



Advanced MAQL Reference  
version 1.0

December 9, 2011

# Contents

<b>1</b>	<b>Basic elements of GDC MAQL</b>	<b>3</b>
1.1	Supported Data Types . . . . .	3
1.2	Data Types used the GoodData Platform . . . . .	3
1.3	GDC Objects . . . . .	4
1.3.1	Identifiers . . . . .	4
1.3.2	LDM Objects . . . . .	4
1.3.3	PDM Mapping . . . . .	5
<b>2</b>	<b>Operators</b>	<b>6</b>
<b>3</b>	<b>Functions and macros</b>	<b>7</b>
3.1	Functions FOR Previous, FOR PreviousPeriod . . . . .	8
3.1.1	FOR Next() and FOR NextPeriod() . . . . .	9
3.2	Time macros This, Previous . . . . .	10
<b>4</b>	<b>Expressions</b>	<b>11</b>
4.1	Logical expressions . . . . .	11
<b>5</b>	<b>Common MAQL DDL Clauses</b>	<b>12</b>
5.1	Create statement . . . . .	12
5.1.1	Create attributes . . . . .	12
5.1.2	Create dataset . . . . .	14
5.1.3	Create facts . . . . .	14
5.1.4	Create folder . . . . .	15
5.1.5	Create metric . . . . .	15
5.2	Alter statement . . . . .	15
5.2.1	Alter attribute . . . . .	15
5.2.2	Alter dataset . . . . .	16
5.2.3	Alter datatype . . . . .	17
5.2.4	Alter fact . . . . .	17
5.2.5	Alter folder . . . . .	18
5.3	Drop statement . . . . .	18
5.3.1	DROP . . . . .	18
5.3.2	DROP ALL IN . . . . .	18
5.4	Common example on MAQL DDL . . . . .	19
<b>6</b>	<b>Common MAQL DML Clauses</b>	<b>20</b>
<b>7</b>	<b>MAQL Queries and subqueries (Metrics)</b>	<b>22</b>
7.1	Metrics . . . . .	22
7.2	Aggregation . . . . .	23
7.2.1	COUNT . . . . .	24

7.3	Simple Arithmetic . . . . .	25
7.4	Filters . . . . .	25
7.4.1	Conditional filtering . . . . .	25
7.5	BY Clause . . . . .	26
7.5.1	BY ALL Clause . . . . .	27
7.5.2	BY project_attribute ALL IN ALL OTHER DIMENSIONS . . . . .	29
7.5.3	BY ALL IN ALL OTHER DIMENSIONS . . . . .	29
7.5.4	WITHOUT PARENT FILTER . . . . .	30

## Preface

This reference contains a complete description of the Multidimensional Query Language (MAQL) used to manage projects and metrics in the GoodData platform. MAQL (Multidimensional Query Language) is a simple yet powerful query language that provides an underpinning of GoodData's reporting capabilities. Its extension MAQL DDL (MAQL Data Definition Language) is used for building and adapting a data model. MAQL DQL (Multi-Dimensional Query Language or metric language) is our flexible language for describing metrics in GoodData.

## 1 Basic elements of GDC MAQL

### 1.1 Supported Data Types

Below is a table giving you a rough idea on best practices. If you fall outside the boundaries, be sure to get in touch with us – usually we can find a solution to work around problems. Specifically larger data sets (more rows) can be supported depending on how the data model is constructed and/or pre-aggregation can be used before data upload.

<b># of columns</b>	60 attributes, references & connection_points dataset
<b>Attribute size</b>	128 characters by default, extensible to 256 characters
<b>Fact size</b>	DECIMAL (12, 2) by default ( $-10^{10}..10^{10}$ , 2 decimal places) extensible to DECIMAL (15, 6) by default ( $-10^9..10^9$ , 6 decimal places)

### 1.2 Data Types used the GoodData Platform

By default the system automatically stores all facts as DECIMAL (12, 2) and all attributes and labels as 128-character strings. For performance reasons or to store other data types, you can redefine your column data type.

Data type	Description	Specific form
VARCHAR ( <i>n</i> )	$n \in (1..255)$	'YYYY-MM-DD'
DECIMAL ( <i>m,d</i> )	$m$ : $\min(-10^{15}), \max(10^{15})$ , $d$ : $\max = 6$	
INT	$\min(-2147483648), \max(2147483647)$	
BIGINT	$\min(-10^{15}), \max(10^{15})$	
DATE		
DOUBLE	discouraged	

The DATE datatype automatically maps with the GoodData-provided date dimension, if you have previously included it into the project.

## 1.3 GDC Objects

### 1.3.1 Identifiers

Identifiers in MAQL are denoted by curly brackets. Once an object is created, the identifier is persistent and cannot be changed. You can choose your own naming conventions for identifiers (in the examples below we often put identifiers into "namespace" by prefixing them with "folder.", "fact." etc.) Identifiers can contain alphanumeric characters, underscore and dot (e.g. [A-Za-z0-9\_\.]).

### 1.3.2 LDM Objects

LDM (Logical Data Model) describes the logical structure an organization's data in terms like datasets, attributes and facts. While the LDM is representing the logical structure, the physical structure is defined as PDM (Physical Data Model).

The LDM is necessary for creating your reports. It consists of attributes and facts that GoodData users add to their reports. Besides attributes and facts, LDM includes also datasets and folders. Simple example is shown on the schema bellow.

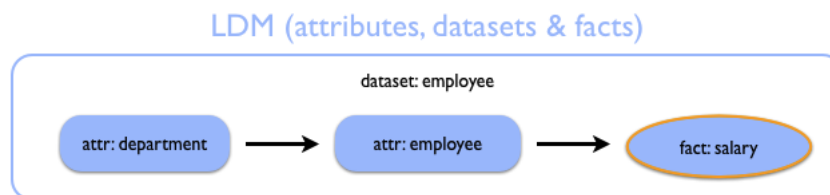


Figure 1: Example of simple LDM

As you can see, the LDM contains one fact and two attributes in a hierarchy (department and employee). As we have mentioned above, we'll use the MAQL DDL language to define the LDM.

**Attributes** An attribute is the unit that allows you to specify how to aggregate (or slice) your data. Examples would include: Assignee, City, Day, ID, Group etc. Attributes can optionally have additional labels. These are alternate string representation of the same semantic value. For example a person John Doe is the same person, regardless if they are visualized as "J. Doe", "Doe, John", "Johnny" etc. Or "Jan 10", "January 2010" and "01/2010".

**Attribute IDs** Every attribute has its own unique number. The number is invisible to end users. However, you can rely on the creation rules in regards to attribute IDs. In the specific case of date values, the ordinal attribute IDs are ordered by the attribute value. So, the Month attribute value of May 2007 is higher than the ID value of April 2007. Therefore, the relational operators (>, <, >=, <=, =) can be applied to the attribute IDs. So the filter `WHERE Month > April 2007` returns all months after April 2007.

**Facts** A fact is a data column containing computational data - e. g. prices, amounts. Generally, facts are measurable items of data attached to each record in the source data. Facts are always numbers, and cannot be broken down further. Examples include: Sales (a price for one transaction), Salary (the amount of money received), Cost (the amount paid for an item, Items Shipped (the number of items shipped in one delivery), etc. These are amounts reflected in each “transaction” or record.

**Datasets** A dataset is one joint source of data and comprises of attributes and facts.

**Folders** Top-level categories for organizing attributes or hierarchies of attributes. For example, you can create a Time folder containing the Year, Quarter, Month and Day attributes. Folders are used to visually organize facts or attributes and metrics for the user. Folders are types - e. g. they can only contain objects of one kind (hence the TYPE section). Folders are used to organize attributes and facts visually for users.

### 1.3.3 PDM Mapping

The PDM is used for the data storage and query. It is de-facto a DBMS schema (tables, columns, primary/foreign keys etc.).

Most analytical tools and platforms will force you to develop both LDM and PDM that are perfectly aligned. Unlike the average tools, the GoodData interface with user only via the LDM. The corresponding PDM is automatically generated from the LDM. The LDM model is created and modified via the MAQL DDL language statements, that are described in the section 5 of this reference.

The figure below outlines the PDM that has been automatically generated from the LDM above LDM (Fig. 1).

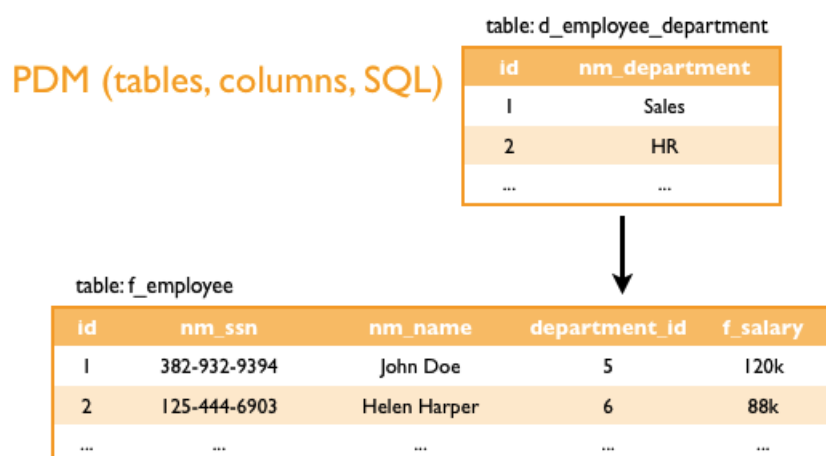


Figure 2: Generated PDM from LDM

As we mentioned before, the PDM is standard DBMS schema with tables, columns, primary and foreign keys. A LDM ATTRIBUTE is represented by a database table with primary key and couple text columns (one for each LABEL) in the PDM. A LDM FACT is mapped to a column in a PDM fact table. The figure below outlines the PDM that has been automatically generated from the LDM above.

Once we have the LDM, the PDM and DLI (Data Loading Interface) will be generated automatically via the SYNCHRONIZE MAQL DDL command. The DLI is used for loading data to the PDM.

The GoodData platform loads data in self-describing packages. The data package is a ZIP archive that contains the data (delimited file) and a manifest that describes how the data map to the project's LDM and PDM. The figure below shows the data file only. As you can see this is de-normalized (flattened d\_employee\_department and f\_employee PDM tables) version of the PDM.

Data (columns, rows, CSV)

ssn	name	department	salary
382-932-9394	John Doe	Sales	120k
125-444-6903	Helen Harper	HR	88k
...	...	...	...

Figure 3: De-normalized version of PDM

## 2 Operators

Logical and special operators

**= (equals)** This filter allows you to compute the metric from a specific attribute value.

**syntax**      `SELECT project_metric`  
                  `WHERE {project_attribute} = {project_attribute_value}`

**example**      `SELECT Salary WHERE Year = 2007;`

**example**      `SELECT Revenues WHERE Month = {This};`

**example**      `SELECT Expenses WHERE Quarter = {Previous};`

**example**      `SELECT Expenses WHERE Quarter = {This} - 2;`

**example**      `SELECT Amount WHERE Date ordered = Date Shipped;`

**example**      `SELECT Employee WHERE Payment = 10000;`

**<> (does not equal)** This filter allows you to compute the metric by excluding a particular attribute value.

**syntax**      `SELECT project_metric;`  
                  `WHERE {project_attribute} <> {project_attribute_value};`

*example*        `SELECT Revenues WHERE Year <> 2006;`  
*example*        `SELECT Amount WHERE Date Ordered <> Date Shipped;`

**IN**    This filter allows you to compute the metric from multiple attribute values.

*syntax*        `SELECT project_metric`  
                  `WHERE {project_attribute}`  
                  `<> (proj_attr_val1, proj_attr_val2);`  
*example*        `SELECT Profit WHERE Year IN (2006, 2007);`

**NOT IN**   This filter allows you to compute the metric by excluding particular attribute values.

*syntax*        `SELECT project_metric`  
                  `WHERE {project_attribute}`  
                  `NOT IN (project_attribute_value1, project_attribute_value2);`  
*example*        `SELECT Profit WHERE Year NOT IN (2006, 2007);`  
*example*        `SELECT Sales WHERE Quarter NOT IN ({This}, {Previous});`

**BETWEEN**   This filter allows you to compute the metric by using attribute values from a specified range (including endpoints).

*syntax*        `SELECT project_metric`  
                  `WHERE {project_attribute}`  
                  `BETWEEN project_attribute_value1`  
                  `AND project_attribute_value2;`  
*example*        `SELECT Profit WHERE Year BETWEEN 2005 AND 2008;`  
*example*        `SELECT Sales WHERE Quarter BETWEEN {This} - 5 AND {This});`

**NOT BETWEEN**   This filter allows you to compute the metric by using attribute values outside of a specified range (excluding endpoints).

*syntax*        `SELECT project_metric`  
                  `WHERE {project_attribute}`  
                  `NOT BETWEEN project_attribute_value1`  
                  `AND project_attribute_value2;`  
*example*        `SELECT Profit WHERE Year NOT BETWEEN 2005 AND 2008;`

You can use these clauses to create fixed numbers which can be used in other computations to create complex metrics. Fixing numbers is achieved by locking the aggregation level to an attribute in one or more dimensions.

### 3 Functions and macros

All the available functions and macros are handling time comparisons and global operations with time dimensions.



### 3.1 Functions FOR Previous, FOR PreviousPeriod

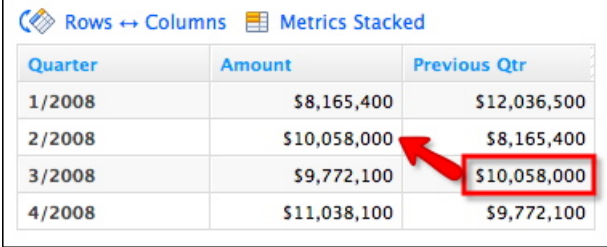
Two special functions in MAQL allow you to create metrics showing a time over time comparison (e. g., month over month, or quarter over quarter):

FOR Previous() and  
FOR PreviousPeriod().

**FOR Previous ()** This statement allows you to compute a metric for a previous, fixed time period.

*example* SELECT Revenues FOR Previous(Quarter);

This example result is shown on the figure below. As you can see it provides the value from previous period, in this example is the previous quarter.

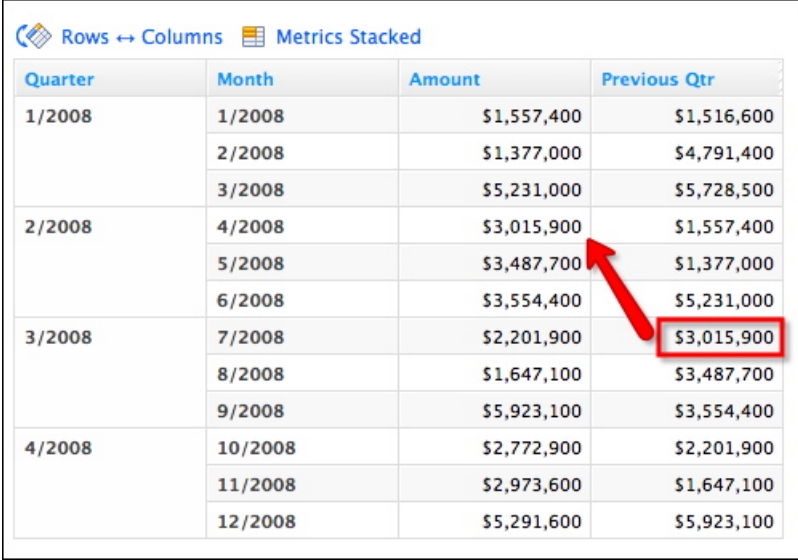


Quarter	Amount	Previous Qtr
1/2008	\$8,165,400	\$12,036,500
2/2008	\$10,058,000	\$8,165,400
3/2008	\$9,772,100	\$10,058,000
4/2008	\$11,038,100	\$9,772,100

Figure 4: SELECT Revenues FOR Previous(Quarter)

*example* SELECT Revenues FOR Previous(Quarter,2);

This example result is shown on the figure 6. In this second example, the extra "2" specifies to skip back two quarters, rather than simply last quarter.



Quarter	Month	Amount	Previous Qtr
1/2008	1/2008	\$1,557,400	\$1,516,600
	2/2008	\$1,377,000	\$4,791,400
	3/2008	\$5,231,000	\$5,728,500
2/2008	4/2008	\$3,015,900	\$1,557,400
	5/2008	\$3,487,700	\$1,377,000
	6/2008	\$3,554,400	\$5,231,000
3/2008	7/2008	\$2,201,900	\$3,015,900
	8/2008	\$1,647,100	\$3,487,700
	9/2008	\$5,923,100	\$3,554,400
4/2008	10/2008	\$2,772,900	\$2,201,900
	11/2008	\$2,973,600	\$1,647,100
	12/2008	\$5,291,600	\$5,923,100

Figure 5: SELECT Revenues FOR Previous(Quarter,2)

**FOR PreviousPeriod()** This statement allows you to compute a metric for a contextual previous period.

*example* `SELECT Payment FOR PreviousPeriod(Quarter);`

Both of the example's results are shown on the figure below. As you can see it provides the value from previous period, in this example is the previous quarter. In this second example, the extra "2" specifies to skip back two periods, rather than simply the previous period.

*example* `SELECT Payment FOR PreviousPeriod(Quarter, 2);`

Rows ↔ Columns Metrics Stacked

Quarter	Payment	FOR PreviousPeriod (Q)	FOR PreviousPeriod (Q, 2)
Q3/2006	669,315.00	669,250.00	669,392.00
Q4/2006	669,231.00	669,315.00	669,250.00
Q1/2007	1,143,775.00	669,231.00	669,315.00
Q2/2007	1,143,571.00	1,143,775.00	669,231.00
Q3/2007	1,143,619.00	1,143,571.00	1,143,775.00
Q4/2007	1,143,692.00	1,143,619.00	1,143,571.00

Figure 6: Payment FOR Previous(Quarter) and FOR Previous(Quarter,2)

### The Difference Between FOR Previous() and FOR PreviousPeriod()

The `FOR Previous()` statement will always return the previous aggregated result for the time period specified within the parenthesis. For example, entering `FOR Previous(Year)` will always return the metric aggregated for the previous year, based on the values in the report. For example, with current sales of April 2010, `FOR Previous(Year)` will display April 2009. Likewise, `FOR Previous(Quarter)` will display January 2010 (e. g., since April is M1/Q2, the "Previous()" function would look for M1/Q1 – the previous quarter – or January 2010).

On the other hand, `FOR PreviousPeriod()` will adjust the aggregation level of the returned metric based on the attributes contained within the report. For example, if `Quarter` is displayed in the report, the Previous Period would also be preceding Quarter. If this changes to `Year`, then the Previous Period would also change to the preceding Year. However, it is still necessary to specify an attribute within parenthesis so that the metric knows from which date dimension to select the previous period.

#### 3.1.1 FOR Next() and FOR NextPeriod()

Alternatively, it is also possible to compare against future dates using two other special MAQL functions:

`FOR Next()` and  
`FOR NextPeriod()`.

Both of these constructs function in the same way as `FOR Previous()` and `FOR PreviousPeriod()` above.

Rows → Columns Metrics Stacked

Quarter	Month	Payment	FOR Previous	FOR PreviousPeriod
Q1/2006	Jan 2006	223,195.00		
	Feb 2006	223,148.00		223,195.00
	Mar 2006	223,049.00		223,148.00
Q2/2006	Apr 2006	223,032.00	223,195.00	223,049.00
	May 2006	223,135.00	223,148.00	223,032.00
	Jun 2006	223,083.00	223,049.00	223,135.00
Q3/2006	Jul 2006	223,108.00	223,032.00	223,083.00
	Aug 2006	223,075.00	223,135.00	223,108.00
	Sep 2006	223,132.00	223,083.00	223,075.00
Q4/2006	Oct 2006	223,063.00	223,108.00	223,132.00
	Nov 2006	223,087.00	223,075.00	223,063.00
	Dec 2006	223,081.00	223,132.00	223,087.00
Q1/2007	Jan 2007	381,244.00	223,063.00	223,081.00
	Feb 2007	381,350.00	223,087.00	381,244.00
	Mar 2007	381,181.00	223,081.00	381,350.00
Q2/2007	Apr 2007	381,269.00	381,244.00	381,181.00

Figure 7: The difference between FOR Previous() and FOR PreviousPeriod()

**FOR Next()** This statement allows you to compute a metric for a future, fixed time period.

*example* `SELECT Revenues FOR Next(Quarter);`

Again as in FOR Previous() function, if we add "2" as second function parameter, it specifies to jump ahead two quarters, rather than simply next quarter.

*example* `SELECT Revenues FOR Next(Quarter, 2);`

**FOR NextPeriod()** This statement allows you to compute a metric for a contextual next period.

*example* `SELECT Revenues FOR NextPeriod(Quarter);`

Again, as in FOR PreviousPeriod() function, if we add "2" as second function parameter, it specifies to jump ahead two quarters, rather than simply next quarter.

*example* `SELECT Revenues FOR NextPeriod(Quarter, 2);`

### 3.2 Time macros This, Previous

MAQL provides special constructs for specifying days in your reports. We define them as time macros and allow you to use create reports for "Today" or "Yesterday", and can also help create Year-to-Date reports.

The time macros can be used as attribute values for attributes contained in the GoodData Date hierarchy. These commands take inherit their value based on their context within the metric.

- In this example, {This} would mean "This Date," or "Today".

*example*      `SELECT Payment WHERE Date = {This};`

- In this example, {This} - 1 would mean "This Date" minus one, or "Yesterday".

*example*      `Example: SELECT Payment WHERE Date = {This} - 1;`

- In this example, {This} would mean "This Month".

*example*      `SELECT Payment WHERE Month = {This};`

- In this example, {Previous} would mean "Previous Quarter".

*example*      `SELECT Expenses WHERE Quarter = {Previous};`

- In this example, {This} would mean "This Year".

*example*      `SELECT Payment WHERE Year = {This};`

- In this example, the first {This} would mean "This Year" while the second {This} would refer to "This Day of Year" (e. g., Day 1 for Jan. 1, Day 2 for Jan. 2, etc).

*example*      `SELECT Payment WHERE Year = {This}  
AND Day of Year = {This};`

- In this example, {Previous} would mean "Previous (Last) Quarter" and {This} would refer to "Today as a numeric Day of the Quarter" (e. g., Day 62 of the quarter).

*example*      `SELECT # of Employees WHERE Quarter = {Previous}  
AND Day of Quarter = {This};`

## 4 Expressions

### 4.1 Logical expressions

Allow you to combine multiple filters.

**NOT** If you specify a filter and precede it with NOT, then everything that is specified by the filter will be excluded from the report computation.

*example*      `SELECT Revenues WHERE NOT (Year = 2006 AND Month = 5);`

**AND** If you combine filters using AND, then both filters are applied when computing the metric.

*example*      `SELECT Revenues WHERE Year = 2006 AND Month = 5;`

**OR** If you combine filters using OR, then the result for each filter is computed individually and the results are combined.

*example*      `SELECT Revenues WHERE Year = 2006 OR Year = 2005;`

## 5 Common MAQL DDL Clauses

All the DDL statements are used when you creating your project with the CL tool. But you can use them also on the project's gray pages. The command line for running the DDL statements is located on the following URL

[https://secure.gooddata.com/gdc/md/<project\\_id>/ldm/manage](https://secure.gooddata.com/gdc/md/<project_id>/ldm/manage)

### 5.1 Create statement

The MAQL DDL is used to create the LDM. Such LDM is shown on the Fig. 8. This model was simplified for better understanding, only some of the date related attributes are displayed.

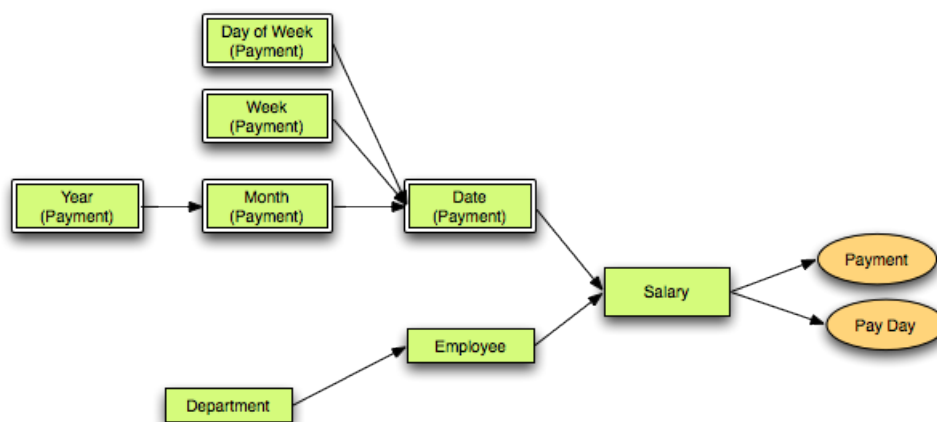


Figure 8: HR demo project LDM

#### 5.1.1 Create attributes

An attribute is represented by a database table with a primary key and couple text columns. Attributes are categories that are used for slicing and dicing the facts (countable numbers). An attribute can be empty or with keys and labels. Creating attributes with primary and foreign keys allows us to construct the LDM as on Fig. 8. If we create attribute Employee with this statement:

**example**

```
CREATE ATTRIBUTE {attr.employee.employee}
VISUAL(TITLE "Employee", FOLDER {dim.employee})
AS KEYS {f_employee.id} FULLSET;
```

This statement creates the LDM attribute named Employee. The LDM attribute maps to a PDM table that has the primary key {f\_employee.id} (FULLSET). The key word FULLSET marks the LDM attribute which is defined in the PDM as primary key. We can also add the foreign key, but we will show it later. The attribute is defined as a PDM table. The attribute itself is the auto-generated ID f\_employee.id.

Now, we will create the Department attribute and we'll create the foreign key to the f\_employee table:

**example**      `CREATE ATTRIBUTE {attr.employee.department}  
                  VISUAL(TITLE "Department")  
                  AS KEYS {d_employee_department.id} FULLSET,  
                  {f_employee.department_id};`

Note that the MAQL DDL statement contains two keys. The first one d\_employee\_department.id is the primary key and the second one f\_employee.department\_id is the foreign key that ties the d\_employee\_department table to the f\_employee table.

**Create attribute full syntax**    The full syntax of the attribute creation command is specified as

**syntax**      `CREATE ATTRIBUTE {identifier}  
                  [VISUAL({properties}) [HYPERLINK]  
                  AS {identifier} KEYS {object} KEY_TYPE]  
                  WITH LABELS {label_list} ORDER {identifier}  
                  [ASC | DESC];`

Create empty attribute.

**example**      `CREATE ATTRIBUTE {attr.employee.name}  
                  VISUAL(TITLE "Name",  
                  FOLDER {folder.employee}) AS;`

Create attribute with primary key.

**example**      `CREATE ATTRIBUTE {attr.employee.employee}  
                  AS {f_employee.id} FULLSET;`

Create attribute with label.

**example**      `CREATE ATTRIBUTE {attr.employee.name}  
                  AS WITH LABELS {label.employee.name}  
                  VISUAL(TITLE "Employee's name");`

For the rest of the example in this section we will work with opportunities. First, we create attribute with title and mapping for primary key and with label *Opp. category* mapped to column in the same table.

**example**      `CREATE ATTRIBUTE {attr.opportunity.category}  
                  VISUAL(TITLE "Category of opportunity",  
                  FOLDER {folder.opportunity})  
                  AS KEYS {tab_cat.col_id} FULLSET, {tab_opp.col_cat}  
                  WITH LABELS {label.opportunity.category}  
                  VISUAL(TITLE "Opp. category") AS {tab_cat.col_lbl};`

Create attribute with order (sorting options on attributes).

**example**      `CREATE ATTRIBUTE {attr.opportunity.category}  
                  AS LABELS {label1} VISUAL(TITLE "Opp. category")  
                  ORDER {label1} ASC;`

**example**      `CREATE ATTRIBUTE {attr.employee.employee}  
                  {attr.employee.firstname} VISUAL(TITLE "First Name")`

```
ORDER {attr.employee.firstname} DESC;
```

To allow attribute to be shown as hyperlink.

**example**      `CREATE ATTRIBUTE {attr.employee.web}  
AS LABELS {attr.employee.web}  
VISUAL(TITLE "First Name") HYPERLINK;`

If we'll create the foreign key to the `f_employee` table, note that the first MAQL DDL statement contains two keys. The first one `d_employee_department.id` is the primary key and the second one `f_employee.department_id` is the foreign key that ties the `d_employee_department` table to the `f_employee` table.

**example**      `CREATE ATTRIBUTE {attr.employee.department}  
VISUAL(TITLE "Department")  
AS KEYS {d_employee_department.id}  
FULLSET, {f_employee.department_id};`

### 5.1.2 Create dataset

Dataset groups all following logical model elements together.

**syntax**      `CREATE DATASET {identifier}  
[VISUAL({properties})];`

To create empty dataset. Create dataset with label *Employee*.

**example**      `CREATE DATASET {dataset.employee}  
VISUAL(TITLE "Employee");`

### 5.1.3 Create facts

Its purpose is to create fact with or without expression. Again, if we use our LDM from fig. 8, we can add to our employees salary fact:

**example**      `CREATE FACT {fact.employee.salary}  
VISUAL(TITLE "Salary")  
AS {f_employee.f_salary};`

This statement creates new Salary fact (LDM) and the corresponding `{f_salary}` column in the `f_employee` table (PDM). Remember that the fact is represented as a database column.

After successful creation, the fact must be added to the employee dataset:

**example**      `ALTER DATASET {dataset.employee}  
ADD {fact.employee.salary};`

All PDM tables, columns, keys etc. are automatically generated after calling the SYNCHRONIZE MAQL DDL command that is usually the last command of a MAQL DDL script.

**syntax**      `CREATE FACT {identifier}  
[VISUAL({properties})];  
AS [{object}];`

Create empty fact.

**example**      `CREATE FACT {fact.opportunity.sales}  
VISUAL( TITLE "Sales" ) AS;`

Create fact with expression.

**example**      `CREATE FACT {fact.opportunity.sales}  
                  VISUAL( TITLE "Sales" ) AS {tab_opp.col_sales};`

### 5.1.4 Create folder

Create the folders that group attributes and facts. Folders keep our dataset readable and organized.

**syntax**      `CREATE FOLDER {identifier}  
                  [VISUAL({properties})]  
                  TYPE {type_identificator};`

Create folder of attributes.

**syntax**      `CREATE FOLDER {dim.employee}  
                  VISUAL(TITLE "Employee Title") TYPE ATTRIBUTE;`

Create folder of facts.

**example**      `CREATE FOLDER {ffld.salary}  
                  VISUAL(TITLE "Salary") TYPE FACT;`

Create folder of metrics.

**example**      `CREATE FOLDER {ffld.salary}  
                  VISUAL(TITLE "Salary", DESCRIPTION "salary description")  
                  TYPE METRIC;`

### 5.1.5 Create metric

Not supported in MAQL DDL. The recommended way to create metrics is inside the GoodData platform by using metric editor (see section 7)

## 5.2 Alter statement

The ALTER statements allows you to make changes in the LDM model.

### 5.2.1 Alter attribute

To change attribute property as visual as key or label. The attribute has to exists.

**syntax**      `ALTER ATTRIBUTE {identifier}  
                  [VISUAL({properties})]  
                  [[DEFAULT LABEL {identifier}] | [ DROP [KEYS |  
LABELS] [ADD [KEYS [PRIMARY | FULLSET] |  
LABELS] | ALTER LABEL {identifier} [HYPERLINK]]  
                  {object} ORDER BY {identifier} [ DESC | ASC ];`

Add three keys to attribute (first is primary).

**example**      `ALTER ATTRIBUTE {dataset.salary}  
                  ADD KEYS {dt.salary.payday_id} PRIMARY,`



```
{dt.salary.payday}, {dt.salary.employee};
```

Add label with identifier *label.employee.employee* and definition *attr.employee.employee*.

**example**

```
ALTER ATTRIBUTE {attr.employee.employee}
ADD LABELS {label.employee.employee}
AS {f_employee.nm_employee};
```

Drop key which identify the employee name from employee's attribute.

**example**

```
ALTER ATTRIBUTE {attr.employee.employee}
DROP KEYS {d_employee.id};
```

Drop all keys from attribute.

**example**

```
ALTER ATTRIBUTE {attr.employee.employee}
DROP KEYS ALL;
```

To change attribute title and folder.

**example**

```
ALTER ATTRIBUTE {attr.employee.employee}
VISUAL(FOLDER {dim.employee}, TITLE "Employee");
```

To change title of two attribute's labels.

**example**

```
ALTER ATTRIBUTE {attr.company}
ALTER LABELS {label.company.company.fullname}
VISUAL(TITLE "GoodData Corporation"),
{label.company.company.shortname} VISUAL(TITLE "GDC");
```

To change ORDER.

**example**

```
ALTER ATTRIBUTE {attr.employee.employee}
ORDER BY {label.employee.employee.firtsname} ASC;
```

To change DEFAULT.

**example**

```
ALTER ATTRIBUTE {attr.employee.employee}
DEFAULT LABEL {label.employee.employee};
```

To allow attribute to be shown as a hyperlink.

**example**

```
ALTER ATTRIBUTE {attr.employee.web}
DEFAULT ALTER LABELS {attr.employee.web} HYPERLINK;
```

## 5.2.2 Alter dataset

To add or drop objects (attributes, facts) to/from dataset. Or only change visual property of dataset. One attribute or fact should always belong to exactly one dataset (no more, no less). Otherwise the validation of the project will fail.

**syntax**

```
ALTER DATASET {identifier}
[VISUAL({properties})]
[OPERATION {objects}];
```

Add fact or attribute to dataset.

**example**

```
ALTER DATASET {dataset.quotes}
ADD {probability};
```

Add a new attribute into the dataset.

**example**

```
ALTER DATASET {dataset.employee}
ADD {attr.employee.employee};
```

If we'll create the foreign key to the `f_employee` table, note that the first MAQL DDL statement contains two keys. The first one `d_employee_department.id` is the primary key and the second one `f_employee.department_id` is the foreign key that ties the `d_employee_department` table to the `f_employee` table.

Drop attribute from dataset.

```
example      ALTER DATASET {dataset.quotes}
               DROP {Cutomer};
```

Change title of dataset.

```
example      ALTER DATASET {dataset.quotes}
               VISUAL(TITLE "Internal Quotes Data");
```

### 5.2.3 Alter datatype

This statement allows change data type of DataLoadingColumn (DLC). DLC is a column that is exposed to user API and represents column in user's CSV file. Data column identifiers is the only exception, where identifier name has to follow a certain structure. These identifiers reference specific data columns in the data files that you upload through the upload API. The identifiers must take form of `file.column`. The `file` part corresponds to the data file, while `column` corresponds to a specific column. All columns that in 3NF share a common file must also share the same prefix in their identifiers. A LDM is defined through MAQL statements, where the `table.column` is specified for the mapping on PDM. Within that MAQL statement the `table.column` identifier is handled as DataLoadingColumn (DLC) and concurrently PDM objects table and column are established.

By default the system automatically stores all facts as `DECIMAL(12,2)` and all attributes and labels as 128-character strings. For performance reasons or to store other data types, you can redefine your column data type (supported data types see section 1.2).

```
syntax      ALTER DATATYPE {object.identifier}
               {supported_datatype};
```

Changing data type at three columns of the employee table.

```
example      ALTER DATATYPE {dataset.employee.empl_id} INT,
               {dataset.employee.empl_name} CHAR(100),
               {dataset.employee.empl_fact} DECIMAL(10,2);
```

### 5.2.4 Alter fact

When we need to change visual or expression property.

```
syntax      ALTER FACT {fact.identifier}
               [VISUAL] |
               [VISUAL ADD | DROP object.identifier];
```

Change title of fact.

```
example      ALTER FACT {fact.payment}
               VISUAL(TITLE "Payment");
```

Add expression.

**example**      ALTER FACT {fact.payment}  
                  ADD {fact.quotes.currency};

Drop expression.

**example**      ALTER FACT {fact.payment}  
                  DROP {fact.payment.currency};

### 5.2.5 Alter folder

At the moment only the visual change is supported.

**syntax**      ALTER FOLDER {folder.identifier}  
                  VISUAL(TITLE <string>);

Change title of folder.

**example**      ALTER FOLDER {dim.quotes}  
                  VISUAL(TITLE "New Quotes");

## 5.3 Drop statement

### 5.3.1 DROP

There are two types of Drop statements. Simple drop, drops only objects with no dependants. Especially objects that hasn't any edge or are not connected to other objects. Cascade drop, removes everything what has affinity to dropping object.

**syntax**      DROP [IF EXISTS] {identifier}  
                  [CASCADE];

Drop attribute.

**example**      DROP IF EXISTS {attr.opportunity.category};

Drop attributes.

**example**      DROP IF EXISTS {attr.opportunity.category},  
                  {attr.opportunity.name};

Drop attributes cascade. This drop all metrics, reports etc.

**example**      DROP IF EXISTS {attr.opportunity.category},  
                  {attr.opportunity.name} CASCADE;

### 5.3.2 DROP ALL IN

To drop objects in a folder or dataset and also folder (dataset) itself. Again, this statement invokes simple or cascade drop to all objects in folder and dataset.

**syntax**      DROP [ALL IN] [IF EXISTS] {identifier}  
                  [CASCADE];

Drop attributes in dataset.

**example**      DROP ALL IN IF EXISTS {dataset.opportunity};

Drop all metrics in folder cascade.

**example**      DROP ALL IN IF EXISTS {folder.opportunity} CASCADE;

## 5.4 Common example on MAQL DDL

This example describes a typical situation when we create a new attribute and which operations need to be done to do it correctly. We start with the definition of the attribute.

*example*      `CREATE ATTRIBUTE {attr.employee.employee}  
                  VISUAL(TITLE "Employee", FOLDER {dim.employee})  
                  AS KEYS {f_employee.id} FULLSET;`

And then we add the newly defined attribute into the dataset

*example*      `ALTER DATASET {dataset.employee}  
                  ADD {attr.employee.employee};`

### *Statement 1*

```
CREATE ATTRIBUTE {attr.employee.department}
VISUAL(TITLE "Department")
AS KEYS {d_employee_department.id} FULLSET,
{f_employee.department_id};
```

### *Statement 2*

```
ALTER ATTRIBUTE {attr.employee.department}
ADD LABELS {label.employee.department.name}
VISUAL(TITLE "Department")
AS {d_employee_department.nm_department};
```

### *Statement 3*

```
ALTER DATASET {dataset.employee}
ADD {attr.employee.department};
```

And finally add labels to the new attributes.

```
ALTER ATTRIBUTE {attr.employee.employee}
ADD LABELS {label.employee.employee.firstname}
VISUAL(TITLE "First Name") AS {f_employee.nm_firstname};
```

```
ALTER ATTRIBUTE {attr.employee.employee}
DEFAULT LABEL {label.employee.employee.firstname};
```

```
ALTER ATTRIBUTE {attr.employee.employee}
ADD LABELS {label.employee.employee.lastname}
VISUAL(TITLE "Last Name") AS {f_employee.nm_lastname};
```

```
ALTER ATTRIBUTE {attr.employee.employee}
ADD LABELS {label.employee.employee}
VISUAL(TITLE "Employee") AS {f_employee.nm_employee};
```

In the second line of the script, we again added the label to the Department attribute. The label is defined as `nm_department` column in the `{d_employee_department}` table. Finally, we add the Department attribute to the Employee dataset.

## 6 Common MAQL DML Clauses

MAQL DML has a nickname which is Destructive MAQL. It should be used very carefully, because if you delete the row of an attribute, You can't roll-back it. We have a several use-cases which explain how powerful the delete command is. The most important issue is the referential integrity corruption. When deleting rows from an attribute table, you should keep in mind, which of the effected rows are directly connected to fact through the facts of table (in our HR example LDM is the facts of table Salary).

**syntax**      `DELETE FROM {attribute.factsof}`  
                 `WHERE <condition>`

GoodData doesn't support running the DML statements from the UI, you need to navigate you on the gray pages. The command line for running the DML statements is located on the following URL

`https://secure.gooddata.com/gdc/md/<project_id>/dml/`

We also extremely recommend you to validate your model before and after each DML statement to be sure no referential integrity errors are produced after your deletion. The following example deletes or destroys all the rows in the Salary table and represents the elementary MAQL delete statement.

**example**      `DELETE FROM {attr.salary.salary}`

This is very unusual example because normally, we don't like to delete all of the rows. In the syntax of the destructive delete are couple of different language details that as in DDL or later DQL. Let's explain the `FROM` part of the statement. By using the `FROM` clause you specify the objects from which you are deleting rows. E.g. in the previous example, the attribute `{attr.employee.id}` is the `CONNECTION_POINT` of the Salary dataset. The dataset also contains the `{attr.employee.lastname}` attribute. The `{attr.employee.lastname}` has the `{label.employee.lastname}` label. With next example you will delete only those employees who first name is Ed or Lin.

**example**      `DELETE FROM {attr.employee.employee.firstname}`  
                 `WHERE {label.employee.employee.firstname}`  
                 `IN ("Ed", "Lin");`

The dataset's records are preserved. This statement most probably breaks the referential integrity of the project as the records with the Ed and Lin values have no longer reference any value in the `{attr.employee.employee.firstname}` attribute.

If we want to delete all rows where the name of the person (defined by label) is Ed or Lin, we use following statement. This statement deletes all records with the attribute `{attr.employee.employee.firstname}` equal to Ed or Lin.

**example**      `DELETE FROM {attr.employee.employee.id}`  
                 `WHERE {label.employee.employee.firstname}`

```
IN ("Ed", "Lin");
```

Next part to explain is the condition construct. The conditions supports most of the rational (>, <, =) and logical (AND, OR, NOT) operators. The left side of the condition can be defined from the fact element or the label defined on the attribute. The right side is formulated by the value which is used as the pattern.

The following example shows one of the elementary situations where we delete all the rows from Salary facts of table, where the value of payment is lower or equal to 10000. For now the lower or equal sign ("=<") is not supported in the DML conditioning.

```
example    DELETE FROM {attr.salary.salary}
            WHERE {fact.salary.payment} < "10000"
            OR {fact.salary.payment} = "10000";
```

Another situation where the delete statement is very handy is situation when you need to delete all the rows, where the fact has a zero value. This example is shown on the following example:

```
example    DELETE FROM {attr.salary.salary}
            WHERE {fact.payment} = 0;
```

Think about a situation when you have some rows in facts tables where are values of zero. In this situation, the facts Volume or Close price having zero value are affecting the resulting report. Then, we'll need to delete all of the rows which have zero Volume or the Close Price is zero.

```
example    DELETE FROM {attr.quotes.quotes}
            WHERE {fact.quotes.volume} = 0
            OR {fact.quotes.close_price} = 0;
```

Another interesting use-case is deleting of the old data attribute values. E. g. when you need to delete all rows where the date is older than 1<sup>st</sup> January of 1999.

```
example    DELETE FROM {attr.salary.salary}
            WHERE {label.salary.dt_payday} < "2006-02-01";
```

Note that if you, by mistake, try to use anything other than a label in the condition of our delete statement, then an error will occur.

You are not allowed to delete any rows from the date dimension provided by Good Data. Delete all rows where for the 2010 (all the dates are between 1<sup>st</sup> Jan and 31<sup>st</sup> Dec 2010).

```
example    DELETE FROM {attr.salary.salary}
            WHERE {label.salary.dt_payday}
            BETWEEN "2010-01-01" AND "2010-12-31";
```

Globally if you want to delete any attribute row you have to start deleting on the referencing attribute values of the attribute we like to delete to not corrupt the referential integrity of the whole model. E. g. if you want to delete department with ID 1, you have to first delete all the employees who work in that department.

After you finish all the delete operations, it's important to validate Your model again. The validation shows you the actual status and checks the referential integrity of the whole model. The validation is located in the gray pages of your projects which you can find in the following URL:

[https://secure.gooddata.com/gdc/md/<project\\_id>/validate/](https://secure.gooddata.com/gdc/md/<project_id>/validate/)

## 7 MAQL Queries and subqueries (Metrics)

Metrics are specific part of MAQL. Previously described MAQL DDL and DML are running as part of the CL tool, or we can execute the MAQL DDL and DML from the projects' gray pages. The only part of MAQL which is fully integrated in the GoodData UI is the MAQL DQL, or metrics statements in other words.

We define the metric in the Manage section of a project. When you create a new metric, you will come across the same menu as in the figure below:

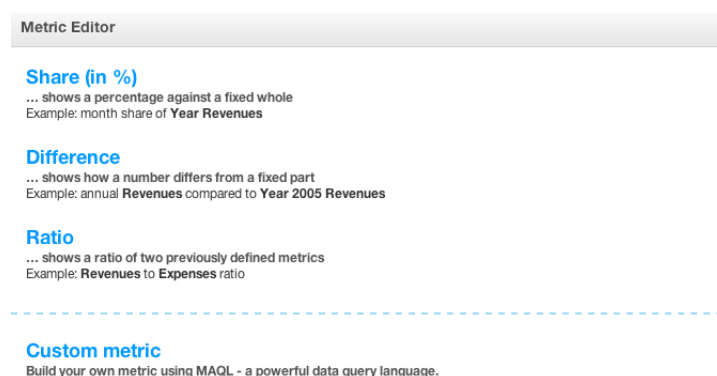


Figure 9: User menu for metric creation

The first 3 options (Share, Difference, Ratio) represent complex metric statements prepared for typical use cases. The fourth option allows you to build your own metric using MAQL DQL.

Each metric is defined by MAQL (MAQL DQL) and consists of 2 main parts – aggregation and filters (optional).

### 7.1 Metrics

Every report is defined by metrics, which determines WHAT will be computed in the report. Since metrics are numbers, they can be used to calculate Sums, Averages, Minimum Values, Maximum Values and so on.

The metrics are divided into Global Metrics and Report-Specific Metrics with no difference on the functionality level.

- *Global Metric* a metric that has been created and is available for all reports,
- *Report Specific Metric* a metric that has been user-defined for the current report. A Report Specific Metric can be added to the list of global metrics after creation.

The only difference is that Global Metrics are available to be used in other reports, while Report Specific Metrics are available only in the current report. Users wanting to use a similar metric will need to create it again in the other report. If you deletes a global metric that is used in a report, the metric is converted to the local metric in the report.

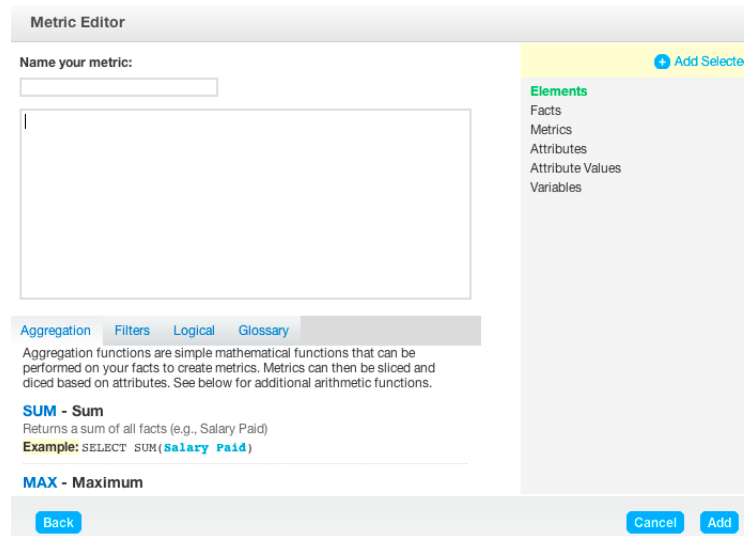


Figure 10: Custom metric editor in GoodData Platform

Following metric definitions examples have semantics used in the custom metric editor in the GoodData platform (as shown on fig. 10). If you like to validate your MAQL DQL statement in the gray pages of your projects, you need to specify the attributes and facts with the brackets as in previous sections of this reference.

## 7.2 Aggregation

GoodData reports are multi-dimensional pivot tables and each metric must define an aggregation.

**Aggregation** Data aggregation is the process in which information is gathered and expressed in a summary form, or the procedure where data values are grouped with the goal of managing each data unit as a single entity. Sales is a key metric which can be aggregated by Day, Month, Quarter, etc. For aggregation, we do use aggregation function similar to SQL functions but much more intuitive way.

You can specify the level at which the data are aggregated. In other words, you can take a look at the data aggregated by US state, county, city or ZIP levels. You can aggregate by multiple attributes at the same time. For example, you can define your metric to show yearly Payments by Departments. The default aggregation is the whole data set (e.g. one number for the entire country and all years).

**syntax** `SELECT FUNCTION(project_fact);`

**example** `SELECT AVG(Payment);`

Metrics can then be sliced and diced based on attributes. The following description includes all supported arithmetic functions.



**SUM** Returns a sum of all facts (e.g., Sales)

*syntax*      `SELECT SUM(project_fact) ;`

*example*      `SELECT SUM(Payment) ;`

**MAX** Returns the maximum value of all facts in the set (e.g., Sales)

*syntax*      `SELECT MAX(project_fact) ;`

*example*      `SELECT MAX(Payment) ;`

**MIN** Returns the minimum value of all numbers in the set (e.g., Sales)

*syntax*      `SELECT MIN(project_fact) ;`

*example*      `SELECT MIN(Payment) ;`

**AVG** Returns the average value of all numbers in the set (e.g., Sales)

*syntax*      `SELECT AVG(project_fact) ;`

*example*      `SELECT AVG(Payment) ;`

## 7.2.1 COUNT

Globally COUNT returns a count of values for a selected attribute (e.g., as shown on the first syntax definition). But in GoodData, there is an option to pass a second attribute in COUNT definition, referring to the data set where you would like to perform the count (as shown on the second syntax definition).

*syntax*      `SELECT COUNT(project_attribute) ;`

*example*      `SELECT COUNT(Employees) ;`

This example counts the number of rows entering that are being aggregated (e.g. Employees). In it's default incarnation, it will count the number of unique values in an attribute.

*example*      `SELECT COUNT(Employees) ;`

If you use the count function with a primary key of it's dataset (specified as a Connection Point) then this will count the number of rows of that dataset, but now you will receive the number of unique records in that column.

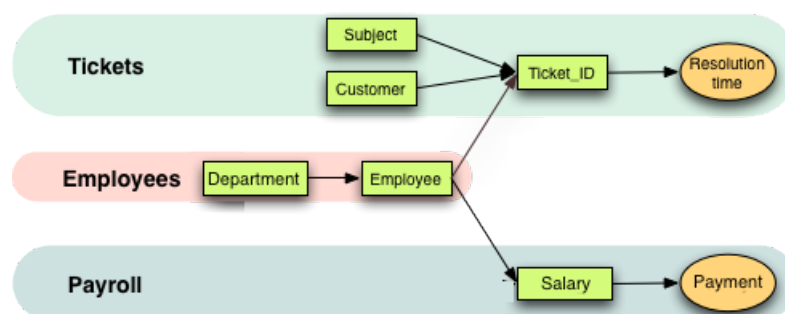


Figure 11: HR demo project LDM – extended version

The issue gets a bit more complicated with multiple datasets. Let's say we have a shared dataset – for example the Employees dimension. It is connected to both Tickets dataset (support tickets were handled by employees) and to Salary dataset, containing salary data. You can see the extended LDM on figure 11.

Now with the enhanced LDM, adding a metric `SELECT COUNT(Employee)` to a report will render both dataset Tickets and Salary unusable. The reason is that COUNT will scan all lines of the Employee attribute, but only some of them might be actually connected to Ticket\_ID or Salary\_ID. Thus, a second, optional (but very useful) parameter is used in the COUNT function:

**syntax**        `SELECT COUNT(project_attribute1, project_attribute2);`  
**example**       `SELECT COUNT(Employee, Salary_ID)`

This will count the number of unique employees that are listed in the Salary dataset (connected to Salary\_ID). As a result, no employees have corresponding rows to the Salary dataset scope are used and you can use all other attributes/metrics from the dataset.

## 7.3 Simple Arithmetic

Once a fact has been aggregated and saved as a metric, it can be used in additional computations. You can modify the result by adding to it, multiplying it, etc. using supported mathematical operators: +, -, \*, /.

**syntax**        `SELECT project_metric1 Operator project_metric2;`  
**example**       `SELECT Revenues - Costs;`  
**example**       `SELECT Profit * 0.03;`

Where Revenues, Costs and Profit are previously defined metrics.

## 7.4 Filters

Filters allow us to narrow the set of data from which the metric is computed. We can also combine filters as itself (see the section 2) and use them for conditional filtering. It's also possible to use the time macros (section 3.2) along with some simple arithmetic (e.g., using {Previous} is the same as {This} - 1).

### 7.4.1 Conditional filtering

Filtering is about defining the set of data that you are going to analyse. You can specify one or more expressions that apply to the attributes or metrics.

**syntax**        `SELECT project_metric WHERE <expression>`

For example you can specify that your metric is going to only compute the Department for HQ Marketing and HQ Human Resources. This limitation is defined in the expression clause. The expression clause can contain attributes, facts, metrics, functions, and logical operators (AND, OR, NOT), operators >, <, >=, <=, =, IN, and NOT IN functions.

Payments for year 2006 and first four months of 2007 are counted by following metric

**example**       `SELECT SUM(Payment) WHERE Year = 2006`

```
OR Month IN (January 2007, February 2007,
March 2007, April 2007);
```

You can also combine the dates with facts as in this metric, where we sum all the big payments (higher than 10000) for year 2006 and first four months of 2007.

**example**      `SELECT SUM(Payment) WHERE Year = 2006`  
                  `OR Month IN (January 2007, February 2007,`  
                  `March 2007, April 2007) AND (Payment > 10000);`

For more examples see the section 2 at page 6.

We can use these all of the previous clauses to create fixed numbers which can be used in other computations to create complex metrics. Fixing numbers is achieved by locking the aggregation level to an attribute in one or more dimensions. This lock is made with BY clause.

## 7.5 BY Clause

By default, each metric is aggregated over a scope of inheritance by its position in the pivot table. The intersection (resulting set) of Month(March) and Department(HQ Marketing) will aggregate all rows that have those corresponding attribute values. This allows metrics to be easily drilled into (drill into March to individual days and the metric starts showing numbers for individual days). You might want to override this behaviour though when using the metric as part of some larger calculation.

BY in fact locks the value of the metric at the aggregation level specified by the attribute after the BY clause. Multiple attributes separated by commas from different dimensions can be specified. Please note that if you lock the value by Year and set the active report to display at a lower aggregation level (e.g., Month), the numbers returned would be the computed value for the whole Year.

**syntax**      `SELECT project_metric BY project_attribute;`

**example**      `SELECT Payment BY Year;`

**example**      `SELECT Payment / (SELECT Payment BY Year);`

The second metric example returns variable Payment (determined by how the metric is sliced and diced in the report) divided by fixed metric Yearly Payment. The following example uses the extended version of LDM (see figure 11 on page 24) example is solving situation when we need to know how many hours were spent on resolving tickets in a particular month, out of all hours in a quarter – we need both the month and quarter aggregation levels in a metric:

**example**      `SELECT (SELECT SUM(Resolution time)) /`  
                  `(SELECT SUM(Resolution time) BY Quarter);`

This metric results in following table. For better orientation we also added two simple aggregation metrics `SELECT SUM(Resolution time) BY Quarter` and `SELECT SUM(Resolution time)` for the whole resolution time.

You can see that the `SELECT SUM(Resolution time) BY Quarter` metric displays the identical result for all month in the same quarter, since its computing results for all months that belong into that quarter. If the report's aggregation level is higher than the one defined in the metric BY clause, the metric's aggregation is ignored.

Solved Month	SUM	SUM BY Quarter	SUM/SUM BY Quarter
7/2009	121.95k h	381.60k h	31.96%
8/2009	107.15k h	381.60k h	28.08%
9/2009	152.50k h	381.60k h	39.96%
10/2009	170.72k h	495.99k h	34.42%
11/2009	184.06k h	495.99k h	37.11%
12/2009	141.21k h	495.99k h	28.47%
1/2010	267.41k h	859.95k h	31.10%
2/2010	319.65k h	859.95k h	37.17%
3/2010	272.89k h	859.95k h	31.73%
4/2010	178.73k h	474.43k h	37.67%
5/2010	250.10k h	474.43k h	52.71%
6/2010	45.61k h	474.43k h	9.61%

Figure 12: Results table of share (%) metric with BY clause

Continuing with this example, if we replace the Month attribute with a Year attribute, the differences between the metrics will be ignored. We say that the BY clause forces the lowest granularity of the metric to the value for the entire quarter. The situation is shown on following figure.

Solved Year	SUM	SUM BY Quarter	SUM/SUM BY Quarter
2009	921.91k h	921.91k h	100.00%
2010	1334.39k h	1334.39k h	100.00%

Figure 13: Results table of share (%) metric with BY Year

This metric SUMs all the Payments across whole data set, it means, that it sums Payments from each quarter (e. g. Q1, Q2) in all years in the data set.

**example** `SELECT SUM(Payment) BY Quarter;`

The BY clause supports multiple attributes. For example the

**example** `SELECT SUM(Payment) BY Quarter, Department;`

forces aggregation to the Quarter and Department levels. If you add this metric to a report with the Department attribute in the rows and the Year attribute in columns, you'll see different numbers for each cell (as on following figure).

Once you change the Year to Month or Day then values for all months or days are going to be the same because of the forced aggregation levels. Once you go below the lock level, the values are going to be the same for these children attributes. Obviously, the report with Month and Department attributes shows all the same numbers.

### 7.5.1 BY ALL Clause

In short, the BY Quarter clause stops breaking down the payments amounts by anything smaller than a quarter (e. g. months and days). BY ALL clause locks the value of the metric

Year	2006	2007
Department	Payment [Sum]	Payment [Sum]
HQ Finance and Accounting	233,280.00	405,840.00
HQ General Management	180,480.00	185,520.00
HQ Human Resources	56,040.00	122,160.00
HQ Information Systems	161,160.00	168,840.00
HQ Marketing	120,480.00	182,160.00
Store Management	1,264,440.00	2,376,240.00
Store Permanent Checkers	335,400.00	642,720.00
Store Permanent Stockers	148,560.00	297,840.00
Store Temporary Checkers	100,842.00	110,555.00
Store Temporary Stockers	76,506.00	82,782.00

Figure 14: Sum of all Payments BY Quarter and Department

Month	Jan	Feb	Mar	Apr
Department	Payment [Sum]	Payment [Sum]	Payment [Sum]	Payment [Sum]
HQ Finance and Accounting	53,260.00	53,260.00	53,260.00	53,260.00
HQ General Management	30,500.00	30,500.00	30,500.00	30,500.00
HQ Human Resources	14,850.00	14,850.00	14,850.00	14,850.00
HQ Information Systems	27,500.00	27,500.00	27,500.00	27,500.00
HQ Marketing	25,220.00	25,220.00	25,220.00	25,220.00
Store Management	303,390.00	303,390.00	303,390.00	303,390.00
Store Permanent Checkers	81,510.00	81,510.00	81,510.00	81,510.00
Store Permanent Stockers	37,200.00	37,200.00	37,200.00	37,200.00
Store Temporary Checkers	17,721.00	17,705.00	17,653.00	17,572.00
Store Temporary Stockers	13,288.00	13,363.00	13,147.00	13,299.00

Figure 15: Sum of all Payments BY Quarter and Department changed to Month

at the highest possible aggregation level in that hierarchy (one dimension). Let's say we'd like to create showing percentage of Resolution time not just out of the Quarter or Year, but all time. Since the highest-level attribute in the date dimension is Year, we need a new construct BY ALL.

There is one problem with this approach. One might need to compute yearly sales as a percentage of total sales. There is a slight problem as "year" is the topmost attribute in the time hierarchy. So we do not have anything above it for our BY clause. Fortunately there is a magic ALL keyword that helps us handle such situations.

ALL keyword forces aggregation of the whole data set for a particular hierarchy defined after the BY ALL clause. So the metric

**example** `SELECT SUM(Payment) BY ALL Year;`

always shows the same value (the total of salary per the whole dataset) no matter what time attribute you put in your report.

However, if you substitute the Department attribute for the Year attribute to the report, the

result is the same as in case of simple `SUM(Payment)` metric. No aggregation level is forced, since Department is not part of the time (Year) hierarchy. Obviously, if we were to use the `SUM(Payment) BY ALL Department`, then aggregation by department would be forced and the report would show the same value for all departments.

This fixes the aggregation for all years, but still can break down by attributes from other dimensions. This is best shown by adding a second attribute from a different dimension into our report

*example*        `SELECT SUM(Payment) BY ALL Year;`

### 7.5.2 BY project\_attribute ALL IN ALL OTHER DIMENSIONS

This complements the BY statement by specifying how the aggregation of the metrics should be calculated in all other dimensions not previously specified in the BY statement. Omitting ALL IN ALL OTHER DIMENSIONS allows the metric to be sliced and diced in other dimensions.

*syntax*        `SELECT project_metric  
                 BY project_attribute ALL IN ALL OTHER DIMENSIONS;`

It is useful when we need to create a metric that is fixed at highest aggregation (undrillable) in all dimension but the date dimension, where it is fixed at the Quarter level:

*example*        `SELECT SUM(Resolution time)  
                 BY Quarter, ALL IN ALL OTHER DIMENSIONS;`

Another similar construct is BY ALL IN ALL OTHER DIMENSIONS EXCEPT FOR project\_attribute. The only difference between these two constructs is that EXCEPT FOR project\_attribute preserves natural aggregation in the dimension specified by the project\_attribute (e. g., date dimension specified by Quarter) but Quarter, ALL IN ALL OTHER DIMENSIONS construct will never drill in the date dimension below the level specified by Quarter.

*syntax*        `SELECT project_metric  
                 BY ALL IN ALL OTHER DIMENSIONS  
                 EXCEPT [FOR] project_attribute`

By adding the EXCEPT FOR project\_attribute to the BY ALL clause specifies an exception where the metric will be sliced and diced by the specified attribute (and its hierarchy) if the attribute is contained in the report (or any other attribute from the same hierarchy).

*example*        `SELECT Payment  
                 BY ALL IN ALL OTHER DIMENSIONS  
                 EXCEPT FOR Date;`

### 7.5.3 BY ALL IN ALL OTHER DIMENSIONS

This clause locks the value of the metric at the highest possible aggregation level across all dimensions. Therefore, this returns a grand total that is indivisible. Simply said, BY ALL IN

ALL OTHER DIMENSIONS clause excludes attributes one by one can be pretty annoying in projects with hundreds of attributes.

**syntax**      `SELECT project_metric BY ALL IN ALL OTHER DIMENSIONS`

**example**      `SELECT Payment /  
                  (SELECT Payment BY ALL IN ALL OTHER DIMENSIONS);`

In this example, this metric returns variable Revenues (determined by how the metric is sliced and diced in the report) divided by a fixed grand total for Revenues.

If you want to exclude all attributes, you can use the OTHER and IN ALL OTHER DIMENSIONS keywords and define metric that forces aggregation level to the whole data set in all dimensions:

**example**      `SELECT SUM(Sale Amount) BY ALL IN ALL OTHER DIMENSIONS;`  
Metric that forces aggregation level to Month in the time dimension and to the whole data set in all other dimensions:

**example**      `SELECT SUM(Sale Amount) BY Month,  
                  ALL IN ALL OTHER DIMENSIONS;`

Metric that computes a percentage to the whole data set for any attribute that you add to the report:

**example**      `SELECT    SUM(Sale Amount) /  
                  (SELECT SUM(Sale Amount)  
                  BY ALL IN ALL OTHER DIMENSIONS);`

## 7.5.4 WITHOUT PARENT FILTER

Simply said WITHOUT PARENT FILTER clause excludes filters of active report. This clause allows to compute the metric with no regard to any filter or metric currently in place in the active report. It's really handy when we are reusing any metric in the report and we need to omit any applied filters.

**syntax**      `SELECT project_metric BY ALL project_attribute  
                  WITHOUT PARENT FILTER;`

This clause results the amount (e.g. SUM) for all Years even if some Years were filtered out of the active report.

**example**      `SELECT Payment -  
                  (SELECT Payment BY ALL Year  
                  WITHOUT PARENT FILTER);`