Pentaho Metadata Formulas

Formulas have multiple uses in Pentaho Metadata. The first use of formulas in Pentaho Metadata is in the constraint definition of a Metadata Query, also known as MQL. A constraint function references business table columns and uses various comparison operators to determine which subset of data the business user is interested in.

The second use is in the definition of Physical Table Columns. In addition to Physical table columns mapping directly to a database table column, physical table columns defined in Pentaho Metadata may also be defined as a formula. This allows for combining of multiple columns into a single column, and also for doing more advanced aggregate calculations within aggregate table definitions.

The third use is in the definition of complex joins within business model relationships. This allows for multiple key joins as well as other logic when joining tables.

The fourth use is row level security. Under the covers, Pentaho Metadata uses JFreeReport's libFormula package for interpreting formulas. The goal is to support OpenFormula syntax in the Metadata environment. Formulas are first interpreted by libFormula, and then in the Metadata system are converted to native SQL depending on the type of database used.

First Use — MQL Constraints

Below is an example of an MQL Constraint formula:

```
OR([BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMERNAME] = "EuroCars";
    (([BT_CUSTOMERS.BC_CUSTOMERS_CREDITLIMIT] * 2) / 3 > 1000))
```

Note the `OR` function. This is a boolean function that has two parameters, separated by semicolons. These parameters are boolean expressions. The first boolean expression references a business column from the Metadata model. All references appear with brackets around them `[]`. This reference refers to the business table, and then to the business column. This boolean expression does some arithmetic and checks to see if the final value is larger than 1000. In the second expression, business column `BT_CUSTOMERS.BC_CUSTOMERS_CUSTOMERNAME` is compared to `EuroCars`. Notice that double quotes are used when referring to text; double quotes are required.

Second Use — Physical Table Column Formulas

Below is an example of a Physical Table Column Formula:

```
[QUANTITYORDERED] * [PRICEEACH]
```

The references in this example specifically refer to the database column, not derived the physical column definitions. All operators and functions may be used in the definition of the physical table column. For this
formula to be recognized, the isExact property of the physical table column must be set to true. The referenced physical column must be explicitly defined in the metadata model.

Note: In earlier versions of Pentaho Metadata Editor, (prior to the 3.5 release), aggregation functions had to be specified explicitly and the aggregation rule had to be selected. This is no longer necessary; the query that is generated will use the selected aggregation rule during execution.

**Multi-table expressions — Formulas can use any business column in the model**

Since the latest versions (after 2008/03/14) it is possible to define formulas that use business columns from anywhere in the business model. For example suppose there are two business tables:

- Orders (fact table), ID=BT_ORDER_FACT
- Product (dimension), ID=BT_PRODUCT

Suppose you want to calculate the turnover based on:

- Orders (fact table), ID=BT_ORDER_FACT
- Product (dimension), ID=BT_PRODUCT

  The number of products sold, from the Orders table, ID=BC_FACT_ORDER_NRPRODUCTS
  The price of the product, from the Product table, ID=BC_DIM_PRODUCT_PRICE

To get there, you must define a new business column, say in the Orders business table (although you could take Product too):

- Table: Orders (BT_ORDER_FACT)
  ID = BC_FACT_ORDER_TURNOVER
  Name = Turnover
  Formula = [BT_ORDER_FACT.BC_FACT_ORDER_NRPRODUCTS]
  Exact = Yes
  Aggregation Rule = SUM

The SQL generator is now going to replace the business columns by their respective SQL variants. As such, you must make sure that the business columns are resolving correctly. In this specific case, this means you want the two columns to be non-aggregated. If you now select the single business column BT_FACT_ORDER_TURNOVER, below is the SQL that is generated:

```
SELECT
  SUM( BT_ORDER_FACT.NRPRODUCTS * BT_PRODUCT.PRICE ) AS COL0
FROM
  FACT_ORDER BT_ORDER_FACT ,DIM_PRODUCT BT_PRODUCT
WHERE
  ( BT_ORDER_FACT.PRODUCT_TK = BT_PRODUCT.PRODUCT_TK )
```

Now, suppose you want to generate the multiplication of the two sums (different use-case). You define the formula as "[BT_ORDER_FACT.BC_FACT_ORDER_NRPRODUCTS] * [BT_PRODUCT.BC_DIM_PRODUCT_PRICE]" (without the SUM) and specify an aggregation for the two business columns in use. The generated SQL will be as follows:
SELECT SUM(BT_ORDER_FACT.NRPRODUCTS) * SUM(BT_PRODUCT.PRICE) AS COL0 FROM FACT_ORDER_BT_ORDER_FACT, DIM_PRODUCT_BT_PRODUCT WHERE (BT_ORDER_FACT.PRODUCT_TK = BT_PRODUCT.PRODUCT_TK)

It is possible to create two versions of the used business columns, one aggregated (exposed to the users) and one non-aggregated (hidden from the users) for example.

The SQL generator works recursively. That means that it is possible to create a formula that calculates 7% (taxes for example) of the turnover:

- **ID** = BC_FACT_ORDER_TURNOVER_TAXES
- **Name** = Turnover Taxes
- **Formula** = \([BT_ORDER_FACT.BC_FACT_ORDER_TURNOVER] \* 7 / 100\)
- **Exact** = Yes

If you add that column to the selection, you get one extra column as shown below:

\((\text{SUM}(BT_ORDER_FACT.NRPRODUCTS \* BT_PRODUCT.PRICE) \* 7 / 100)\) AS COL1

**Formula Syntax**

*Function syntax:*

`FUNCTION_NAME (PARAM; PARAM)

*Text (requires double quotes):*

"TEXT"

*Parenthesis are used for formula precedence:*

\((1 + 2) \* 3\)

**Metadata References**

*Business Column References:*

`[<BUSINESS_TABLE_ID>.<BUSINESS_COLUMN_ID>]`

*Physical Column References (only used in physical column formula definitions):*

`[<PHYSICAL_COLUMN_NAME>]`

- **Supported Functions**
- **Supported Operators**
- **Supported Aggregate Functions**