	All_Sales
1	202487.00

	CATEGORY	Sum(SALES)	Pct_of_Sales
1	Electronics	39915	19.7100
2	Food	10938	5.4000
3	Gifts	36362	17.9600
4	Health & Beauty	11707	5.7800
5	Household	88774	43.8400
6	Kid's Korner	5502	2.7200
7	Travel	9289	4.5900

Advanced Calculations

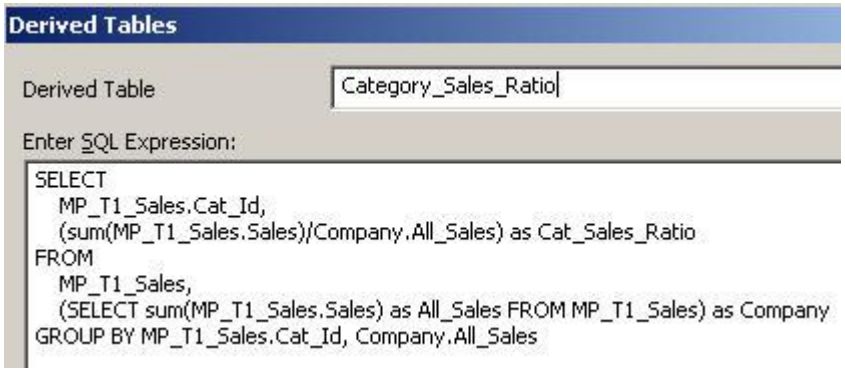
- How does Business Objects handle this?
  - Commonly done as a variable at the report level
- Advantages
  - Some calculations are very simple
  - With the *For Each*, *For All*, and *Where* context operators many calculations can be created
  - No need to wait on semantic layer development
- Disadvantages
  - Not able to share report variables
  - Incorrect formulas
  - Multiple versions of the *Truth* possible
  - Ease of use



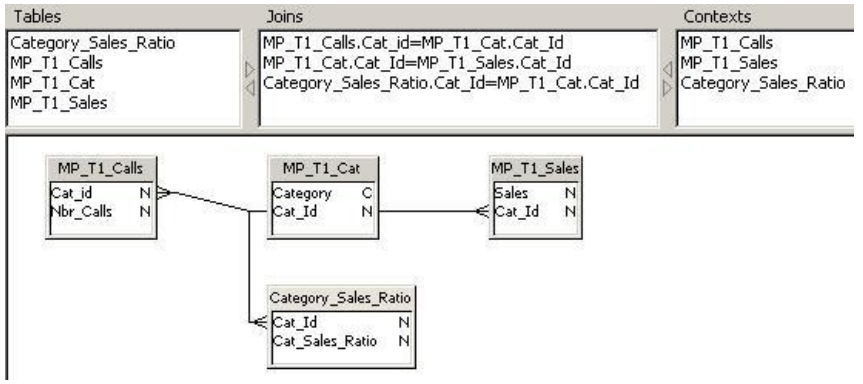
Category	Sales	Percentage
Electronics	39,915	19.71%
Food	10,938	5.40%
Gifts	36,362	17.96%
Health & Beauty	11,707	5.78%
Household	88,774	43.84%
Kid's Korner	5,502	2.72%
Travel	9,289	4.59%
	Percentage:	100.00%

Advanced Calculations

- Create derived table within Universe to calculate required measure



- Derived table becomes another fact table within the universe



Advanced Calculations

- Derived table SQL

```
SELECT
   MP_T1_Sales.Cat_Id,(sum(MP_T1_Sales
   .Sales)/Company.All_Sales ) as
   Cat_Sales_Ratio
FROM MP_T1_Sales,
    ( SELECT sum(MP_T1_Sales.Sales) as
   All_Sales FROM MP_T1_Sales ) as
   Company
GROUP BY MP_T1_Sales.Cat_Id,
   Company.All_Sales
```

Advanced Calculations

- Derived table SQL
  - The derived table also contains a derived table
  - The interior derived table returns the total of all sales across all categories

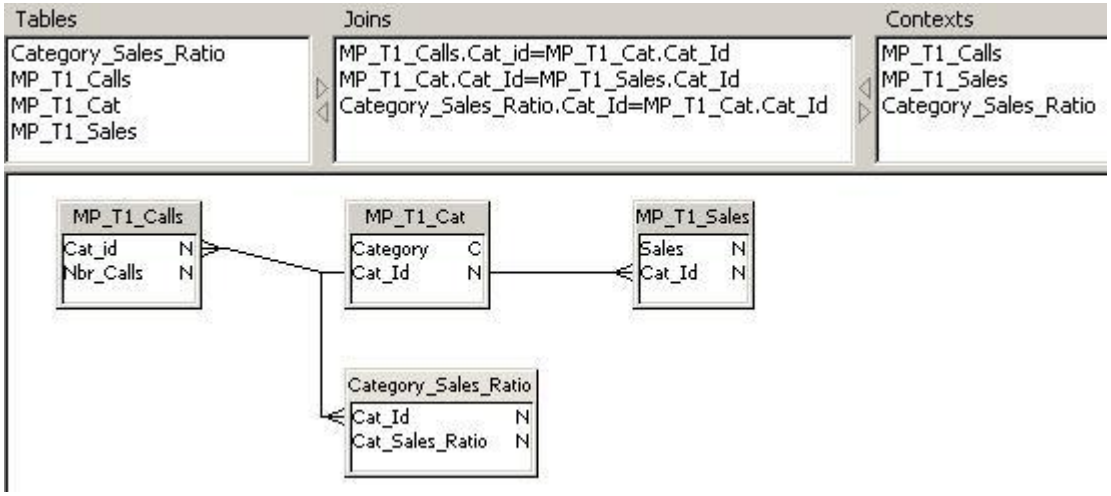
```
SELECT
   MP_T1_Sales.Cat_Id,(sum(MP_T1_Sales
   .Sales)/Company.All_Sales ) as
   Cat_Sales_Ratio
FROM MP_T1_Sales,
    ( SELECT sum(MP_T1_Sales.Sales) as
   All_Sales FROM MP_T1_Sales ) as
   Company
GROUP BY MP_T1_Sales.Cat_Id,
   Company.All_Sales
```

Advanced Calculations

- Derived table SQL
  - The derived table also contains a derived table
  - The interior derived table returns the total of all sales across all categories
  - The primary select uses the overall sales total to calculate the sales percentage by category

```
SELECT
   MP_T1_Sales.Cat_Id,(sum(MP_T1_Sales
   .Sales)/Company.All_Sales ) as
   Cat_Sales_Ratio
FROM MP_T1_Sales,
    ( SELECT sum(MP_T1_Sales.Sales) as
   All_Sales FROM MP_T1_Sales ) as
   Company
GROUP BY MP_T1_Sales.Cat_Id,
   Company.All_Sales
```

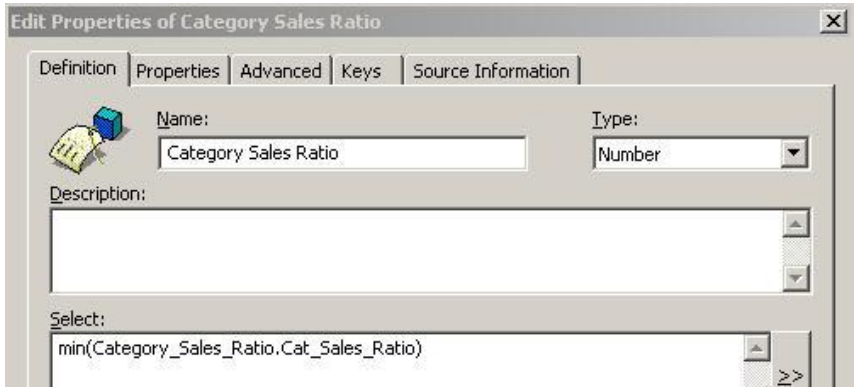
Advanced Calculations



- Derived table used as a fact table within the universe
  - Include in context when appropriate

Advanced Calculations

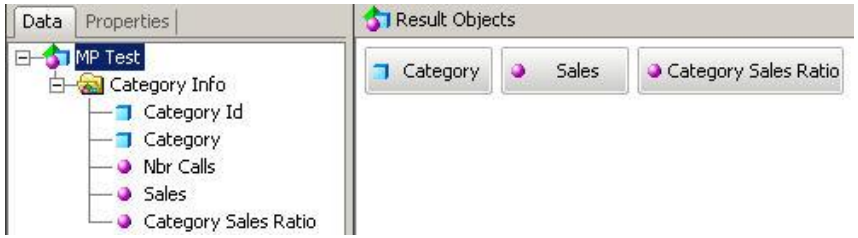
- Specify a database aggregation on the universe object
  - Objects becomes a measure
  - Results in object being omitted from GROUP BY in generated SQL
  - No affect on results returned from database
  - As there is only one row per category, the database aggregations of *avg*, *max*, *min*, and *sum* can all be used



Cat_Id	Cat_Sales_Ratio
1	.197123
2	.054018
3	.179576
4	.057816
5	.438418
6	.027172
7	.045874

Advanced Calculations

- Within the query panel, the ratio object is used as any other object



- Results returned are correct, the same as if a report variable had been used
  - No further actions required from report author or adhoc user

Category	Sales	Category Sales Ratio
Electronics	39,915	19.71%
Food	10,938	5.40%
Gifts	36,362	17.96%
Health & Beauty	11,707	5.78%
Household	88,774	43.84%
Kid's Korner	5,502	2.72%
Travel	9,289	4.59%

Advanced Calculations

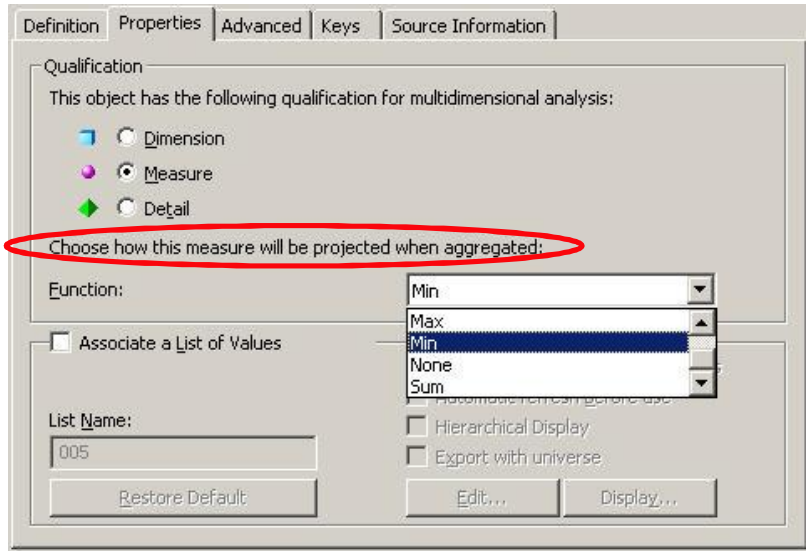
- Same objects used in query panel as before
- Pass 1 - Result set of this SELECT becomes known as *Company,* which is used to calculate the total overall sales that is then labeled *All_Sales*.
- Pass 2 - Result set of this SELECT becomes known as *Cat_Sales_Ratio*, which uses *All_Sales* from the first pass to calculate the sales ratio by category.
- Pass 3 – Primary SELECT pulls results together with the category descriptions

```
SELECT
  MP_T1_Cat.Category, min(
   Category_Sales_Ratio.Cat_Sales_Ratio)
FROM MP_T1_Cat,
  ( SELECT MP_T1_Sales.Cat_Id,
   (sum(MP_T1_Sales.Sales)/Company.All_S
   ales) as Cat_Sales_Ratio
FROM MP_T1_Sales,
    ( SELECT sum(MP_T1_Sales.Sales) as
   All_Sales FROM MP_T1_Sales ) as
   Company
GROUP BY MP_T1_Sales.Cat_Id,
   Company.All_Sales )
   Category_Sales_Ratio
WHERE (
   Category_Sales_Ratio.Cat_Id=MP_T1_Cat
   .Cat_Id )
GROUP BY MP_T1_Cat.Category
```

Advanced Calculations

- *Category Sales Ratio* only has meaning when used *Category* object
- Two concerns arise from this
  - Nothing forces the two objects to be used together in the query panel
  - Nothing prevents the end user from deleting the *Category* column from an existing report

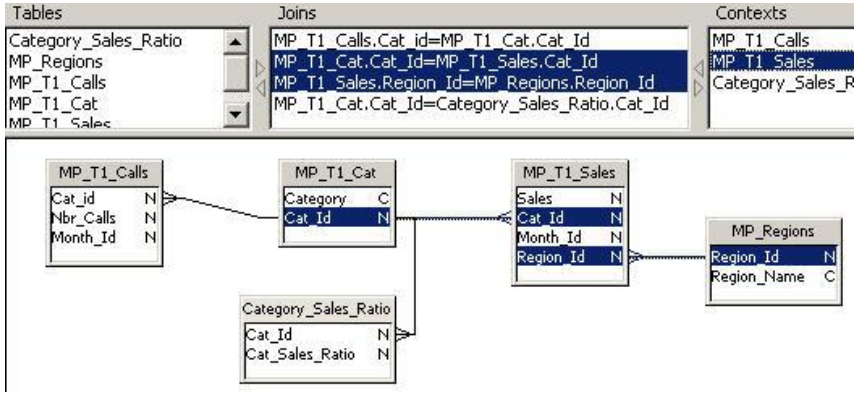
Advanced Calculations



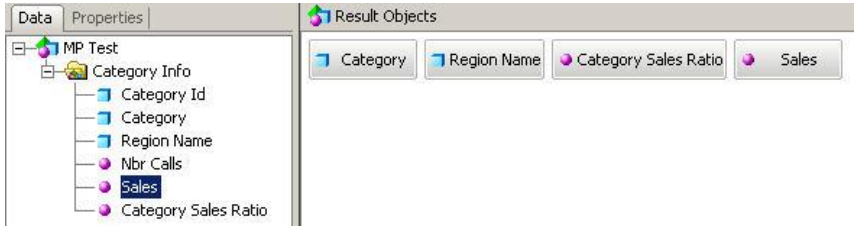
- Properties tab of a measure object
  - *Choose how this measure will be projected when aggregated*
  - Options are: Average, Count, Max, Min, None, Sum
    - Avoid settings of *Average, Max, Min,* and *Count*
  - Determines report level aggregation after results returned by database
  - No affect on SQL generation or database aggregation
  - Default setting will be database aggregation used on the definition tab

Advanced Calculations

- For testing a *region* table is added to the schema and included in the context



- For all projection aggregate examples, the same query is used



Advanced Calculations

- The initial results return data in two blocks as the *Region Name* dimension is not associated with the derived table containing *Category Sales Ratio*
- When *Category Sales Ratio* is placed into the same block as *Region Name*, the projection aggregate is used to determine how to calculate the ratio across category and region
  - As long as *Category* remains in the report block, all projection aggregates settings react the same

Category	Region Name	Sales
Electronics	Central	16,921
Electronics	East	22,994
Food	Central	10,938
Gifts	West	36,362
Health & Beauty	Central	11,707
Household	East	88,774
Kid's Korner	West	5,502
Travel	East	9,289

Category	Category Sales Ratio
Electronics	19.71 %
Food	5.40 %
Gifts	17.96 %
Health & Beauty	5.78 %
Household	43.84 %
Kid's Korner	2.72 %
Travel	4.59 %

Category	Region Name	Category Sales Ratio	Sales
Electronics	Central	19.71 %	16,921
Electronics	East	19.71 %	22,994
Food	Central	5.40 %	10,938
Gifts	West	17.96 %	36,362
Health & Beauty	Central	5.78 %	11,707
Household	East	43.84 %	88,774
Kid's Korner	West	2.72 %	5,502
Travel	East	4.59 %	9,289

Advanced Calculations

- Projection aggregate of *Average*
  - Seven (7) categories
  - Sum of *Category Sales Ratio* is 100%
  - 100% / 7 = 14.29%

Region Name	Category Sales Ratio	Sales
Central	14.29 %	39,566
East	14.29 %	121,057
West	14.29 %	41,864

- Projection aggregate of *Count*
  - Seven (7) categories
  - As field is formatted as a percentage, 700% is displayed

Region Name	Category Sales Ratio	Sales
Central	700.00 %	39,566
East	700.00 %	121,057
West	700.00 %	41,864

Advanced Calculations

- Projection aggregate of *Maximum*
  - 43.84% is largest ratio of all *Category Sales Ratio* values
  - Belongs to the Household category

Category	Region Name	Category Sales Ratio	Sales
Electron			16,921
Electron	Central	43.84 %	39,566
Food	East	43.84 %	121,057
Gifts	West	43.84 %	41,864
Health & Beauty	Central	5.78 %	11,707
Household	East	43.84 %	88,774
Kid's Korner	West	2.72 %	5,502
Travel	East	4.59 %	9,289

- Projection aggregate of *Minimum*
  - 2.72% is the smallest of all the *Category Sales Ratio* values
  - Belongs to the Kids Korner category

Category	Region Name	Category Sales Ratio	Sales
Electronics	Central	19.71 %	16,921
Electroni			22,994
Food	Central	2.72 %	39,566
Gifts	East	2.72 %	121,057
Health &	West	2.72 %	41,864
Household			88,774
Kid's Korner	West	2.72 %	5,502
Travel	East	4.59 %	9,289

Advanced Calculations

- Projection aggregate of *Sum*
  - Sum of ratio across all categories is 100%

Region Name	Category Sales Ratio	Sales
Central	100.00 %	39,566
East	100.00 %	121,057
West	100.00 %	41,864

- Projection aggregate of *None*
  - No aggregate projection takes place
  - *Category Sales Ratio* is not recalculated or redistributed
  - In some aspects a setting of *None* causes the *Category Sales Ratio* to be treated as dimension once the category is removed
    - Original ratio values are maintained preventing *Sales* from being aggregated to the region level

Region Name	Category Sales Ratio	Sales
Central	19.71 %	16,921
Central	5.40 %	10,938
Central	5.78 %	11,707
East	19.71 %	22,994
East	43.84 %	88,774
East	4.59 %	9,289
West	17.96 %	36,362
West	2.72 %	5,502

Advanced Calculations

- Compare with report variables
  - Without the use of the *ForEach*, *ForAll*, *Where*, or *In* context operators a report variable will be projected across all dimensions in the report block
  - Sales Percentage is by region and by category

Region Name	Category	Sales	Percentage
Central	Electronics	16,921	8.36%
Central	Food	10,938	5.40%
Central	Health & Beau	11,707	5.78%
East	Electronics	22,994	11.36%
East	Household	88,774	43.84%
East	Travel	9,289	4.59%
West	Gifts	36,362	17.96%
West	Kid's Korner	5,502	2.72%
		Percentage:	100.00%

- If category is removed then ratio is by region only

Region Name	Sales	Percentage
Central	39,566	19.54%
East	121,057	59.79%
West	41,864	20.67%
	Percentage:	100.00%

Advanced Calculations

- Is Report Variable Multi-Pass?
  - Initial passes are at the database level
  - Subsequent passes for calculations occur at the reporting tier
    - Utilize reporting tier for advanced functionality
  - Derived table utilizes the more traditional approach
    - All work done at the database

Advanced Calculations - Recap

- Derived Table Method
  - Reuse of formula via the Universe
    - Maintains one version of the truth
  - Ease of use as calculation is object available for query panel
  - Single location for maintenance if formula changes
  - Projection aggregate controls behavior when associated dimensions not in report block
    - *None* or *Sum* should be used to maintain proper behavior
- Report Variable Method
  - Recalculates according to dimensions in block
    - Context operators available if behavior is not desired
  - Does not require Universe development to implement new calculations

Grains of Measurement

- Compare measures across timeframes
  - Compare the number of units sold this month with the number of units sold last month
- Business Objects offers many solutions
  - One query for each timeframe
  - Case statement parses date for appropriate timeframe
  - Multiple SELECT statement generation
    - Focus on this solution

Grains of Measurement

- Traditionalist approach
  - Create a temp table with columns of Cat_ID, This_Month_Calls, and Last_Month_Calls
  - Insert number of calls by category in table for the current month
  - Insert number of calls by category in table for the previous month
  - Select results for report
  - Drop table

```
CREATE TABLE This_and_Last
    (Cat_Id integer, This_Month_Calls integer,
    Last_Month_Calls integer)
```

```
INSERT INTO This_and_Last (Cat_Id,
    This_Month_Calls, Last_Month_Calls)
    SELECT MP_T1_Calls.Cat_ID,
    sum(MP_T1_Calls.Nbr_Calls), 0
    FROM MP_T1_Calls, MP_T1_Calendar
WHERE (MP_T1_Calendar.Month_ID =
    MPP_T1_Calls.Month_Id)and
    (MP_T1_Calendar.Month_Rpt_Desc ='Current
    Month')
    GROUP BY  MP_T1_Calls.Cat_Id
```

```
INSERT INTO This_and_Last (Cat_Id,
    This_Month_Calls, Last_Month_Calls)
    SELECT MP_T1_Calls.Cat_ID, 0,
    sum(MP_T1_Calls.Nbr_Calls)
    FROM MP_T1_Calls, MP_T1_Calendar
WHERE (MP_T1_Calendar.Month_ID =
    MP_T1_Calls.Month_Id)and
    (MP_T1_Calendar.Month_Rpt_Desc = 'Current Month
    - 1')
    GROUP BY  MP_T1_Calls.Cat_Id
```

```
SELECT MP_T1_Cat.Cat_Id, Category,
    sum(This_Month_Calls), sum(Last_Month_Calls)
FROM This_and_Last JOIN MP_T1_Cat
ON MP_T1_Cat.Cat_Id = This_and_Last.Cat_Id
GROUP BY MP_T1_Cat.Cat_Id, Category
ORDER BY MP_T1_Cat.Cat_Id
```

```
DROP TABLE This_and_Last
```

Grains of Measurement

- Data inserted into the temporary table contains separate rows for This_Month_Calls and Last_Month_Calls
- Once pulled together for report, results are correct
- Process is very database centric
  - Very similar to the Sharing of Dimensions scenario

	Cat_Id	This_Month_Calls	Last_Month_Calls
1	1	25	0
2	1	0	5
3	2	6	0
4	2	0	4
5	3	24	0
6	3	0	36
7	4	0	18
8	4	12	0
9	5	0	30
10	5	25	0
11	6	0	20
12	6	40	0
13	7	0	7
14	7	14	0

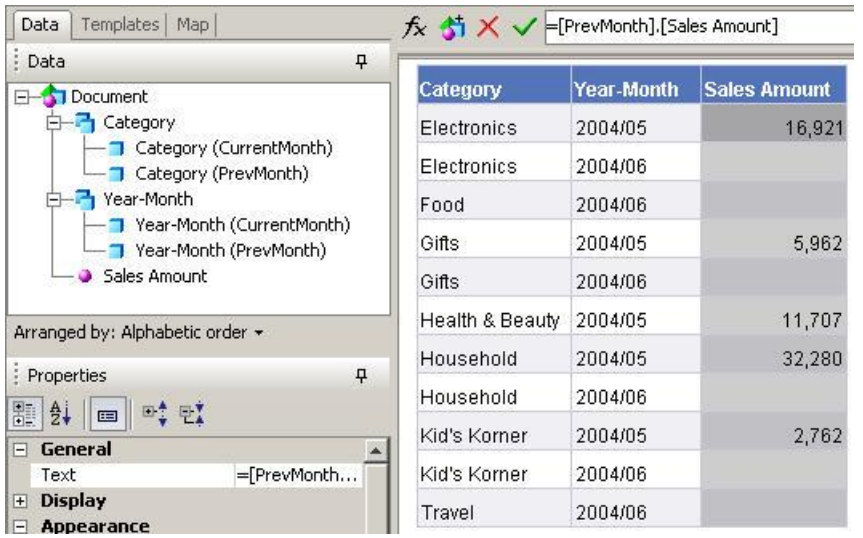
	CAT_ID	CATEGORY	Sum(This_Month_Calls)	Sum(Last_Month_Calls)
1	1	Electronics	25	5
2	2	Food	6	4
3	3	Gifts	24	36
4	4	Health & Beauty	12	18
5	5	Household	25	30
6	6	Kid's Korner	40	20
7	7	Travel	14	7

Grains of Measurement

- One query for each timeframe
  - Queries usually identical except for their filters

Grains of Measurement

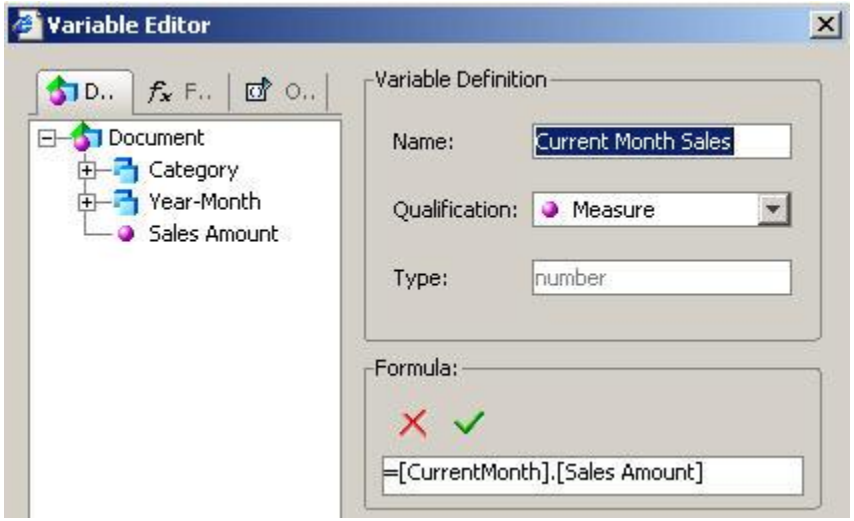
- One query for each timeframe
  - Queries usually identical except for their filters
  - Initial results can be strange
    - Depends on SP level
    - Dimensions are combined
    - Measures are not

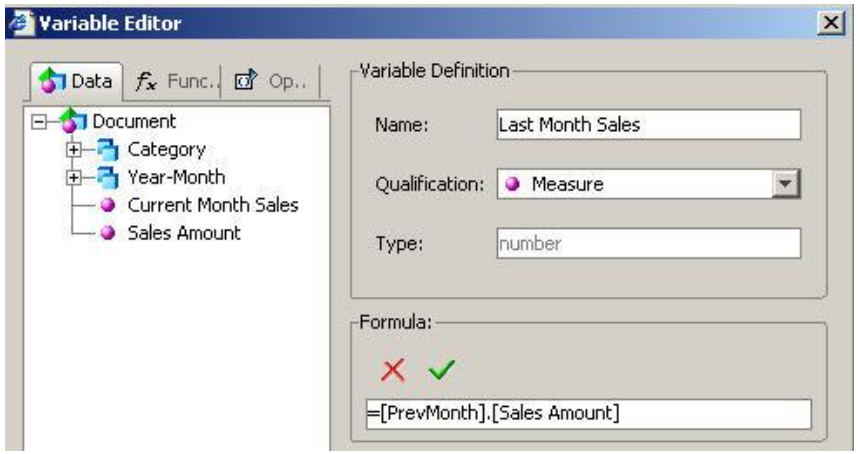


Category	Year-Month	Sales Amount
Electronics	2004/05	16,921
Electronics	2004/06	
Food	2004/06	
Gifts	2004/05	5,962
Gifts	2004/06	
Health & Beauty	2004/05	11,707
Household	2004/05	32,280
Household	2004/06	
Kid's Korner	2004/05	2,762
Kid's Korner	2004/06	
Travel	2004/06	

Grains of Measurement

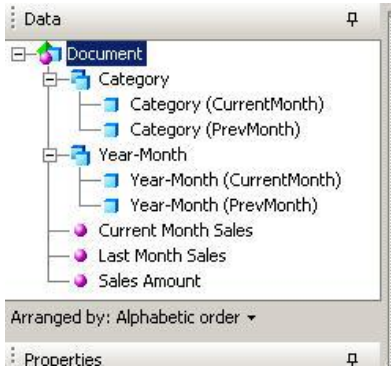
- One query for each timeframe
  - Queries usually identical except for their filters
  - Initial results can be strange
    - Depends on SP level
    - Dimensions are combined
    - Measures are not
  - Report variables required
    - Use measures from each query
    - Meaningful column headers





Grains of Measurement

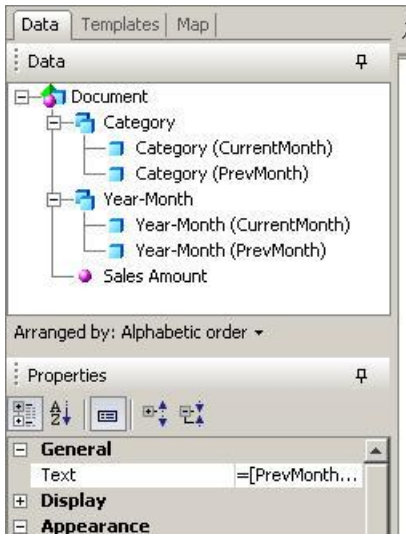
- One query for each timeframe
  - Queries usually identical except for their filters
  - Initial results can be strange
    - Depends on SP level
    - Dimensions are combined
    - Measures are not
  - Report variables required
    - Use measures from each query
    - Meaningful column headers
    - Variables provide final results



Category	Current Month Sales	Last Month Sales
Electronics	22,994	16,921
Food	10,938	
Gifts	30,400	5,962
Health & Beauty		11,707
Household	56,494	32,280
Kid's Korner	2,740	2,762
Travel	9,289	

Grains of Measurement

- One query for each timeframe
  - Queries usually identical except for their filters
  - Initial results can be strange
    - Depends on SP level
    - Dimensions are combined
    - Measures are not
  - Report variables required
    - Use measures from each query
    - Meaningful column headers
    - Variables provide final results
  - Not a graceful solution for ad hoc user; Burden placed on report developer
  - For numerous requests becomes a tiresome process
  - Prompts for each query increases potential for erroneous input

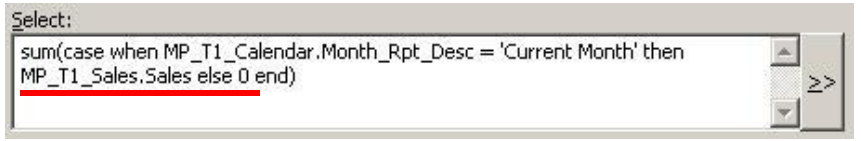


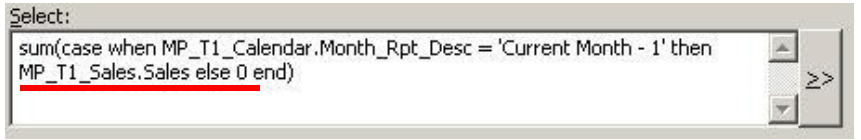
=[PrevMonth].[Sales Amount]

Category	Year-Month	Sales Amount
Electronics	2004/05	16,921
Electronics	2004/06	
Food	2004/06	
Gifts	2004/05	5,962
Gifts	2004/06	
Health & Beauty	2004/05	11,707
Household	2004/05	32,280
Household	2004/06	
Kid's Korner	2004/05	2,762
Kid's Korner	2004/06	
Travel	2004/06	

Grains of Measurement

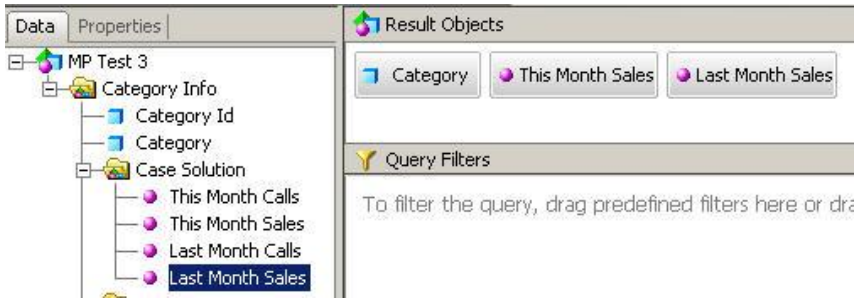
- Need separate universe objects which represent each timeframe
  - Removes filter from query panel
  - Eliminate the need for report variables
- Common solution: Use CASE statement in object definition to parse time span
  - Case statement parses date for appropriate timeframe
    - Within timeframe, value is added to the measure
    - Outside of timeframe, zero is added to the measure
  - One object for each timeframe





Grains of Measurement

- CASE based objects simplify query panel
  - No query filter needed
  - No prompts
  - Single query created by end user



- Single SELECT generated

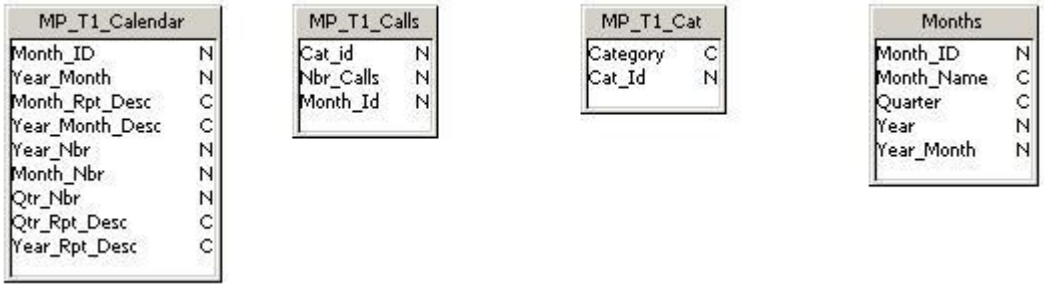
```
SELECT
  MP_T1_Cat.Category,
  sum(case when MP_T1_Calendar.Month_Rpt_Desc = 'Current Month' then MP_T1_Sales.Sales else 0 end),
  sum(case when MP_T1_Calendar.Month_Rpt_Desc = 'Current Month - 1' then MP_T1_Sales.Sales else 0 end)
FROM
  MP_T1_Cat,
  MP_T1_Sales,
  MP_T1_Calendar
WHERE
  ( MP_T1_Cat.Cat_Id=MP_T1_Sales.Cat_Id  )
  AND  ( MP_T1_Calendar.Month_ID=MP_T1_Sales.Month_Id  )
GROUP BY
  MP_T1_Cat.Category
```

Grains of Measurement

Category	This Month Sales	Last Month Sales
Electronics	22,994	16,921
Food	10,938	0
Gifts	30,400	5,962
Health & Beauty	0	11,707
Household	56,494	32,280
Kid's Korner	2,740	2,762
Travel	9,289	0

- CASE based objects return correct results
- Is this answer?
- **NO**
- Issue is performance
  - Every row in the fact table has to be read to determine if time span criteria is met
  - Fact table: 84 rows of which 15 meet time criteria
  - SQL Server Trace shows 85 reads occurring for SELECT using CASE objects
  - For multiple query solution, SQL Server Trace indicates 37 reads per query

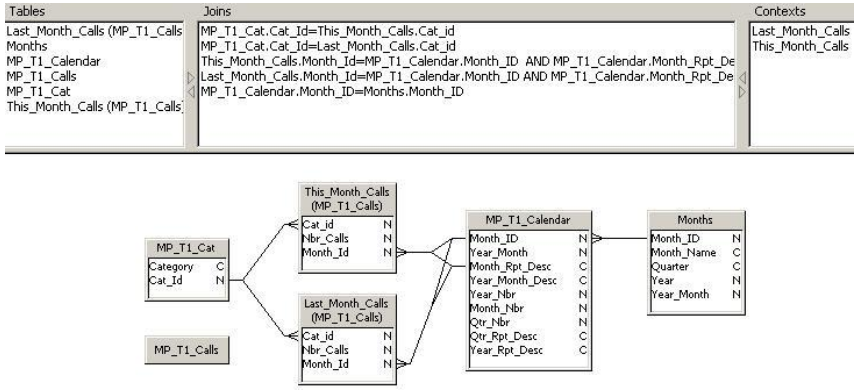
Grains of Measurement



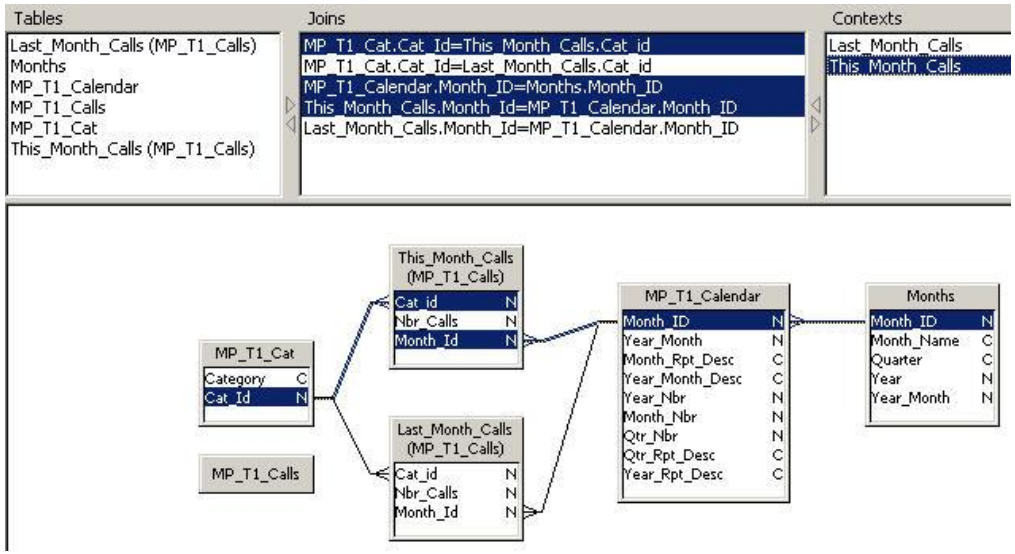
- Combine the good of each approach
  - Separate objects for each timeframe
  - Eliminate query filters
  - Maintain performance
- Force context for each timeframe
  - Alias fact table for each timeframe
  - Measure objects from each aliased fact table
  - Utilize “Where” clause on measure objects

Grains of Measurement

- First, create alias for each fact for each timeframe
  - Alias for this month and previous month
- Insert joins
  - Joins to alias no different than to original table
  - Ensure proper cardinality is identified
    - Allows for automatic context detection
  - Access to alias tables will be determined by object filters



Grains of Measurement



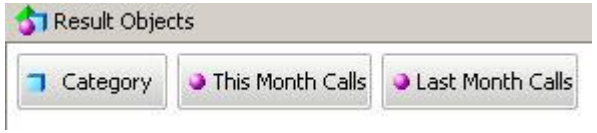
- Create contexts
  - Use *Detect Context* option in Designer if join cardinality has been maintained
  - Delete all contexts and regenerate as new dimension tables are added or fact tables aliased for additional timeframes
- Context created for each time based fact table alias
  - Alias table names are default names of contexts, so name aliases wisely

Grains of Measurement

- Create measures for each timeframe
  - Object names should reflect timeframe
  - Such as *This Month* Calls and *Last Month Calls*
- Where clause enforces timeframe against the date table
  - If moved to the join between alias and date table, the *where* clause only added to the generated SQL when object from date table is also a result object in the query panel

Grains of Measurement

- Alias based objects simplify query panel
  - No query filter needed
  - Single query created by end user
  - No prompts



- Correct results returned

Category	This Month Calls	Last Month Calls
Electronics	25	5
Food	6	4
Gifts	24	36
Health & Beauty	12	18
Household	25	30
Kid's Korner	40	20
Travel	14	7

Grains of Measurement

- Contexts result in separate SELECTs for each timeframe
- If *JOIN_BY_SQL* setting is *Yes*, then one SELECT statement generated and derived tables created for each timeframe
  - Database combines result sets

Grains of Measurement - Recap

- Alias fact table for each desired timeframe
  - All joins to each alias are identical
  - All dimension tables joined to each alias
- Create contexts
  - Make use of *Detect Context* feature in Designer
- Measure objects created from each alias reflect timeframe
  - *Where* clause in object definition enforces timeframe

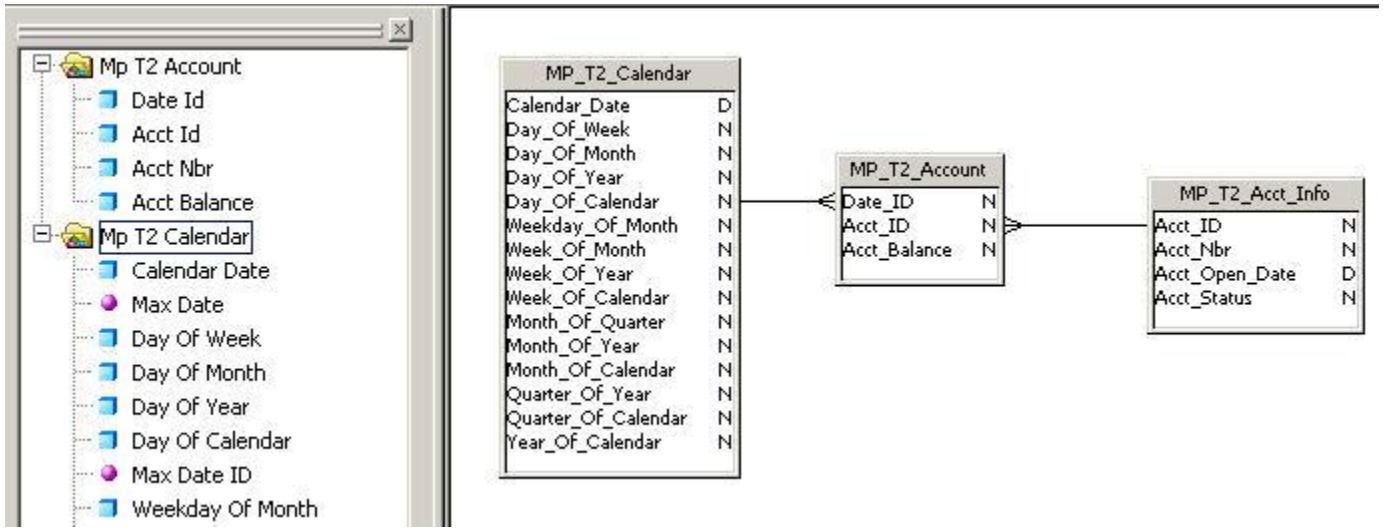
Semi-additive Measures

- Additive Measures
  - Most common
  - Aggregation is applied consistently to all dimensions
  - Measures roll up within a dimensional hierarchy
- Semi-additive Measures
  - Semi-additive measures are additive across some dimensions
  - But are not additive across one or more of the dimensions
    - Time is the usually the exception dimension
    - Other aggregate functions such as average, minimum, maximum may be valid over the dimension but not sum
  - Examples
    - Periodic measures such as account balances
    - Level measures such as inventory and headcount

Semi-additive Measures

- Traditionalist approach
  - OLAP driven
  - No relational solution beyond re-query of database
- For Business Objects, only complete solution exists using Web Intelligence
  - Desktop Intelligence can be used if drilling on the exception dimension is not required
- Requires use of many features
  - Derived tables
  - Aggregate awareness
  - Query drill option

Semi-additive Measures



- Example will be based on 3 database tables
  - Calendar, MP_T2_Calendar, as a dimension
  - Base account table, MP_T2_Info, as a dimension
  - Ending balance by account by date, MP_T2_Account, as a fact

Semi-additive Measures

- Need derived table to reflect last balance entered for each account for each month
  - Date for balance entry should be last date of the month even if entry is for an earlier date
  - Use Business Objects to create the required SQL

Semi-additive Measures

- Create month end derived table by account
  - Returns latest date for each account within the fact table
  - Entry in the ending balance table only on those days having activity

Semi-additive Measures

- Sample results from using the account month end derived table
  - February 2006 month end date for account 101 is 2/28/2006
  - February 2006 month end date for account 104 is 2/27/2006

Acct Id	Month Of Calend	Max Date	Max Date ID
101	1273	1/31/2006	38747
101	1274	2/28/2006	38775
101	1275	3/31/2006	38806
101	1276	4/1/2006	38807

103	1285	1/31/2007	39112
103	1286	2/28/2007	39140
103	1287	3/31/2007	39171
103	1288	4/1/2007	39172
103	1290	6/30/2007	39262
103	1291	7/31/2007	39293
103	1293	9/30/2007	39354
103	1296	12/31/2007	39446
104	1273	1/31/2006	38747
104	1274	2/27/2006	38774
104	1275	3/31/2006	38806

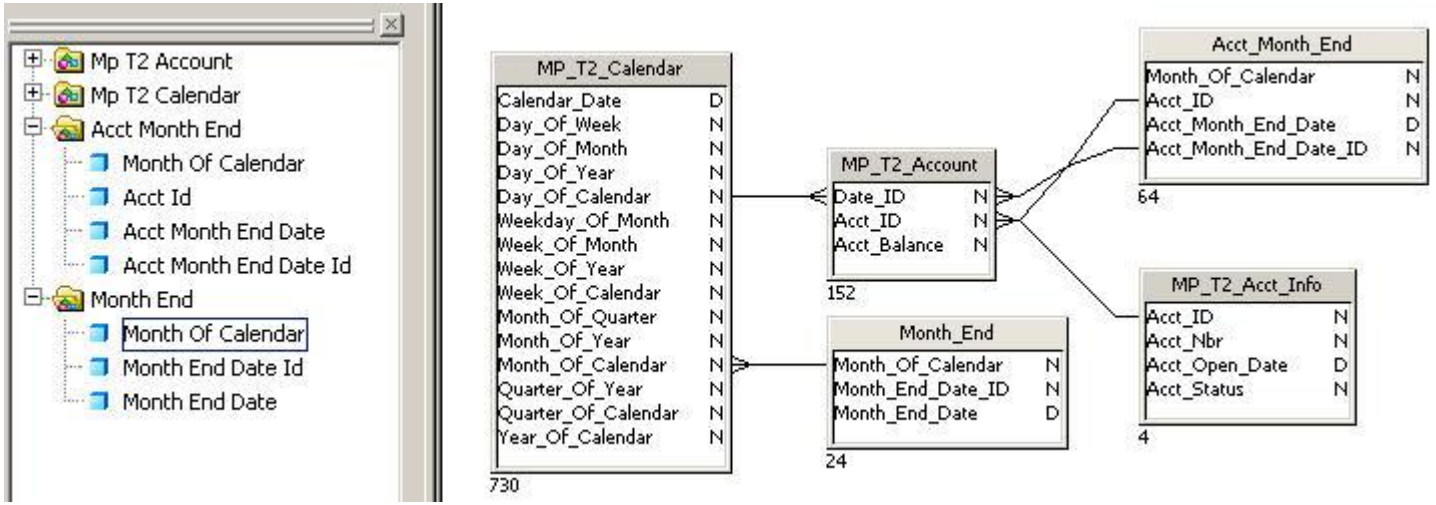
Semi-additive Measures

- Second derived table for month ending dates
  - Used to force ending balance dates to be the same for all accounts
  - Actual month end dates

- Sample results from derived table
  - Month end for February 2006 is 02/28/2006

Month Of Calendar	Max Date ID	Max Date
1273	38747	1/31/2006
1274	38775	2/28/2006
1275	38806	3/31/2006
1276	38836	4/30/2006
1277	38867	5/31/2006
1278	38897	6/30/2006
1279	38928	7/31/2006
1280	38959	8/31/2006

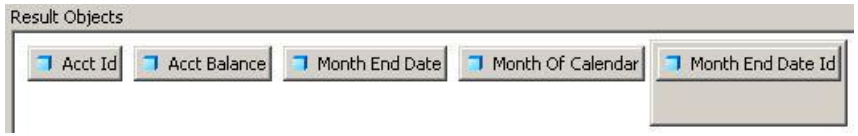
Semi-additive Measures



- Add derived tables to universe
  - Month end derived table is joined to calendar table
  - Month end by account derived table is joined to the fact table

Semi-additive Measures

- Resulting universe used to create query
  - SQL from this query will be used to create the derived table for the reporting universe

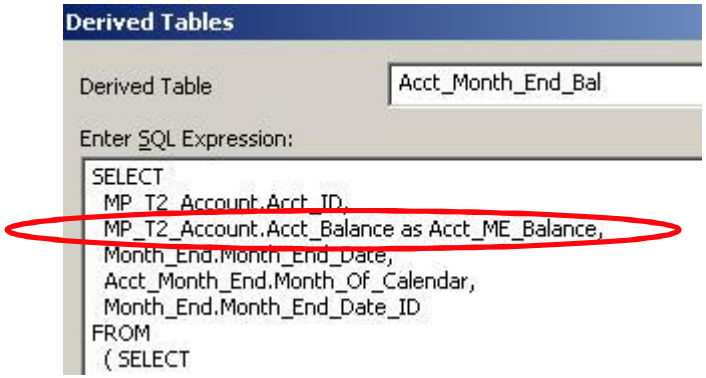


- Sample results from query
  - As previously seen, the last ending balance entry in February 2006 for account 104 was on 02/27/2006. Using the month ending date from the month end derived table in the query has the month end date show as 02/28/2006

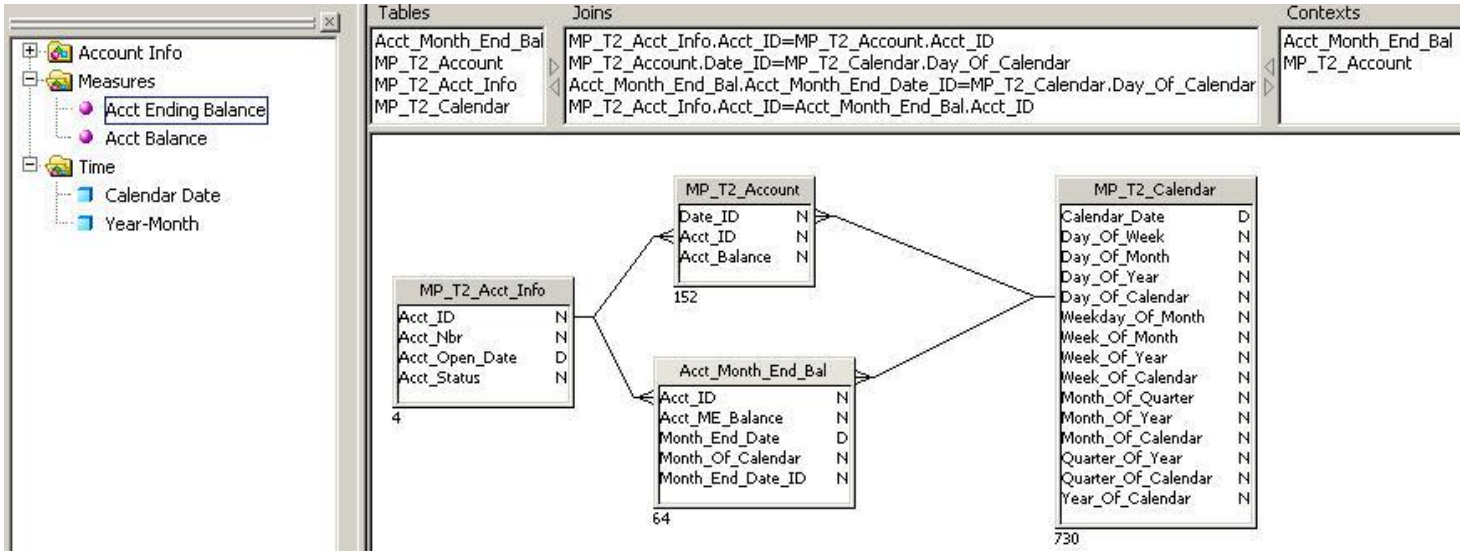
Month End Date	Acct Id	Acct Balance	Month Of Calendar	Month End Date Id
1/31/2006	101	160131	1273	38747
1/31/2006	102	260131	1273	38747
1/31/2006	103	360131	1273	38747
1/31/2006	104	460131	1273	38747
2/28/2006	101	160228	1274	38775
2/28/2006	102	260228	1274	38775
2/28/2006	103	360228	1274	38775
2/28/2006	104	460227	1274	38775
3/31/2006	101	160331	1275	38806

Semi-additive Measures

- SQL used for derived table within reporting universe
  - Rename the account balance column to reflect its month ending value



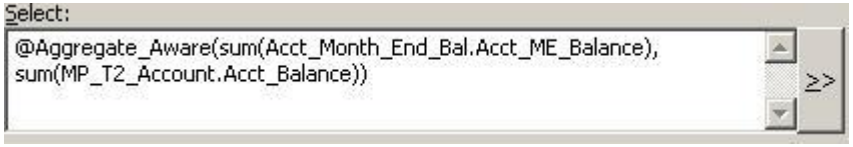
Semi-additive Measures



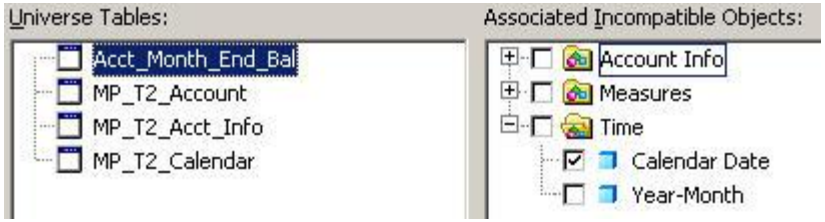
- Add derived table for month end balance to universe
  - Becomes a fact table
  - Join additional dimension tables

Semi-additive Measures

- Create *Acct Ending Bal* object
  - Aggregate aware
  - Uses either the derived table or the ending account balance table



- Setup aggregate navigation
  - Derived table should not be referenced if the *Calendar Date* object is part of the query
  - If *Calendar Date* is used then the ending account balance table (MP_T2_Acount) is referenced



Semi-additive Measures

- Query to return month ending balance for each account

- Results are correct
  - Balances within table MP_T2_Acount are formatted as *AYMMDD* with:
    - A = account id
    - Y = year
    - MM = month
    - DD = day
  - Ending balance for account 104 in February 2006 occurred on the 27th while it occurred for all other accounts on the 28th

Year-Month	Acct Id	Acct Nbr	Acct Ending Balance
2006-01	101	100101	160,131
2006-01	102	100202	260,131
2006-01	103	100337	360,131
2006-01	104	104044	460,131
2006-02	101	100101	160,228
2006-02	102	100202	260,228
2006-02	103	100337	360,228
2006-02	104	104044	460,227
2006-03	101	100101	160,331

Semi-additive Measures

- Generated SQL shows derived table being used

```
SELECT
  cast(MP_T2_Calendar.Year_Of_Calendar as char(4)) + '-' + case when MP_T2_Calendar.M
onth_Of_Year > 9 then cast(MP_T2_Calendar.Month_Of_Year as char(2)) else '0' + cast(M
P_T2_Calendar.Month_Of_Year as char(1)) end,
  MP_T2_Acct_Info.Acct_ID,
  MP_T2_Acct_Info.Acct_Nbr,
  sum(Acct_Month_End_Bal.Acct_ME_Balance)
FROM
```

- Drilling into time hierarchy requires query drill to be enabled
  - Access the document properties
  - Enable *Use query drill*
  - Do not use *Scope of Analysis* on the query panel

Semi-additive Measures

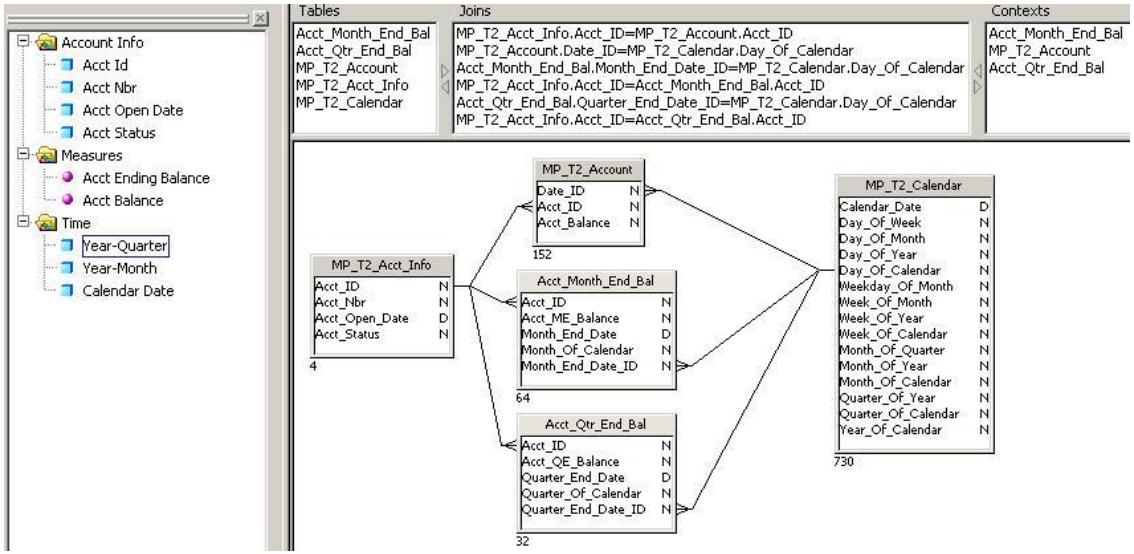
- Drilling down to *Year-Month* reveals balances by date

Calendar D	Acct Id	Acct Nbr	Acct Ending Balance
2/1/06	101	100101	160,201
2/1/06	102	100202	260,201
2/1/06	103	100337	360,201
2/1/06	104	104044	460,201
2/27/06	101	100101	160,227
2/27/06	102	100202	260,227
2/27/06	103	100337	360,227
2/27/06	104	104044	460,227
2/28/06	101	100101	160,228

- Query drill adds *Calendar Date* to the query
  - Since *Calendar Date* is now part of the query, aggregate awareness invokes the balance ending table (MP_T2_Account) in place of the month end derived table

```
SELECT
  cast(MP_T2_Calendar.Year_Of_Calendar as char(4)) + '-' + case when MP_T2_Calendar.M
onth_Of_Year > 9 then cast(MP_T2_Calendar.Month_Of_Year as char(2)) else '0' + cast(M
P_T2_Calendar.Month_Of_Year as char(1)) end,
  MP_T2_Acct_Info.Acct_ID,
  MP_T2_Acct_Info.Acct_Nbr,
  sum(MP_T2_Account.Acct_Balance),
  MP_T2_Calendar.Calendar_Date
FROM
```

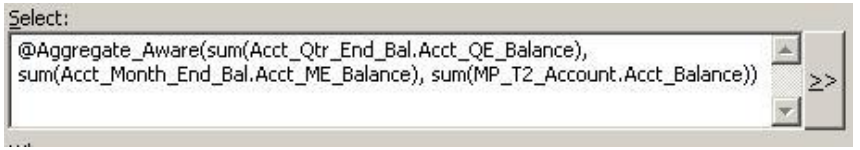
Semi-additive Measures



- The next logical step is add more levels to the time hierarchy
  - Build a derived table for quarter ending balances
    - Modify the existing derived table for month ending balances
    - Use same technique used to develop month ending table
  - Add *Year-Quarter* object to the time hierarchy

Semi-additive Measures

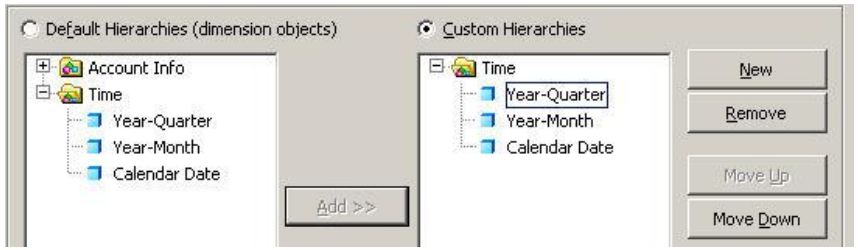
- Modify *Acct Ending Bal* object
  - Add the quarter ending balance table to the aggregate aware



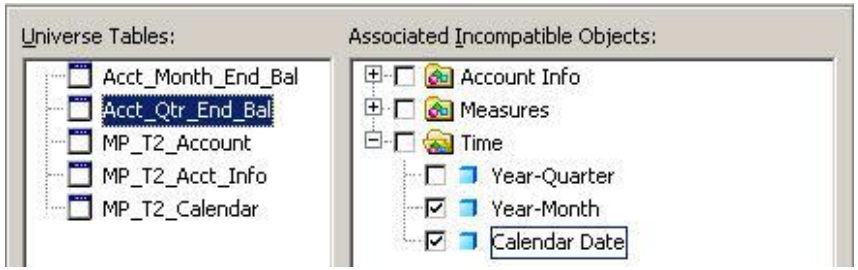
- Create the *Year-Quarter* object

Semi-additive Measures

- Add *Year-Quarter* object to the time hierarchy



- Alter the aggregate navigation
  - The new derived table for quarter ending balances is not compatible with either the *Year-Month* object or the *Calendar Date* object



Semi-additive Measures

- Query to return quarter ending balance for each account

- Results are correct
  - Remember, balances within table MP_T2_Acount are formatted as *AYMMDD* with:
    - A = account id
    - Y = year
    - MM = month
    - DD = day
  - Ending balance for account 104 in 2006Q2 occurred on the June 29th while it occurred for all other accounts on the June 30th

Year-Quarter	Acct Id	Acct Nbr	Acct Ending Balance
2006-Q1	101	100101	160,331
2006-Q1	102	100202	260,331
2006-Q1	103	100337	360,331
2006-Q1	104	104044	460,331
2006-Q2	101	100101	160,630
2006-Q2	102	100202	260,630
2006-Q2	103	100337	360,630
2006-Q2	104	104044	460,629

Semi-additive Measures

- Drilling into Q2 shows the ending balance for each account at the end of each month in Q2
  - Since *Year-Month* is now part of the query, aggregate awareness will use the derived table for month ending balances instead of the quarter ending balances table

Year-Month	Acct Id	Acct Nbr	Acct Ending Balance
2006-04	101	100101	160,401
2006-04	102	100202	260,401
2006-04	103	100337	360,401
2006-04	104	104044	460,401
2006-06	101	100101	160,630
2006-06	102	100202	260,630
2006-06	103	100337	360,630
2006-06	104	104044	460,629

- Drilling into June shows the ending balance for each account for every day a transaction occurred
  - Now that *Calendar Date* is part of the query, aggregate awareness will use the daily ending balance table instead of the month ending derived table

Calendar D	Acct Id	Acct Nbr	Acct Ending Balance
6/28/06	101	100101	160,628
6/28/06	102	100202	260,628
6/28/06	103	100337	360,628
6/28/06	104	104044	460,628
6/29/06	101	100101	160,629
6/29/06	102	100202	260,629
6/29/06	103	100337	360,629
6/29/06	104	104044	460,629
6/30/06	101	100101	160,630
6/30/06	102	100202	260,630
6/30/06	103	100337	360,630

Semi-additive Measures - Recap

- Create derived table for each desired timeframe
  - All joins to each derived table should be identical
  - All dimension tables joined to each derived table
- Create contexts as a precaution
- Object definition for associated measure uses aggregate awareness to navigate across derived tables
- Create time dimensions that reflect granularity of derived tables
- Setup aggregate navigation such that each derived table is compatible only with its time dimension
- Enable *Use query drill* on the document properties within Web Intelligence
  - Do not use *Scope of Analysis* within the query panel

Data Subset

- Analyze a subset of data against the whole
  - Compare the origins of most recent transactions to historical transaction origins – discover a change in user behavior
- Analyze a subset of data that meet specific criteria
  - Locate latest status record of a customer, product and analyze those that meet certain requirements
- Business Objects offers many solutions
  - Derived tables used as a filter to obtain subset
  - Separate queries for each subset
  - Or use *Grains of Measurement* methodology to refine

Data Subset

- Traditionalist approach
  - Create a work table for most recent transaction date for each account
  - Obtain the most recent transaction date for each account
  - Create a work table for the historical and recent transaction counts by channel
  - Insert number of recent transactions for each account by channel
  - Insert total number of transactions by channel
  - Select results for report
  - Drop work tables

```
CREATE TABLE LAST_TRANS
   (Acct_Nbr char(16), Tran_Date date)
```

```
INSERT INTO LAST_TRANS (Acct_Nbr, Tran_Date)
   SELECT Acct_Nbr, max(Tran_Date)
     FROM checking_tran
 GROUP BY Acct_Nbr
```

```
CREATE TABLE TRANS_COUNTS
    (CHANNEL CHAR(1), HIST_CNT INTEGER,
    RECENT_CNT INTEGER)
```

```
INSERT INTO TRANS_COUNTS (CHANNEL, HIST_CNT,
    RECENT_CNT)
    SELECT a.CHANNEL, 0, COUNT(a.TRAN_ID)
    FROM   checking_tran a, LAST_TRANS t
    WHERE  a.acct_nbr = t.Acct_Nbr
    AND    a.tran_date = t.Tran_Date
    GROUP BY a.channel
```

```
INSERT INTO TRANS_COUNTS (CHANNEL, HIST_CNT,
    RECENT_CNT)
    SELECT a.CHANNEL, COUNT(a.TRAN_ID), 0
    FROM   checking_tran a
    GROUP BY  a.channel
```

```
SELECT c.channel_descr, sum(t.HIST_CNT),
    sum( t.RECENT_CNT)
FROM channel_descr c, TRANS_COUNTS t
WHERE c.channel_nbr = t.channel
GROUP BY c.channel_descr
ORDER BY c.channel_descr
```

```
DROP TABLE LAST_TRANS

DROP TABLE TRANS_COUNTS
```

Data Subset

- The LAST_TRANS work table contains one entry for each account representing the last transaction date for that account.

	Acct_Nbr	Tran_Date
1	0000000013624802	12/30/1995
2	0000000013624842	12/31/1995
3	0000000013624852	12/31/1995
4	0000000013624862	12/17/1995
5	0000000013624872	12/12/1995
6	0000000013624882	12/30/1995
7	0000000013624892	12/23/1995
8	0000000013624922	12/28/1995
9	0000000013624982	12/31/1995

- The TRANS_COUNTS work table contains separate rows for the historical and most recent counts

	channel_descr	HIST_CNT	RECENT_CNT
1	ACH	9972	0
2	ACH	0	127
3	Branch	0	202
4	Branch	20317	0
5	Check	0	100
6	Check	10714	0
7	Electronic	0	90
8	Electronic	9037	0
9	Internet	4496	0
10	Internet	0	37
11	Other	0	60
12	Other	6427	0
13	Paper	43443	0
14	Paper	0	343
15	Wire	0	79
16	Wire	8583	0

Data Subset

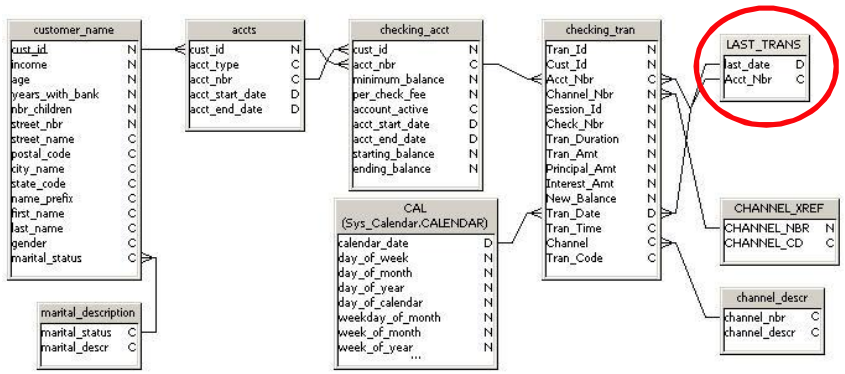
	channel_descr	Sum(HIST_CNT)	Sum(RECENT_CNT)
1	ACH	9972	127
2	Branch	20317	202
3	Check	10714	100
4	Electronic	9037	90
5	Internet	4496	37
6	Other	6427	60
7	Paper	43443	343
8	Wire	8583	79

- Once the results are pulled together from TRANS_COUNTS, the numbers are correct
- Process is very database centric
  - 8 SQL statements sent to database
  - 2 work tables used to house temporary results
  - Very similar to the Grains of Measurement scenario
    - Only worse

Data Subset

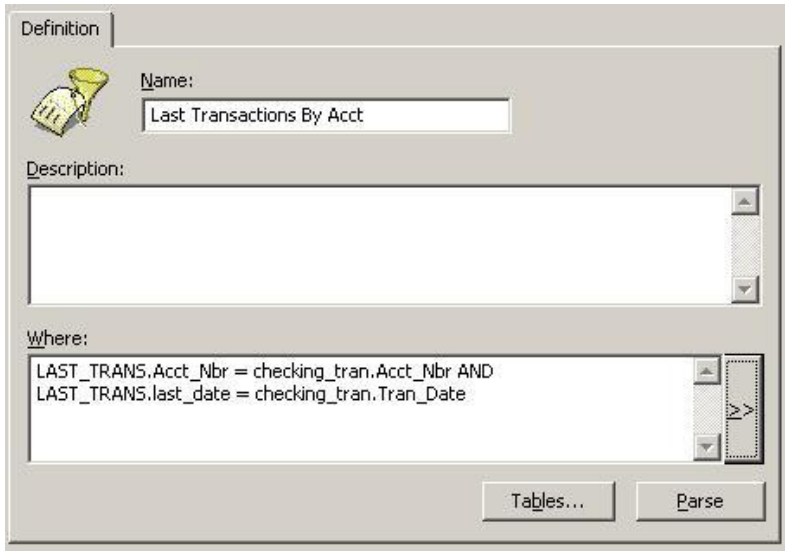
- Derived table *LAST_TRANS*
  - Returns most recent date for each account number on which a transaction took place

- Add to universe
  - Join to the transaction table
  - `checking_tran.Acct_Nbr = LAST_TRANS.Acct_Nbr AND checking_tran.Tran_Date = LAST_TRANS.last_date`



Data Subset

- Create a new filter object
  - Specifies the join between the derived table (LAST_TRANS) and the transaction table



Data Subset

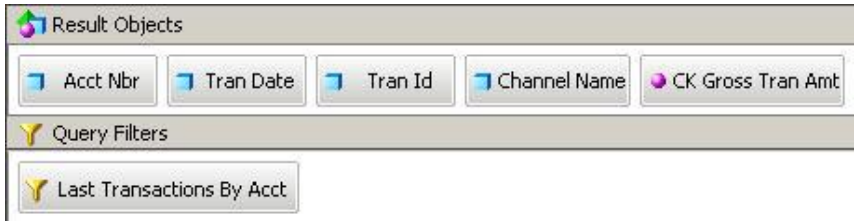
- Does the filter work?
  - List all transactions by account

- Query returns all transactions for each account as expected

Acct Nbr	Tran Date	Tran Id	Channel Name	CK Gross Tran Amt
0000000013624862	1/4/95	1	Paper	$38.70
0000000013624862	1/11/95	2	Branch	$379.28
0000000013624862	4/8/95	7	Paper	$67.32
0000000013624862	4/19/95	8	Check	$0.00
0000000013624862	7/14/95	12	Electronic	$47.61
0000000013624862	7/25/95	13	Internet	$0.00
0000000013624862	8/13/95	15	Paper	$18.30
0000000013624862	9/12/95	17	Paper	$123.76

Data Subset

- Does the filter work?
  - Same query as before but with filter object added



- Query returns only transactions which occurred on the most recent day for each account

Acct Nbr	Tran Date	Tran Id	Channel Name	CK Gross Tran Amt
0000000013624862	12/17/95	24	Other	$15.54
0000000013624892	12/23/95	161	Wire	$0.00
0000000013624982	12/31/95	56	Paper	$191.11
0000000013625002	12/25/95	18	Electronic	$691.07
0000000013625032	12/23/95	132	Paper	$19.56
0000000013625512	12/29/95	78	Paper	$228.89
0000000013626052	12/23/95	81	ACH	$17.99

Data Subset

- Generated SQL for the first query is fairly simple

```
SELECT
  checking_acct.acct_nbr,
  checking_tran.Tran_Date,
  checking_tran.Tran_Id,
  channel_descr.channel_descr,
  sum (checking_tran.Tran_Amt)
FROM
  checking_tran,
  checking_acct,
  channel_descr
WHERE
  ( checking_acct.acct_nbr=checking_tran.Acct_Nbr  )
  AND  ( checking_tran.Channel=channel_descr.channel_nbr  )
GROUP BY
  1,
  2,
  3,
  4
```

- Generated SQL for the second query is similar to the first but…
  - Derived table add to the FROM clause
  - Filter object enforces join in the WHERE clause which restricts transactions to the most recent day for each account

```
SELECT
  checking_acct.acct_nbr,
  checking_tran.Tran_Date,
  checking_tran.Tran_Id,
  channel_descr.channel_descr,
  sum (checking_tran.Tran_Amt)
FROM
  checking_tran,
  checking_acct,
  (
  select max(tran_date) as last_date, acct_nbr from checking_tran group by acct_nbr
  ) LAST_TRANS,
  channel_descr
WHERE
  ( checking_acct.acct_nbr=checking_tran.Acct_Nbr  )
  AND  ( checking_tran.Acct_Nbr=LAST_TRANS.Acct_Nbr and checking_tran.Tran_Date=LAST_TRANS.last_date  )
  AND  ( checking_tran.Channel=channel_descr.channel_nbr  )
  AND
  ( LAST_TRANS.Acct_Nbr = checking_tran.Acct_Nbr AND LAST_TRANS.last_date = checking_tran.Tran_Date  )
GROUP BY
  1,
  2,
  3,
  4
```

Data Subset

- Historical query
  - Count of transactions by channel across all transactions

- Recent transactions
  - Same as the historical query but with filter object specified
  - Count of transactions by channel across the most recent transaction for each account

Data Subset

- Each query returns a separate block
  - Counts for each channel category higher for historical query

Channel Name	Trans Count
ACH	9,972
Branch	20,317
Check	10,714
Electronic	9,037
Internet	4,496
Other	6,427
Paper	43,443
Wire	8,583

Channel Name	Trans Count
ACH	127
Branch	202
Check	100
Electronic	90
Internet	37
Other	60
Paper	343
Wire	79

Data Subset

- Additional formatting required
  - Add percentage for each channel
  - Add additional header row to each block to label which block represents which count

Historical		
Channel Name	Trans Count	Percentage
ACH	9,972	8.83%
Branch	20,317	17.98%
Check	10,714	9.48%
Electronic	9,037	8.00%
Internet	4,496	3.98%
Other	6,427	5.69%
Paper	43,443	38.45%
Wire	8,583	7.60%
	Percentage:	100.00%

Most Recent		
Channel Name	Trans Count	Percentage
ACH	127	12.24%
Branch	202	19.46%
Check	100	9.63%
Electronic	90	8.67%
Internet	37	3.56%
Other	60	5.78%
Paper	343	33.04%
Wire	79	7.61%
	Percentage:	100.00%

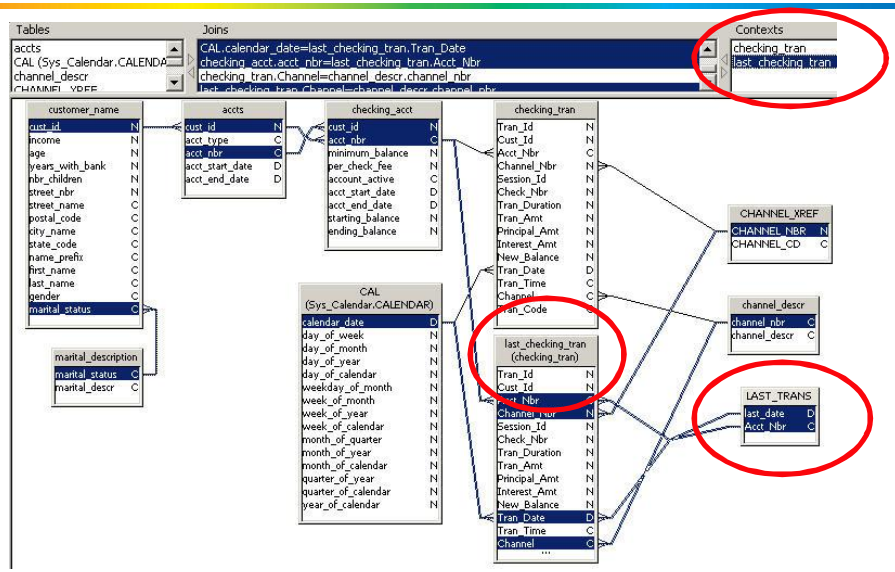
- Or with a bit more work
  - Combine into a single block
  - Add additional header row to label which count is which
  - Add percentages

	Historical		Most Recent	
Channel Name	Trans Count	Percentage	Trans Count	Percentage
ACH	9,972	8.83%	127	12.24%
Branch	20,317	17.98%	202	19.46%
Check	10,714	9.48%	100	9.63%
Electronic	9,037	8.00%	90	8.67%
Internet	4,496	3.98%	37	3.56%
Other	6,427	5.69%	60	5.78%
Paper	43,443	38.45%	343	33.04%
Wire	8,583	7.60%	79	7.61%
	Percentage:	100.00%		100.00%

Data Subset

- Solution #1
  - Derived table returns most recent transaction day by account
  - Join derived table to transaction history table
  - Filter object which references join between derived table and transaction history table
- Issues with current solution – Burden on report builder
  - Multiple queries required
  - Extensive formatting required once result sets are returned
- Refinement to current solution
  - Alias transaction table
  - Derived table joined only to the transaction table alias
  - Create contexts
  - Separate objects for historical and recent transaction counts
  - Similar techniques as used in Grains of Measurement scenario

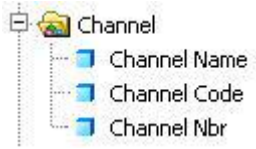
Data Subset



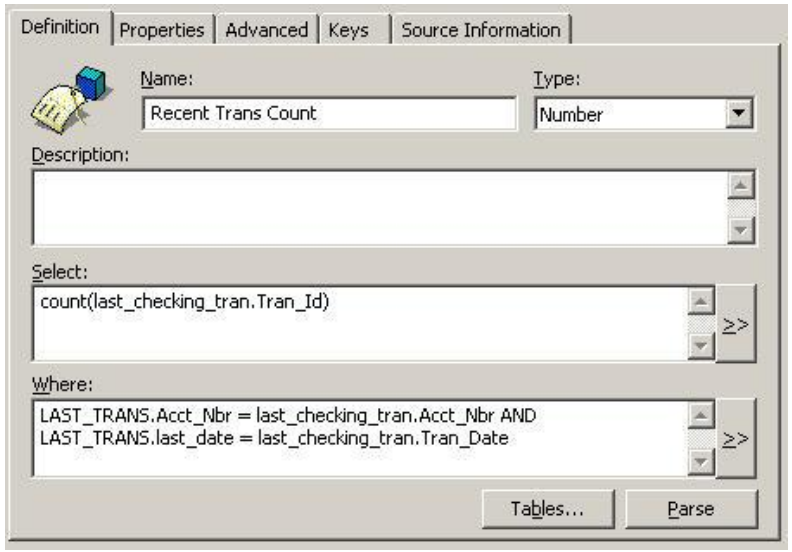
- Alias the transaction table
  - Join same tables to the alias as joined to the transaction table itself
  - *LAST_TRANS d*erived table is joined only to the alias
  - Detect/create contexts

Data Subset

- Create channel class
  - Move objects from the transaction class
  - All objects reference the dimension tables joined to the transaction table and the newly aliased transaction table



- New measure object which counts only the recent
  - Includes only the most recent transactions for each account
  - *Where* clause will enforce join to the derived table (*LAST_TRANS*)



Data Subset

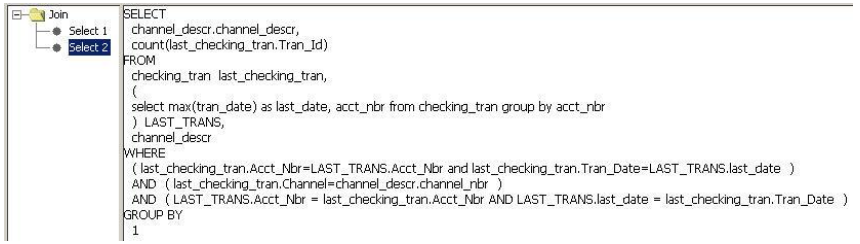
- Single query
  - *Trans Count* counts all transactions
  - *Recent Trans Count* counts only the most recent transactions on each account
  - No filter required as it is part of the *Recent Trans Count* object

- Initial Results
  - Single block
  - Appropriate column headers

Channel Name	Trans Count	Recent Trans Count
ACH	9,972	127
Branch	20,317	202
Check	10,714	100
Electronic	9,037	90
Internet	4,496	37
Other	6,427	60
Paper	43,443	343
Wire	8,583	79

Data Subset

- Multiple SELECTs generated
  - Multiple SELECTs created as the measures come from separate contexts
  - First SELECT returns count against all transactions
  - Second SELECT returns count against only the most recent transactions
  - *JOIN_BY_SQL* rules still apply



Data Subset

Channel Name	Trans Count	Percentage	Recent Trans Count	Percentage
ACH	9,972	8.83%	127	12.24%
Branch	20,317	17.98%	202	19.46%
Check	10,714	9.48%	100	9.63%
Electronic	9,037	8.00%	90	8.67%
Internet	4,496	3.98%	37	3.56%
Other	6,427	5.69%	60	5.78%
Paper	43,443	38.45%	343	33.04%
Wire	8,583	7.60%	79	7.61%
	Percentage:	100.00%		100.00%

- Final formatted results
  - Use the calculation wizard to add percentages to each measure
- Can more be done?
  - Use techniques in Advanced Calculations section to move percentage calculations to the universe

Data Subset - Recap

- Derived table used to identify data subset to be analyzed
  - Derived table used as filtering mechanism
- Solution #1
  - Create filter object to force join to derived table
  - Multiple queries using filter object(s) when appropriate
  - Format report to properly label resulting figures
- Solution #2
  - Alias data table being segmented
  - Create same joins to alias as to the original table
  - Derived table only joined to the alias
  - Detect/create contexts
  - Measure object from derived table forces join to derived table

Multi-Pass - Recap

- Covered 5 different scenarios calling for multi-pass SQL
  - Sharing Dimensions Across Fact Tables
  - Calculations Which Require End Results
  - Blending Grains of Measurements
  - Need for Semi-Additive Measures
  - Analyzing a Subset of Data
- Often not restricted to one solution
  - Good news….
    - Business Objects does not force you into a solution
  - Bad news…
    - Business Objects does not force you into a solution
  - Decide where to place burden and to what degree
- Think in terms of traditionalist approach to solve new situations

Multi-Pass SQL

Questions?