Set Analysis - Análisis de Conjuntos

José Manuel Martínez Mayoral

Julio 2017

# Contenido

[Contenido 2](#_Toc491706188)

[Introducción 3](#_Toc491706189)

[Sintaxis 4](#_Toc491706190)

[Identificadores de conjuntos 6](#_Toc491706191)

[Ejemplos: 7](#_Toc491706192)

[Operadores de Conjuntos 8](#_Toc491706193)

[Ejemplos 9](#_Toc491706194)

[Modificadores de Conjuntos 10](#_Toc491706195)

[Selección basada en valores conocidos. 11](#_Toc491706196)

[Selección con Comodines. 13](#_Toc491706197)

[Selección de todos los valores. 15](#_Toc491706198)

[Selecciones numéricas con límites 16](#_Toc491706199)

[Selecciones usando una variable 17](#_Toc491706200)

[ Para almacenar una cadena de búsqueda 17](#_Toc491706201)

[ Para almacenar el conjunto de elementos 18](#_Toc491706202)

[ Para almacenar elementos numéricos 19](#_Toc491706203)

[ Para almacenar todo el modificador 19](#_Toc491706204)

[ Para almacenar una función 20](#_Toc491706205)

[Selecciones usando una función 21](#_Toc491706206)

[ Funciones numéricas 21](#_Toc491706207)

[ Funciones que devuelven elementos 22](#_Toc491706208)

[Selecciones usando P() y E() 25](#_Toc491706209)

[Selecciones usando otro campo 27](#_Toc491706210)

[ Cuando ambos campos están en el mismo registro 27](#_Toc491706211)

[ Cuando ambos campos están en tablas diferentes 27](#_Toc491706212)

[Selecciones usando una sentencia IF() 29](#_Toc491706213)

[Tratamiento de nulos 30](#_Toc491706214)

[Con operadores de conjuntos 30](#_Toc491706215)

[Usando la función NullCount() 30](#_Toc491706216)

[Usando la función E() 31](#_Toc491706217)

[Conjuntos vacíos 32](#_Toc491706218)

[REFERENCIAS: 33](#_Toc491706219)

# Introducción

A menudo, cuando construimos expresiones en las que usamos funciones de agregación como SUM, AVG, MAX… etc, queremos que la función se ejecute sólo sobre un subconjunto de registros del conjunto total de registros cargados en la aplicación. Además estas funciones aplican por defecto sobre la selección actual de valores de campos y es posible que nosotros queramos que no se tenga en cuenta esa selección o, incluso, que se tenga en cuenta pero añadiendo más condiciones.

Para los novatos en QlikView y Qlik Sense que estamos familiarizados con SQL, la primera intención es usar una cláusula WHERE, pero ni QlikView ni Qlik Sense tienen cláusulas WHERE que se puedan usar dentro de una expresión.

La siguiente opción es comenzar a construir la expresión usando expresiones condicionales con IF, pero esta función siempre se aplicará a la selección actual y, aun suponiendo que eso no nos importe, penalizan mucho el rendimiento de la aplicación sobre todo cuando hay muchas anidadas.

La solución correcta es usar análisis de conjuntos.

El análisis de conjuntos es una herramienta muy avanzada y potente. Su sintaxis es muy simple, aunque a primera vista pueda parecer todo lo contrario. Permite seleccionar subconjuntos de registros distintos de la selección que esté activa en ese momento, operar entre ellos y forzar selecciones.

Hay que pensar en el análisis de conjuntos como una agregación condicional. La condición en sí misma es como una selección que es evaluada antes de que el gráfico sea calculado. Por eso no es posible tener una expresión que sea evaluada fila por fila.

SUM( {$<Año={2016}>} Ventas)

El análisis de conjuntos de la sentencia anterior está seleccionando las ventas sólo del año 2016, independientemente del año que el usuario haya seleccionado en la aplicación, y devuelve la suma acumulada.

El equivalente en SQL sería SELECT SUM(Ventas) WHERE Año=2016;

Y también es el equivalente a la sentencia SUM(IF(Año=2016,Ventas,0) ) pero ojo; esta sentencia sí tendría en cuenta el Año que el usuario ha seleccionado en la aplicación. Si está seleccionado otro año distinto de 2016, el resultado sería 0.

 *Las expresiones de conjunto solo se pueden usar en los gráficos, nunca en el script.*

# Sintaxis

En el ejemplo anterior hemos visto una sentencia creada con análisis de conjuntos. A pesar de lo que pueda parecer en un primer vistazo, es una sentencia sencilla que contiene ya todos los elementos que vamos a usar continuamente.

Una expresión de conjuntos siempre va dentro de una función de agregación como

SUM(Ventas)

El primer paso es escribir los limitadores de la expresión: **{…}**. Una expresión de análisis de conjuntos siempre va entre llaves.

SUM({…} Ventas)

El segundo paso es definir el “conjunto de registros” (record Set) con el que debe operar la función de agregación. Se pueden usar diferentes conjuntos y operadores. Por ejemplo: ‘**$**’ para las

selecciones actuales, ‘1’ para todos los registros cargados en la aplicación, ‘1-$’ para los registros excluidos en la selección … etc. En un capítulo posterior se describirán con más detalle.

SUM({$....} Ventas)

En la sentencia anterior vamos a aplicar la función de agregación SUM( ) al conjunto de registros acotado por las selecciones actuales, pero quizá sólo queramos aplicarlo a un conjunto de registros más pequeño. Para eso usamos un “modificador de conjunto” (set modifier).

El modificador de conjunto se escribe entre **<…>**, está definido por uno o más campos separados por comas y sirve para añadir o cambiar la selección actual.

SUM({$**<…>**} Ventas)

Cada uno de esos campos tiene su propio “conjunto de elementos” (element set) que se pueden escribir como una “lista de valores” o como una función: P() o E(). Cuando se trata de una lista de valores, el conjunto de elementos va delimitado por llaves. **{…}**

SUM({$<Fecha={…}>} Ventas) o SUM($<Fecha=P(…)>} Ventas)

La lista de valores puede estar delimitada por dobles comillas “…” para indicar que es una “cadena de búsqueda”. De esta forma se seleccionarán los campos cuyo valor coincida con algún valor de la lista.

SUM({$<Fecha={**“…”**}>} Ventas)

A menudo se hacen búsquedas numéricas, en ese caso no es necesario delimitar la lista de valores con comillas dobles, se pueden poner los números directamente.

SUM({$<Año={2015}>} Ventas)

Y también búsquedas por comparación. En este caso la cadena de búsqueda, además de ir delimitada por dobles comillas, deberá iniciarse con un operador relacional.

SUM({$<Fecha={“<=…”}>} Ventas)

Para hacerlo más difícil, el valor con el que se comparan los valores del campo puede ser a su vez un valor calculado. Por tanto se necesitará una expansión de **$** para obtener el valor dentro de la expresión.

SUM({$<Fecha={“<=$(…)”}>} Ventas)

Dentro de la expansión de **$** se necesita una expresión que comience con un signo ‘**=**’ y contenga una función de agregación. Esta función de agregación se evaluará globalmente antes de que se calcule el gráfico.

SUM({$<Fecha={“<=$(=MAX(Fecha))”}>} Ventas)

Como se puede ver, una expresión de conjuntos puede tener varios niveles de anidación. Para evitar errores sintácticos se recomienda escribir a la vez los símbolos de apertura y cierre (paréntesis, llaves, comillas, etc) y volver un carácter hacia atrás para seguir escribiendo el resto de la expresión.

# Identificadores de conjuntos

Llamamos “Identificadores de conjuntos” a cada uno de los conjuntos de registros disponibles en la aplicación. Un sinónimo de “Identificador de conjunto” sería “nombre de conjunto” o simplemente “Conjunto”. En lo sucesivo utilizaremos “conjunto” o “identificador” indistintamente.

En cualquier aplicación existen los siguientes conjuntos:

|  |  |
| --- | --- |
| **Identificador** | **Descripción** |
|  1 | Es el conjunto completo de todos los registros cargados en la aplicación. |
|  $ | Es el conjunto de registros de la selección actual, o sea, un subconjunto de 1. Si no hay selecciones actuales, $ y 1 son lo mismo.  |
| Es el conjunto por defecto en una expresión de conjuntos y por eso, se puede omitir. SUM({$ <Año={2016}>} Ventas) es lo mismo que SUM({<Año={2016}>} Ventas) |
|  $n | Donde n es un número entero positivo. Hace referencia a la n-esima selección hacia atrás.  |
| $1 sería el conjunto de registros de la selección anterior (equivalente a pulsar el botón "Atrás").  |
| $0 es la selección actual (lo mismo que $ o que no poner identificador de conjunto) |
| $\_n | Donde n es un número entero positivo. Hace referencia a la n-esima selección hacia adelante.  |
| $\_1 sería el conjunto de registros de la selección siguiente (equivalente a pulsar el botón "Adelante")  |
| $0 es la selección actual (lo mismo que $ o que no poner identificador de conjunto) |
| marcador\_id | nombre\_marcador | Se pueden utilizar marcadores de servidor y de app como identificadores de conjunto. Se pueden emplear, o bien el marcador ID, o el nombre del marcador. Por ejemplo BM01 o MyBookMark. |
| Únicamente se utiliza la parte de la selección de un marcador. Los valores no se incluyen. Por lo tanto no es posible utilizar campos de entrada en los marcadores para el análisis de conjuntos. |
| Estado alterno | Igual que en los marcadores, se puede usar el nombre de un estado alterno. |

## Ejemplos:

SUM(**{$}** Ventas) 🡪 Ventas de la selección actual. Lo mismo que escribir SUM(Ventas).

SUM(**{1}** Ventas) 🡪 Devuelve la suma total de ventas ignorando las selecciones actuales. O sea, la suma del campo “Ventas” de todos los registros cargados en la aplicación. Ojo. Se descarta la selección pero no la dimensión. En un gráfico que se use, por ejemplo, la dimensión “Producto”, cada Producto obtendrá un valor distinto.

SUM(**{1} TOTAL** Ventas) 🡪 En este caso se descarta tanto la selección como la dimensión, o sea, en un gráfico que se use, por ejemplo, la dimensión “Producto”, todos los productos obtendrán el mismo valor que será el total de ventas de la aplicación.

SUM(**{Marcador01}** Ventas) 🡪 Devuelve las ventas para el marcador con ID “Marcador01”

SUM(**{MiMarcador}** Ventas) 🡪 Devuelve las ventas para el marcador con nombre “MiMarcador”

SUM(**{Server\Marcador01**} Ventas) 🡪 Devuelve las ventas para el marcador de servidor con ID “Marcador01”

SUM(**{Document\MiMarcador}** Ventas) 🡪 Devuelve las ventas para el marcador de documento con nombre “MiMarcador”

SUM(**{$2}** Ventas) 🡪 Devuelve las ventas de 2 selecciones hacia atrás.

SUM(**{$\_1}** Ventas) 🡪 Devuelve las ventas de la selección siguiente.

SUM(**{Estado1}** Ventas) 🡪 Devuelve las ventas del estado alterno “Estado1”.

# Operadores de Conjuntos

Al igual que podemos operar con valores numéricos utilizando operadores numéricos ( +, -, /, \*) también podemos operar con conjuntos utilizando “operadores de conjuntos”. Básicamente estos operadores son los mismos que los numéricos pero tienen otro significado. Podemos operar conjuntos uniéndolos, restándolos (exclusión) intersectándolos o hallando su diferencia simétrica obteniendo siempre un nuevo conjunto.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operador** | **Operación** | **Descripción** |   |
| + | Unión | { A+B} Devuelve un conjunto compuesto por los registros de todos los conjuntos. |   |
| - | Exclusión | {A-B} Devuelve un conjunto con los registros que están en el primer conjunto pero no en el segundo. Sería algo así como restar conjuntos. |   |
| Complemento | Si se usa como operador unario (con un solo conjunto) devuelve el complementario del conjunto {-A} |   |
| \* | Intersección | {A\*B} Devuelve un conjunto formado por los registros comunes (que estén en los dos conjuntos) |   |
| / | Disyunción exclusiva (XOR) | {A/B} Devuelve el conjunto de registros que esté en el primer conjunto o en el segundo pero no en ambos a la vez. |   |

Las operaciones con conjuntos pueden ser tan complejas como necesitemos. Siempre se evaluarán de izquierda a derecha y la prioridad se modifica utilizando paréntesis.

El orden de prioridad es el siguiente:

1.- Paréntesis (de los más internos a los más externos)

2.- Operador unario (complementario)

3.- Intersección y Disyunción exclusiva

4.- Unión y exclusión

Cuando se opera con conjuntos se debe tener en cuenta que no cumplen la propiedad conmutativa. O sea, A + (B - C) es distinto que (A + B) - C y que (A – C) + B

## Ejemplos

SUM({1-$} Ventas) 🡪 Es la operación más usada. Devuelve las ventas de todo lo que no está seleccionado, o sea, todo lo que la selección actual excluye.

SUM({$\*VentasEuropa} Ventas) 🡪 Devuelve las ventas de los registros comunes en la selección actual y en los registros seleccionados por el marcador “VentasEuropa”. O lo que es lo mismo. Dentro de la selección actual suma sólo las ventas que pertenecen al marcador “Ventas Europa”

SUM({-($+Ventas01)} Ventas 🡪 Selecciona todos los registros contenidos en la selección actual y en el marcador con id “Ventas01” (comunes y no comunes) y devuelve todas las ventas menos las de esos registros (obsérvese el operador unario aplicando a toda la operación).

SUM({VendedoresEspaña+VendedoresPortugal} Ventas) 🡪 Devuelve las ventas de todos los vendedores que pertenecen a España o Portugal o a ambas (VendedoresEspaña y VendedoresPortugal son dos marcadores)

SUM({(VendedoresEspaña\*VendedoresPortugal)/VendedoresFrancia} Ventas) 🡪 Devuelve las ventas sólo de los vendedores que venden la vez en España y en Portugal y de los que venden en Francia. Si alguno vende a la vez en los tres países no lo selecciona.

# Modificadores de Conjuntos

Sirven para modificar la selección actual añadiendo y quitando dimensiones o modificando los valores seleccionados de las dimensiones.

Se escriben inmediatamente después del identificador de conjunto al que modifican y constan de uno o varios nombres de campos separados por comas ‘,’. Cada campo va seguido por un conjunto de elementos (element set) que representan la selección que debería efectuarse en el campo. Esta selección va encerrada entre llaves **‘{}’**, y a su vez todo el modificador va acotado por los signos **‘<’** y **‘>’**.

Los modificadores de conjuntos tienen preferencia sobre las selecciones actuales. O sea, si se modifica una dimensión que está en selecciones actuales, esta modificación reemplaza a la selección actual. Si la dimensión modificada no está en selecciones actuales, esta dimensión se añade a las selecciones actuales.

Ejemplo: Supongamos que hemos seleccionado por la aplicación Año=2016.

* SUM({$} Ventas) 🡪 nos da las ventas el año 2016 (el de la selección actual)
* SUM({$ <Año={2015}>} Ventas) 🡪 reemplaza al año 2016 en la selección actual y nos da las ventas del año 2015.
* SUM($ <Mes={“Enero”}>} Ventas) 🡪 Añade la dimensión Mes a la selección actual Año=2016 y sólo mostraría las ventas de Enero de 2016.

Se debe tener cuidado cuando se modifican conjuntos y tener muy claro lo que se está haciendo. Mientras que las selecciones en la aplicación siempre se hacen sobre valores relacionados (los disponibles en blanco), cuando modificamos conjuntos podemos modificar dimensiones que no están relacionadas entre sí o cuyos valores sean incompatibles, dando como resultado productos cartesianos o conjuntos vacíos.

Como ya avanzábamos en la introducción, es momento de recordar que la expresión de conjuntos sólo es aplicable en expresiones (definición de variables, expresiones de gráfico, expresiones de cuadros de texto…. Etc), no en el script.

Cuando se aplica a un gráfico, la expresión se evaluada antes de calcular y dibujar el gráfico. Esto significa que no se evalúa fila a fila o columna a columna. La selección se hace con anterioridad y todo el gráfico ya reflejará el resultado de esa selección, así que un modificador de conjuntos que modifique la dimensión Año provocará que cualquier gráfico que use esa misma dimensión muestre más o menos años (o incluso años diferentes) que los esperados por la selección actual.

## Selección basada en valores conocidos.

Sintaxis:

{Grupo <Dimensión1= {elemento1, elemento2,…} [,[, Dimension2= {elemento1, elemento2, …}] >}

Analicemos un ejemplo:

SUM({<[Provincia Venta] = {Madrid, ”Ciudad Real”, Barcelona}, Año={2014,1015} >} Ventas)

Imaginemos que las selecciones actuales son: Año=2016 y Mes=Enero, Febrero, Marzo

Observamos que:

* Estamos obteniendo la suma de las ventas sólo para las provincias de Madrid, Ciudad Real y Barcelona pero sólo del primer trimestre de los años 2014 y 2015.
* No hemos escrito el identificador de grupo, por tanto está tomando el de por defecto ‘$’ que es el de las selecciones actuales.
* El Mes de las selecciones actuales no se modifica, por tanto seguiremos teniendo Enero, Febrero y Marzo.
* El Año de las selecciones actuales si se modifica pero no se añade a lo seleccionado sino que lo reemplaza. Por eso tenemos las ventas de 2014 y 2015 y no las de 2016.
* Hemos añadido una dimensión a la selección actual (Provincia Venta) y hemos modificado otra (Año)
* Los nombres de las provincias se han escrito con y sin comillas dobles. La sintaxis acepta:
	+ Comilla simple **‘**. Pero como las aplicaciones entienden que todo lo escrito entre comillas simples es un literal, se recomienda no usarlo. De momento funciona pero no se descarta que un se corrija y deje de funcionar.
	+ Comillas dobles **“**. Se usa para cadenas de búsqueda de texto. Si el valor a buscar no lleva caracteres raros como espacios, guiones **‘-‘**, …etc se pueden quitar. En caso contrario son obligatorias.
	+ Corchetes []. Su uso es exactamente el mismo que las comillas dobles.
* Los elementos de los años se han escrito sin comillas. Esto indica que son valores numéricos. Si el campo fuera alfanumérico deberían ir con comillas dobles.
* El conjunto de elementos a buscar va siempre entre llaves **{}**
* La dimensión ‘Provincia Venta’ se ha escrito entre corchetes **[]** porque tiene un espacio en el nombre. Esto es obligatorio si contiene cualquier carácter raro como espacios, guiones,… etc. ‘Año’ no contiene nada raro en el nombre y por eso se ha escrito tal cual.



 **¿*Comillas Simples, dobles o corchetes?***

 *QlikView acepta estas tres sintaxis para búsquedas de texto:*

{<[Provincia Venta] = {**“**Madrid**”**}>}

{<[Provincia Venta] = {**[**Madrid**]**}>}

{<[Provincia Venta] = {**‘**Madrid**’**}>} *(No recomendada)*

*Y además, como no hay ningún carácter raro en el elemento buscado, en este caso también acepta:*

{<[Provincia Venta] = {Madrid}>}

Selección con Comodines.

A veces queremos obtener grupos de registros con un valor específico en un campo o que contengan algunos caracteres determinados.

Sintaxis:

{<Dimension = {“\*cadena buscada\*”}>} (Al ser una cadena va entre comillas dobles)

Existen dos caracteres comodín:

* \* (asterisco) = 0 ó más caracteres (AA\* devolverá AA, AA1, AAA, AAB, AA34J…)
* ? = 1 y solo 1 caracter (AA? Devolverá AA1, AAB, pero no AA ni AA3AJ)

Se pueden combinar varias selecciones con comodines en una misma expresión.

Ejemplo: {<[Provincia Venta] = {“Z\*a”, “?ale\*”, “Ceuta”, “?\*o\*”}>}

Devolvería los registros de Zaragoza, Zamora, Valencia, Palencia, Ceuta, León y Lugo.



 *La expresión de conjuntos es capaz de traducir algunos caracteres especiales, por eso*

 *también devolvería ‘León’ a pesar de que la ‘ó’ está acentuada*

Otro Ejemplo: Seleccionar todas las provincias excepto aquellas que empiecen por ‘A’

 {<[Provincia Venta] = **{“\*”}** – **{“A\*”}** >}

 O lo que es lo mismo {<[Provincia Venta] –=  **{“A\*”}** >}

 Y también {<[Provincia Venta] =– **{“A\*”}** >} (aunque esta da error de sintaxis, funciona.)

Otro más: Todas las provincias de la selección actual más las que empiecen por ‘A’

 {<[Provincia Venta] **+**=  **{“A\*”}** >}

 En cambio {<[Provincia Venta] =**+ {“A\*”}** >} da error y no funciona.



 *Cuidado porque las siguientes expresiones son muy diferentes:*

{<[Provincia Venta] **–**=  **{“A\*”}** >} y {<[Provincia Venta] =**–** **{“A\*”}** >}

*La primera quita de la selección actual todas las provincias que comiencen por ‘A’, mientras que la segunda selecciona todas las provincias excepto las que empiecen por ‘A’: {“\*”} está implícito por defecto aunque no esté escrito.*

*Cuando se usan operadores delante el signo igual:*

* *<Dimension += {Conjunto\_Elementos}> : Añade un nuevo conjunto a la selección actual.*
* *<Dimension –= {Conjunto\_Elementos}> : Quita el conjunto de la selección actual.*

## Selección de todos los valores.

Sintaxis:

{Grupo <Dimension1 = **{\*}** >} (**{\*}** para números, **{"\*"}** para texto)

O:

{Grupo <Dimension1 = >}

El resultado final de ambas sintaxis es el mismo.

Supongamos que en la selección actual tenemos la dimensión1.

* La primera sintaxis lo que hace es seleccionar todos los valores excepto NULL.
* La segunda lo que hace es deseleccionar todos los valores que tenga seleccionados.

El resultado final de ambas sintaxis es el mismo si no tenemos valores NULL en el campo. En las selecciones hechas a través de la aplicación es lo mismo seleccionar todos los valores en una lista de valores que no seleccionar ninguno.

Cuando existen valores NULL y queremos seleccionar TODOS los registros (incluidos NULL) la forma de hacerlo es usando {Grupo <Dimension1 = >}

Se debe tener mucho cuidado con el uso de dimensiones jerárquicas. En la aplicación lo normal es crear un cuadro de lista (list box) por cada uno de los campos que forman la dimensión y el usuario puede seleccionar en cada campo. Como los campos están relacionados, habrá valores que no están disponibles y otros sí.

En estos casos, con análisis de conjuntos, hay que tener la precaución de establecer los valores para cada dimensión porque el hacerlo sólo para una no implica que se haga automáticamente para las demás que están relacionadas

Imaginemos que tenemos una dimensión jerárquica Año, Mes y que hemos seleccionado en los cuados de lista Año=2016 y Mes=Enero, Febrero, Marzo

SUM({<Año={2015}>} Ventas) No nos dará las ventas de todo 2015 porque seguirán seleccionados los meses Enero, Febrero y Marzo. Para obtener las ventas de todo 2015 deberíamos resetear el Mes de esta forma: SUM({<Mes=, Año={2015}>} Ventas)

## Selecciones numéricas con límites

En la mayoría de los casos, las claves de las tablas son numéricas. Esto es debido a que la unión de tablas a través de claves numéricas es más rápida que a través de cadenas de texto.

En las expresiones de conjuntos se pueden limitar las búsquedas entre 2 valores numéricos.
La sintaxis es un poco rara porque a pesar de tratarse de números, la expresión debe ir entre comillas dobles.

Ejemplos:

{<Dia = **{“<10”}** >} : Días del 1 al 9

{<Dia = **{“>=10 <= 20”}** >} : Días del 10 al 20 ambos incluidos

{<Dia = **{“>=10 , <= 20”}** >} : idéntico que el anterior (la coma **‘,’** es opcional)

{<Dia = {1, 2, 3} + **{“>=25 < 31”}** >} Días 1, 2 y 3 y del 25 al 30 (el 31 está excluido)



 *Se debe poner especial cuidado en no dejar ningún espacio entre la doble comilla y el signo
 de mayor* ***‘>’*** *o menor* ***‘<’*** *porque genera errores.*

{<Dia = {**“<**10”} >}

## Selecciones usando una variable

Sintaxis:

{Grupo <Dimension = {$(NombreVariable)} >}

 O

{Grupo <Dimension = {$(**=**NombreVariable)} >}

 O

{Grupo <Dimension = {$(**#**NombreVariable)} >}

 O

{Grupo <Dimension = {$(**#=**NombreVariable)} >}

La diferencia está en el uso de los signos almohadilla **‘#’** e igual ‘=’. Realmente no es una característica de la sintaxis del análisis de conjuntos si no de la expansión del signo ‘**$**’

* El signo igual ‘**=**’ 🡪 Evalúa la variable y el valor se utiliza en la búsqueda. Se suele usar cuando la variable contiene una fórmula y la definición de esta fórmula no comienza por el signo ‘=’ ej: SUM(Año)-1
* El signo almohadilla ‘**#**’ 🡪 Se utiliza cuando la variable devuelve un valor numérico que contiene un separador decimal. Si el separador decimal definido por defecto es la coma ‘**,**’ el resultado se interpretaría como una lista de 2 números enteros separados por una coma. Como no queremos que eso suceda, necesitamos convertir el número en una representación numérica válida y para ello basta con incluir este signo que internamente traducirá la coma por un punto decimal.

### Para almacenar una cadena de búsqueda

Podemos usar la variable para almacenar una cadena de búsqueda con comodines

Ejemplo: Hemos definido la siguiente variable:



La expresión

SUM({<[Provincia Venta] = {“$(varSeleccionProvincias)”}>} Ventas)

 Devolverá las ventas de León y Lugo, mientras que la expresión

 SUM({<[Provincia Venta] = {$(varSeleccionProvincias)}>} Ventas)

Dará error.

¿Dónde está la diferencia? En las dobles comillas que encierran el nombre de la variable.

La cadena de búsqueda de la variable no está delimitada por dobles comillas. Cuando se hace la sustitución de la variable en la primera expresión, al estar e nombre de la variable entrecomillado, la cadena de búsqueda queda sintácticamente perfecta: [Provincia Venta]= {**“**?\*o\***”**}

Mientras que en la segunda expresión queda sintácticamente incorrecta: [Provincia Venta]= {?\*o\*}

La solución para esta segunda expresión sería entrecomillar la cadena dentro de la variable:



Ahora la segunda expresión sería correcta pero la primera daría error porque su sintaxis tiene dos dobles comillas: [Provincia Venta]= {**“**“?\*o\*”**”**}

### Para almacenar el conjunto de elementos

La variable puede contener varios elementos separados por comas.

* Cuando son de texto, dependiendo de su nombre, deberán seguir la sintaxis normal de estar entre dobles comillas o corchetes **[].**

Ejemplo. Tenemos definida la siguiente variable:



El elemento ‘Las Palmas’ va entre comillas dobles porque contiene un espacio.

* Cuando son numéricos, en cuyo caso ni la variable ni la lista de elementos deberían ir entre comillas dobles.

Ejemplo:



 Y la expresión sería: {<Comercial\_Id = {$(varIdComerciales)} >}

### Para almacenar elementos numéricos

En este caso no se trata de contener el conjunto de elementos sino un solo valor con la intención de usarlo como límite en una cadena de búsqueda.

En este caso el nombre de la variable deberá ir obligatoriamente entre dobles comillas.

Ejemplo: Si definimos una variable que tenga la venta media



La podemos usar en la siguiente expresión para obtener los registros cuyas ventas estén por encima de la media: {<Ventas = {“>$(varVentaMedia)”} >}

Y lo mismo para obtener las ventas entre dos fechas. Las fechas deben estar en formato numérico. {<[Fecha Venta] = {“>=varInicioPeriodo <=varFinPeriodo”} >}

Para almacenar una dimensión

De la misma manera que una variable puede contener elementos, también puede contener dimensiones.

Ejemplo: Sum( {$ <$(varDimension) ={“\*”} >} Ventas)

Como el nombre de la dimensión puede contener guiones, espacios y otros caracteres raros, sería más seguro encerrar la variable entre corchetes **[].**

Sum( {$ <**[**$(varDimension)**]** ={“\*”} >} Ventas)

### Para almacenar todo el modificador

Si podemos almacenar el conjunto de elementos o la dimensión ¿Por qué no almacenar el modificador de conjunto completo o una parte de él?

La variable debe contener una sintaxis correcta excepto los caracteres **‘<’** y **‘>’**

Ejemplos:

SUM( {$ <$(varModificador) >} Ventas)

SUM( {$ <$(varModificador), Mes={“Enero”,”Febrero”,”Marzo”} >} Ventas)

Donde $(varModificador) puede contener cualquier cosa, como:

 [Provincia Venta] = {Madrid, ”Ciudad Real”, Barcelona}, Año={2014,1015}

### Para almacenar una función

Podemos almacenar funciones complejas dentro de variables para usarlas una y otra vez dentro de expresiones de conjuntos.

Dos de las más habituales son la del Año anterior al seleccionado y la del año actual independientemente cuál sea la selección actual.



Ejemplos:

SUM({<Año = {"$(varAñoAnterior)"}>} Ventas)

SUM({<Año = {"<=$(varAñoActual)"}>} Ventas)

Selecciones usando una función

A veces no conocemos de antemano los elementos que componen la cadena de búsqueda, así que tenemos que utilizar una función para hallarlos.

### Funciones numéricas

Devuelven siempre un valor numérico

 Sintaxis:

 {Grupo <Dimension = {“$(=func(parametros))”} >}

Observemos que:

* La cadena de búsqueda se escribe entre comillas dobles. **“”**
* Se utiliza la expansión de signo **$**
* Justo antes de la función se escribe un signo igual ‘**=**’ para que la expresión sea evaluada.

Ejemplos:

< Año = {“$(=MAX({1} Año))”}>

Devuelve el último año cargado en la aplicación

Observemos que:

* La función MAX, como cualquier otra función de agregación sólo tiene en cuenta la selección actual.
* Hemos usado un conjunto anidado {1} dentro de la función para usar todos los registros cargados en la aplicación. Si no lo hiciéramos, nos devolvería el año mayor de los seleccionados por el usuario.
* Usamos dos signos igual: 1 antes de las llaves para indicar la cadena de búsqueda del campo y otro justo antes de la función para que sea evaluada.

Si lo que queremos es el año anterior al último año seleccionado, usaríamos cualquiera de estas dos cadenas:

< Año = {“$(=MAX({$} Año)-1)”}>

< Año = {“$(=MAX(Año)-1)”}> {$} es la selección actual y está implícito por defecto.

### Funciones que devuelven elementos

En lugar de devolver un solo valor numérico devuelven un conjunto de elementos de búsqueda.

 Sintaxis:

 {Grupo <Dimension = {“=func(parámetros)”} >}

Esta vez no usamos la expansión de signo **$.**

Ejemplo:

SUM({ $ <Comercial = {“=Sum({1<Año = {2016}, [Provincia Venta] = {Madrid, [Ciudad Real]}>} Ventas) > 250000”} >} Ventas )

Devuelve las ventas de los comerciales que han vendido más 250.000€ en 2016 en Madrid y/o
Ciudad Real (sólo en una o en ambas).

Obsérvese que:

* Partimos de la selección actual y dentro del modificador, para obtener las ventas de 2016 de Madrid y Ciudad Real de más de 250.000€, anidamos una nueva selección del conjunto total de registros cargados en la aplicación ‘{1}’.
* Ciudad Real se ha escrito entre corchetes. La función va escrita entre comillas simples, por eso debemos usar comilla simple o corchetes para encerrar los elementos del conjunto de búsqueda.

Hasta ahora estamos acostumbrados a buscar valores relacionados con la dimensión en cuestión, como <Año={2015}> o <Comercial = {“Carlos”,”Jose”}> y puede resultar extraño buscar un valor de un tipo diferente al de la dimensión.

Da la impresión que en vez de buscar un nombre de comercial, obtenemos un número (el total de ventas) pero esto no es del todo correcto. La búsqueda no está devolviendo un valor si no un subconjunto de registros, con todos sus campos, que cumplen la condición.



 *Las expresiones de conjunto no devuelven valores, si no conjuntos de registros.*

Más ejemplos:

Queremos los comerciales con ventas de más de 250.000 € en Madrid y Ciudad Real en 2016 y de estos, sólo a los que vendieron más de 10.000 € en diciembre de 2015 en cualquier provincia. O lo que es lo mismo; al primer conjunto le tenemos que quitar todos los que vendieron menos de 10.000 € en diciembre de 2015 en cualquier provincia.

Para hacerlo necesitamos 2 conjuntos de registros: {<conjunto1> - <conjunto2>}

**Conjunto1** = < Comercial = {“=Sum({1<Año = {2016}, [Provincia Venta] = {Madrid, [Ciudad Real]}>} Ventas) > 250000”} >

**Conjunto2** = < Comercial = {“=Sum({1<Año = {2015}, Mes={[Diciembre]}, [Provincia Venta] = {[\*]}>} Ventas) < 10000”} >

Poniéndolo todo junto nos queda:

SUM({**1** **<** Comercial = **{**“=Sum({1<Año = {2016}, [Provincia Venta] = {Madrid, [Ciudad Real]}>} Ventas) > 250000”**}** **>** **-** **<** Comercial = **{**“=Sum({1 <Año = {2015}, Mes={[Diciembre]}, [Provincia Venta] = {[\*]}>} Ventas) < 10000”**}** **>**} Ventas)

Que se puede simplificar un poco:

SUM({**1** **<** Comercial = **{**“=Sum({1<Año = {2016}, [Provincia Venta] = {Madrid, [Ciudad Real]}>} Ventas) > 250000”**}** **-** **{**“=Sum({1 <Año = {2015}, Mes={[Diciembre]}, [Provincia Venta] = {[\*]}>} Ventas) < 10000”**}** **>**} Ventas)

¿Cuántos comerciales venden en más de una provincia?

Count({$<Comercial\_Id = {"=Count({1} DISTINCT [Provincia Venta]) > 1"}>} DISTINCT Comercial\_Id)

¿Quién son los 10 mejores comerciales?

SUM( < Comercial = **{**“=RANK(Sum(Ventas),4 ) <=10”} >} Ventas)

¿Quién son los 10 mejores comerciales de Madrid? (necesitamos un conjunto anidado)

SUM( < Comercial = **{**“=RANK(Sum({<[Provincia Ventas]={Madrid}>} Ventas),4 ) <=10”} >} Ventas)



 *Mucho cuidado con NO escribir*

 SUM({<[Provincia Ventas]={Madrid}, Comercial = **{**“=RANK(Sum(Ventas),4 ) <=10”} >} Ventas)

*Porque estaríamos haciendo la intersección entre la provincia de Madrid y los 10 mejores vendedores y quizá no obtengamos nada si ninguno de los 10 mejores comerciales es de la provincia de Madrid*.

Selecciones usando P() y E()

Hasta ahora hemos definido los elementos de forma explícita o mediante búsquedas. También se puede hacer usando definiciones de conjuntos anidadas.

Esto se hace con las funciones P() y E() que representan el conjunto de elementos posibles y excluidos, respectivamente, de una dimensión.

Ejemplos:

SUM({$ <Comercial = P({1 <[Provincia Venta] = {“Madrid”}>} Comercial) >} Ventas)

Devuelve las ventas de los comerciales de la selección actual que alguna vez han vendido en Madrid.
Si el usuario ha seleccionado Provincia Venta = Málaga y Año=2016, lo que tendríamos sería los comerciales que en 2016 han vendido en Málaga y que además, en algún año, han vendido en Madrid. Si no ha vendido nunca en Madrid no aparecería por muchas ventas que haya hecho en Málaga en 2016.

SUM({$ <Comercial = P({1 <[Provincia Venta] = {[Madrid]}>}) >} Ventas)

Idéntico al anterior. El campo de la función P() se puede omitir y devolverá los valores posibles de la dimensión especificada en la asignación externa; en este caso ‘Comercial’.

SUM({$ <Comercial = P({1 <[Producto] = {Pescado} >} Cliente) >} Ventas)

Devuelve las ventas de la selección actual pero sólo de aquellos comerciales que a su vez han actuado como clientes comprándonos pescado.

SUM({$ < Comercial = E({1 <{1 <Producto ={“Colín”}>} Ventas) >} Ventas)

Devuelve las ventas de los comerciales de la selección actual que nunca han vendido un colín. La función de elemento E() aquí devuelve la lista de comerciales excluidos. Aquellos que no están en la selección de ‘Colín’ en el campo ‘Producto’.

También podemos usar P() cuando tenemos dos dimensiones con los mismos valores pero que no están relacionadas entre sí y queremos hacer una selección en una de ellas que se propague a la otra.

Por ejemplo tenemos los campos Año y AñoPedido. Tienen los mismos valores pero no están enlazados. El usuario ha seleccionado un año en un cuadro de lista de la aplicación y queremos seleccionar todos los pedidos de ese mismo año a través del campo AñoPedido para saber su volumen de ventas.

SUM({<AñoPedido = P(Año) >} Ventas)

Selecciones usando otro campo

La forma más fácil de hacer una selección es basándonos en otro campo.

### Cuando ambos campos están en el mismo registro

 Sintaxis:

 {Grupo <Campo1 = Campo2>}

Esta sintaxis sólo se puede utilizar cuando ambos campos están en el mismo registro.

Ejemplo:

Queremos saber el volumen de ventas de los pedidos que se han hecho y entregado el mismo día.

SUM({<FechaPedido = FechaEntrega}> Ventas)



*Si los dos campos están en tablas diferentes, no funciona pero no da error. Simplemente devuelve 0*

Realmente lo que hace es un IF entre los dos campos, por lo que la comparación la va haciendo registro a registro. Esto consume muchos recursos de CPU, por lo que no se recomienda usarlo con más de **200** valores distintos.

### Cuando ambos campos están en tablas diferentes

 Sintaxis:

 {Grupo <Dimension = {“=condición lógica”} >}

Donde la condición lógica se creará comparando dos campos.

Ejemplos:

SUM({<comercial ={“=FechaPedido = FechaEntrega”}>} Ventas)

Se debe tener en cuenta que:

* La dimensión buscada no puede formar parte de la condición lógica. Esto no sería válido:
SUM(({<campo1 ={“=campo1 = campo2”}>} Ventas)
* Aquí no debemos usar la función P() porque esa función la usamos en campos que tienen los mismos valores pero que no están relacionados. Ahora estamos hablando de campos que sí están relacionados pero no están en el mismo registro.
* La dimensión buscada puede ser cualquier campo del mismo registro en el que esté el campo a agregar (en este caso Ventas). De hecho es una buena práctica crear un campo ficticio con autonumber() en la tabla del valor agregado.

Más ejemplos:

Obtener las ventas de todos los pedidos entregados en menos de 7 días desde que se hicieron.

ClaveAutonumber es un campo ficticio creado en el script con la función Autonumber()

SUM({<ClaveAutonumber = {“=FechaEntrega < FechaPedido - 7”}>} Ventas)

SUM({<ClaveAutonumber = {“=FechaEntrega < FechaPedido - $(vG.Retraso)”}>} Ventas)

En el segundo ejemplo se usa una variable para controlar los días de retraso.

Se podría hacer lo mismo usando IF(), pero no se recomienda porque sería mucho más lento porque Qlikview necesitaría calcular cada una de las líneas.

SUM( IF(FechaEntrega < FechaPedido – 7, Ventas))

IF() no necesita un tercer parámetro porque devolverá Null si la condición es falsa.

## Selecciones usando una sentencia IF()

En ocasiones ocurre que debemos aplicar un análisis de conjuntos u otro dependiendo de una selección hecha por el usuario.

Por desgracia no podemos usar la función IF(), así, sin más, dentro de un análisis de conjuntos. pero si podemos usar una función de selección o una variable que contenga la expresión.

Por ejemplo, si un usuario elige un solo producto sólo queremos operar o visualizar ese producto, pero si elige más de uno, entonces queremos operar o visualizar todos (no sólo los seleccionados).

La función de selección sería así:

=SUM({<***$(=IF(GetSelectedCount (Producto) = 1, '$', 'Producto='))***>} Ventas)

Y si queremos usar una variable (p. ej: **vL.Condicion**) que tenga la expresión, comenzaríamos creándola y luego asignándola la siguiente expresión:

=IF(GetSelectedCount (Producto) = 1, '$', 'Producto=')

Y sólo tendríamos que usar la variable dentro del conjunto:

SUM({<$(vL.Condicion)>} Ventas)

# Tratamiento de nulos

No existe una forma directa de identificar los valores nulos, pero se puede hacer indirectamente.

Existen tres formas de seleccionar los valores nulos en el análisis de conjuntos.

## Con operadores de conjuntos

En el análisis de conjuntos se puede usar los operadores de conjuntos para obtener el complementario de una selección, por ejemplo para obtener los registros excluidos.

Así, la expresión

 {$<Comercial={"\*"}>}

devuelve todos los comerciales excepto los NULOS, por tanto

 {1-$<Comercial={"\*"}>}

Devuelve su complementario, esto es, todos los que tienen el campo “Comercial” vacío.

Evitad la tentación usar {<campo ={}>} o usar {<campo ={“”}>} porque no funciona.



*Hasta la versión 11.20 SR4, funcionaba la siguiente expresión, después ha dejado de hacerlo:* {<Comercial=**–**{"\*"}>}

## Usando la función NullCount()

Se puede usar la función NullCount() para obtener los valores nulos. Así:

Concat({$<Comercial={"=NullCount(Telefono)>0"}>} distinct Comercial)

Devuelve los comerciales que no tienen teléfono.

## Usando la función E()

Esta función es especialmente útil para obtener los valores excluidos.

Ejemplo:

{$<Cliente = E({1<Producto={"\*"}>})>}

La expresión selecciona todos los clientes excluidos cuando se seleccionan todos los productos…. O lo que es lo mismo: Selecciona los clientes que no han comprado nada.

# Conjuntos vacíos

Puede ocurrir que el conjunto devuelto esté vacío. El resultado será 0. Esto puede ocurrir porque realmente no haya ningún registro que cumpla las condiciones del análisis o por un error.

Los errores más comunes son:

* Usar las llaves sin nada dentro: {} (recordad que en el tratamiento de nulos decíamos que era un error usar {<campo ={}>} con la intención de obtener los valores nulos).
* Escribir con errores los nombres de campos o miembros. QlikView distingue entre mayúsculas y minúsculas.
* Escribir una expresión errónea: errores de sintáxis, nombres de campos erróneos, faltan paréntesis, comillas… etc.
* Escribir una intersección errónea: por ejemplo entre cliente y producto. Si el cliente no compró el producto la respuesta correcta entre ambos será 0.
* Olvidar resetear un campo que está limitado por una lista de selección: por ejemplo, tienes el campo marca y una lista de selección con los productos. El usuario selecciona un producto y nosotros queremos contar los productos de las marcas ‘A’ y ‘B’.

COUNT( {<Marca={‘A’,’B’}>} Producto)

posiblemente devuelva ‘0’ si el producto seleccionado no pertenece a ninguna marca. Y si el producto pertenece a una de las dos marcas, devolverá sólo ‘1’.

Lo correcto es COUNT( {<Producto=, Marca={‘A’,’B’}>} Producto)

Esto es muy común que ocurra con fechas, cuando queremos obtener un valor acumulado de todo un año seleccionado de una lista de selección y nos olvidamos de que tenemos seleccionado un mes o un trimestre o un día.

Por tanto lo correcto sería hacer

SUM({<Mes=, Dia=, Trimestre=>} Valor)

REFERENCIAS:

Fabrice Aunez - The Set Analisys: <https://community.qlik.com/docs/DOC-4951>

Henric Cronstöm - Tratamiento de nulos: <https://community.qlik.com/docs/DOC-3155>

Denys Smakovskyi – Ignorar todas las selecciones excepto algunos campos específicos: <https://community.qlikview.com/docs/DOC-1334>