Luxms BI

Luxms BI. Описание технологии LPE

ВСТУПЛЕНИЕ

LPE КООВ ЗАПРОСОВ

2024-04-02

Оглавление

1 Введение

1.1 LPE выражения (Lux Path Expressions)

LPE выражения являются собственным языком программирования Luxms BI. Благдаря LPE выражениям вы можете:

- Производить тонкую настройку виджетов
- Осуществлять рассчеты “на лету”, в интерфейсе Luxms BI
- Использовать элементы, заданные в интерфейсе Luxms BI, в SQL-выражениях

1.1.1 Конкретные примеры использования LPE выражений

1. Параметры JSON конфигурации виджета, например `onClickDataPoint`:

```
onClickDataPoint:"lpe:setKoobFilters( 'source.koob',dim1, ['=',dim1])"
```

2. Рассчет фактов “на лету”:

```
measures: [
 'sum(unitprice*quantity)+avg(quantity/unitprice):s',],
```

3. Добавление фильтров, установленных пользователем в SQL запрос:

```
SELECT * FROM table1 WHERE ${filters(dt)}
```

2 LPE-выражения для вычислений в источниках данных (СУБД)

2.1 Введение

Для получения данных для визуализации сервер Luxms BI отправляет SQL запросы в источники данных, а полученный из источников результат перенаправляет на клиент Luxms BI. Данные для визуализации запрашиваются клиентской частью Luxms BI с помощью KOOB API. В частности, клиент передаёт на сервер Luxms BI имя куба, список столбцов, условия фильтрации. В настройках дэшлетов можно указать список вычисляемых столбцов и указать выражения LPE, которые будут преобразованы в соответствующий SQL диалект на сервере Luxms BI.

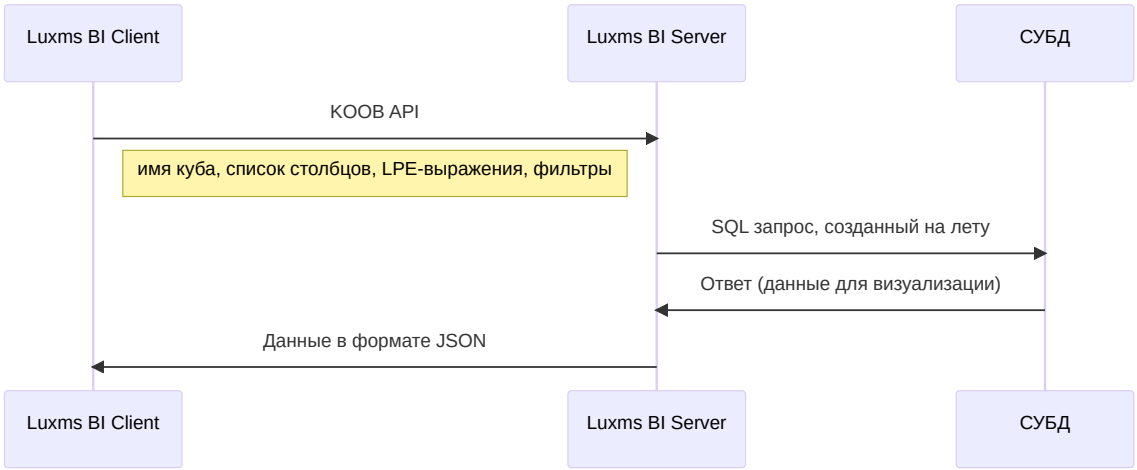


Рис. 2.1

> В Power BI такой подход получения данных называется DirectQuery

Пример LPE для вычисляемого столбца в дэшлете:

```
measures: [
  "(sum(sales_amount)+sum(price))/100:c",
  "max(price):max_price",
    ],
```

В данном примере используется оператор `:`, который аналогичен SQL выражению `AS` и служит для именования столбца в ответе из СУБД. Таким образом, данные для первого столбца вернутся под именем `c`, а для второго столбца — под именем `max_price`.

Если не использовать оператор `:` для вычисляемых столбцов, то имя столбца в ответе будет зависеть от версии и производителя СУБД, версии JDBC драйвера и т.д., и нельзя гарантировать, что это имя не изменится при обновлении компонентов.

Помимо алгебраических выражений можно также использовать условные конструкции:

```
measures: [
  "if(price + vat > 558, 'a', 'b'):c",
  "if(sum(price) < 0, sum(price), 0):c",
  "if(avg(a)=1,0,null):d",
      ],
```

Такая конфигурация заполнит столбец таблицы значениями по условию. Можно использовать как числовые значения, так и строковые.

Для корректной работы оба значения должны быть одного типа.

Ниже приведены другие примеры использования условных выражений в LPE конструкциях.

```
measures: [
  "if(sum(a) != 0 and sum(b) != 0, sum(a)*100/sum(b), 0):d",
  "countIf(quantity > 0 and quantity < 100):countif",
  "if(avg(q)=[1,null,3],0,avg(q)):d"
  ],
```

2.2 Поддерживаемые источники данных

Список поддерживаемых СУБД:

- Clickhouse
- Greenplum
- MS SQL
- Oracle (версии выше 9)
- PostgreSQL
- SAP HANA
- Teradata
- Vertica

СУБД, которые не вошли в список (MySQL, Cassandra и т.д.), можно использовать в качестве источника данных для простых дэшлетов, но сложные визуализации, например, Pivot Table, не будут работать.

2.3 Агрегационные (статистические) функции

2.3.1 avg(expr) – Среднее арифметическое

Функция `avg` возвращает среднее арифметическое значение от выражений `expr`, вычисленных для каждой строки куба, при этом учитываются фильтры, которые могут быть заданы настройками (конфигурацией) или пользователем в интерфейсе Luxms BI. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца или константа.

Примеры:

```
avg(col1)
```

```
avg(col1/col2*100.0)
```

2.3.2 corr($expr_1$, $expr_2$) – Коэффициент корреляции

Функция `corr` возвращает коэффициент корреляции между двумя выражениями, при этом учитываются фильтры, которые могут быть заданы настройками (конфигурацией) или пользователем в интерфейсе Luxms BI. В качестве `expr` можно использовать LPE выражения, но чаще всего используются имена столбцов.

Примеры:

```
corr(col1, col2)
```

```
corr(col1, col2 / col3)
```

2.3.3 count(expr) – Подсчет количества

Функция `count` возвращает количество строк в кубе, при этом учитываются фильтры, которые могут быть заданы настройками (конфигурацией) или пользователем в интерфейсе Luxms BI. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца или константа.

> Для подсчёта уникальных значений можно воспользоваться комбинацией двух функций: `count` и `distinct`

Примеры:

```
count(col1)
```

```
count(100+20)
```

```
count(distinct(colname))
```

Наличие параметра `expr` в LPE функции `count` объясняется стандартом SQL, где у функции `count` обязательно должен быть параметр. Однако, нужно помнить, что сам параметр не влияет на результат, который определяется исключительно условиями фильтров.

2.3.4 countIf(cond) – Подсчет количества при удовлетворении условия

Функция `countIf` возвращает количество строк в кубе, которые удовлетворяют условию `cond`.

Примеры:

```
countIf(col1 > 0 and round(col2) < 100)
```

```
countIf(col1/col2 > 1)
```

В выражениях `cond` можно использовать арифметические выражения, скобки, логические выражения, другие функции. Однако нельзя использовать агрегационные функции. Например, выражение `countIf(col1 < count(col1))` не поддерживается.

Функция `countIf` появилась в luxmsbi-pg 9.2.2

2.3.5 distinct(col) - Уникальные значения

Функция `distinct` модифицирует поведение функции `count` и позволяет получить количество уникальных записей. Для короткой записи рекомендуется использовать функцию uniq.

Пример:

```
count(distinct(col1))
```

Функция `distinct` не является самостоятельной функцией, её можно вызывать только в комбинации с функцией `count`!

2.3.6 max(expr) – Максимальное значение

Функция `max` возвращает максимальное значение для выражения `expr`, вычисленного для каждой строки в кубе, при этом учитываются фильтры, которые могут быть заданы настройками (конфигурацией) или пользователем в интерфейсе Luxms BI. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца.

Примеры:

```
max(col1)
```

```
max(col1+col2)
```

2.3.7 median(expr) – Медиана

Функция `median` возвращает медиану для выражения `expr`, вычисленного для каждой строки в кубе, при этом учитываются фильтры, которые могут быть заданы настройками (конфигурацией) или пользователем в интерфейсе Luxms BI. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца.

Медиана — это число, которое является серединой множества отсортированных чисел, то есть половина чисел имеют значения большие, чем медиана, а половина чисел имеют значения меньшие, чем медиана. Например, медианой для чисел 2, 3, 3, 5, 7 и 10 будет 4.

Примеры:

```
median(col1)
```

```
median(col1+col2)
```

Функция `median` не поддерживается для Vertica.

2.3.8 mode(expr) – Мода

Функция `mode` возвращает наиболее часто встречающееся (типичное) значение для выражения `expr`, вычисленного для каждой строки в кубе, при этом учитываются фильтры, которые могут быть заданы настройками (конфигурацией) или пользователем в интерфейсе Luxms BI. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца.

Примеры:

```
mode(col1)
```

```
mode(concat(col1, ':', col2))
```

Пример использования:

```
 measures: [
      'mode(quantity):mode',
      "mode(concat(package, ':', ' ', quantity, ' штук')):mode_",
    ],
```

Упаковка	mode	mode_
bottle	330	bottle: 330 штук

Рис. 2.2 **Пример применения функции mode**

Функция `mode` не поддерживается для SAP HANA.

Функция `mode` не поддерживается для Vertica.

2.3.9 min(expr) – Минимальное значение

Функция `min` возвращает минимальное значение для выражения `expr`, вычисленного для каждой строки в кубе, при этом учитываются фильтры, которые могут быть заданы настройками (конфигурацией) или пользователем в интерфейсе Luxms BI. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца.

Примеры:

```
min(col1)
```

```
min(col1+col2)
```

2.3.10 quantile(expr, q)

Функция `quantile` вычисляет квантиль для столбца или LPE выражения, указанного в `expr`. Значение `q` должно находится в диапазоне от 0 до 1.

Таблица 2.1 **Функции, на основе которых вычисляется `quantile` в разных СУБД**

СУБД	Функция
PostgreSQL	percentile_disc
Oracle	percentile_disc
Clickhouse	quantileExactLow

Примеры:

```
quantile(col1, 0.5)
quantile(col2, 0.75)
```

Функция `quantile` появилась в luxmsbi-pg 9.2.20

2.3.11 stddevSamp(expr) – Среднеквадратическое (стандартное) отклонение

Функция `stddevSamp` возвращает стандартное (среднеквадратическое) отклонение для выборки для выражения `expr`. `stddevSamp` — это квадратный корень из `varSamp`. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца.

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

Рис. 2.3 **Формула для вычисления stddevSamp**

Примеры:

```
stddevSamp(col1)
```

```
stddevSamp(col1+col2)
```

Функция `stddevSamp` появилась в luxmsbi-pg 9.2.3

2.3.12 stddevPop(expr) – Среднеквадратическое (стандартное) отклонение

Функция `stddevPop` возвращает стандартное (среднеквадратическое) отклонение в генеральной совокупности `expr`. `stddevPop` — это квадратный корень из `varPop`. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

Примеры:

```
stddevPop(col1)
```

```
stddevPop(col1+col2)
```

Функция `stddevPop` появилась в luxmsbi-pg 9.2.3

2.3.13 sum(expr) – Сумма значений

Функция `sum` возвращает сумму значений для выражения `expr`, вычисленного для каждой строки в кубе, при этом учитываются фильтры, которые могут быть заданы настройками (конфигурацией) или пользователем в интерфейсе Luxms BI. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца.

Примеры:

```
sum(col1)
```

```
sum(col1-col2)
```

2.3.14 total(expr) - Итог

Функция `total` вычисляет итоговое значение. В выражении `expr` должна быть хотя бы одна агрегатная функция.

> Если в выражении `expr` нет агрегатной функции, то выполнение запроса приведёт к ошибке!

Примеры:

```
sum(vat) / total(sum(vat))
sum(vat) / total(sum(vat) + sum(revenue))
sum(vat) / total(sum(vat+revenue))
sum(income) / total(count(id))
income / total(count(id))
```

Пример **ошибочного** выражения:

```
sum(vat) / total(vat + revenue) // => Ошибка!!!
```

> Функция `total` появилась в luxmsbi-pg 9.3.0

2.3.15 varPop(expr) - Дисперсия

Функция `varPop` возвращает стандартную дисперсию генеральной совокупности `expr`. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца.

Примеры:

```
varPop(col1)
```

```
varPop(col1+col2)
```

> Функция `varPop` появилась в luxmsbi-pg 9.2.3

2.3.16 varSamp(expr) – Дисперсия

Функция `varSamp` возвращает выборочную дисперсию `expr`. В качестве `expr` можно использовать LPE выражения, но чаще всего используется имя столбца.

Примеры:

```
varSamp(col1)
```

```
varSamp(col1+col2)
```

Функция `varSamp` появилась в luxmsbi-pg 9.2.3

2.3.17 uniq(col) - Уникальные значения

Функция `uniq` подсчитывает количество уникальных значений в столбце `col`. При этом значения `NULL` обрабатываются как одинаковые значения, то есть считается, что `NULL == NULL`.

Пример:

```
uniq(col1)   // => count(distinct(col1))
```

Функция `uniq(col)` транслируется в SQL выражение `count(distinct(col))` и является короткой записью для этого выражения. Однако, для СУБД Clickhouse можно управлять трансляцией с помощью настроек куба. Для этого к поле `config` куба необходимо поместить ключ `count_distinct` значением которого нужно указать настоящее имя функции, которая будет использоваться при трансляции. Например: `uniq`, `UniqExact` или `uniqHLL12`.

Пример настроек куба:

```
{"count_distinct": "uniq"}
```

При этом выражение `uniq(col)`, отправленное в этот куб, транслируется в SQL `uniq(col)`, а не в стандартный `count(distinct(col))`

Функция `uniq` появилась в luxmsbi-pg 9.3.0

2.4 Строковые функции

Строковые функции LPE работают со строками, которые хранятся в СУБД в формате UTF-8.

2.4.1 concat(str_0, str_1, ... , str_n) – Конкатенация

Функция `concat` “склеивает” все свои аргументы в одну строку.

Пример:

```
concat('{', col1, '}')
```

Результат в виде строки:

```
{col1}
```

2.4.2 concatWithSeparator(sep, str_1, str_2, ... , str_n) – Конкатенация с разделителем

Функция `concatWithSeparator` “склеивает” все свои аргументы в одну строку, используя значение параметра `sep` в качестве разделителя.

Пример:

```
concat(',', 'col0', col1, 'col2')
```

Результат в виде строки:

```
col0,col1,col2
```

> Функция `concatWithSeparator` поддерживается только для Clickhouse и PostgreSQL.

2.4.3 ilike(str,pattern) – Проверка соответствия

Функция `ilike` проверяет соответствие строки `str` шаблону `pattern` без учёта регистра символов и возвращает логическое значение (true или false). Функция `ilike` может использоваться в фильтрах в качестве условия. Шаблон `pattern` соотвтествует спецификациям SQL и поддерживает следующие подстановки:

- `%` любое количество (включая 0) любых символов;
- `_` один любой символ.

Примеры:

```
ilike('МоСква', '%сКв%') // true
```

2.4.4 initcap(str) – Верхний регистр у первой буквы и нижний регистр у остальных букв в каждом слове

Функция `initcap` переводит первую букву каждого слова в строке `str` в верхний регистр, а остальные — в нижний. Словами считаются последовательности алфавитно-цифровых символов, разделённые любыми другими символами.

Примеры:

```
initcap('москвА, роССия') // 'Москва Россия'
```

2.4.5 left(str, count) – Крайние левые символы из текстовой строки

Функция `left` возвращает указанное в `count` количество символов из строки `str`, начиная слева (с начала строки). Если в `count` указано отрицательное число, то справа отрезается указанное в `count` количество символов (значение `count` для подсчёта символов справа берётся по модулю).

Примеры:

```
left('asd',1) // 'a'
left('asd',-1) // 'as'
```

2.4.6 length(str) – Длина строки

Функция `length` возвращает длину строки (кол-во символов UTF-8).

Примеры:

```
length('Москва') // 6
```

2.4.7 like(str,pattern) – Проверка соответствия

Функция `like` проверяет соответствие строки `str` шаблону `pattern` и возвращает логическое значение (true или false). Функция `like` может использоваться в фильтрах в качестве условия. Шаблон `pattern` соотвтествует спецификациям SQL и поддерживает следующие подстановки:

- `%` любое количество (включая 0) любых символов;
- `_` один любой символ.

Примеры:

```
like('Москва', '%скв%') // true
```

2.4.8 right(str, count) – Крайние правые символы из текстовой строки

Функция `right` возвращает указанное в `count` количество символов из строки `str`, начиная справа (с конца строки). Если в `count` указано отрицательное число, то слева отрезается указанное в `count` количество символов (значение `count` для подсчёта символов слева берётся по модулю).

Примеры:

```
right('asd',1) // 'd'
right('asd',-1) // 'sd'
```

2.4.9 substring(str, offset, len) – Извлечение подстроки

Функция `substring` вырезает подстроку из строки `str` начиная со смещения `offset` и длиной в `len` символов. Смещение начинается с 1.

Примеры:

```
substring('sdf', 1, 1) // 's'
```

2.5 Математические функции

2.5.1 abs(number) – Абсолютное значение

Возвращает абсолютное значение заданного числа `number`. Как аргумент можно использовать LPE-выражения, которые возвращают число.

Пример:

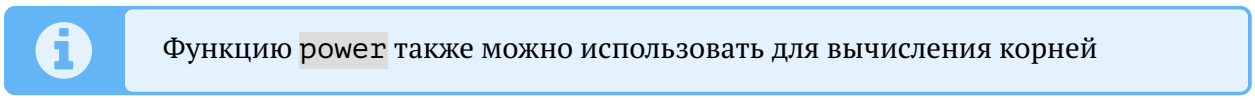
```
abs(col1+col2)
```

2.5.2 power(number, exp) – Возведение в степень

Функция `power` возводит число `number` в степень `exp`.

Пример возведения выражения в куб:

```
 power(col1+col2, 3)
```


Функцию `power` также можно использовать для вычисления корней

```
power(col1, 1/3)
power(col1, 0.5)
```

2.5.3 sqrt(number) – Квадратный корень

Возвращает квадратный корень из `number`.

Пример:

```
sqrt(4)
```

2.5.4 ceil(number) – Округление до большего целого

Функция `ceil` округляет число `number` до ближайшего большего целого. Таким образом, `ceil(12.3) = 13`, но `ceil(-12.3) = -12`

Пример:

```
ceil(col1+col2)
```

2.5.5 floor(number) – Округление до меньшего целого

Функция `floor` округляет число `number` до ближайшего меньшего целого. Таким образом, `floor(12.3) = 12`, но `floor(-12.3) = -13`

Пример:

```
floor(col1+col2)
```

2.5.6 round(number) – Округление до целого по правилам математики

Функция `round` округляет число `number` до целого по правилам математики.

Примеры:

```
round(col1+col2)

round(-12.6) // -13
round(12.6)  //  13
round(-12.2) // -12
round(12.2)  //  12
round(12.5)  //  13
round(-12.5) // -13
```

2.6 Логические функции и операторы сравнения

2.6.1 Функция between(col, interval)

Функция `between` с двумя аргументами принимает на вход столбец `col` и интервал `interval` и возвращает SQL условие between. В качестве аргумента `col` могут использоваться LPE-выражения.

2.6.2 Функция between(col, start, end)

Функция `between` с тремя аргументами принимает на вход столбец `col`, дату начала интервала `start`, дату окончания интервала `end` и возвращает SQL условие between. В качестве аргумента `col` могут использоваться LPE-выражения.

2.6.3 Операторы = и !=

Для сравнения значений на равенство можно использовать операторы `=` и `!=`. При этом, в отличие от операторов SQL, допускается сравнение со списком значений. В этом случае при конвертации в SQL будут использованы операторы `IN` и `NOT IN`.

Примеры:

```
col1 = [1,null,3]
v_rel_pp != [0,1,2,null]
```

При этом будут сгенерированы такие SQL выражения:

```
col1 IN (1,3) OR col1 IS NULL
v_rel_pp NOT IN (0, 1, 2) AND v_rel_pp IS NOT NULL
```

2.6.4 Операторы сравнения <, >, <=, >=

Для сравнения значений можно использовать операторы сравнения. Они работают так же, как и в других языках программирования.

> Операторы сравнения, как и операторы `=` и `!=`, редко используются сами по себе, но они часто используются в выражениях для функций `if`, `countIf` и подобных, где требуется сформулировать условие.

2.6.5 Оператор and

Выполняет логическое соединение двух выражений по условию логического И.

Пример:

```
(col1 + col2) > 0 and (col1/col2) < 1
```

2.6.6 if($cond_0$, $expr_0$, ... , $cond_n$, $expr_n$) – Функция `if` с четным числом аргументов

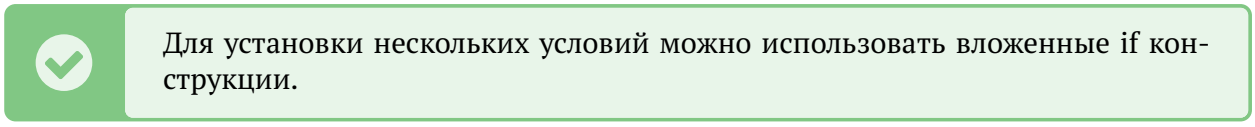
Функция `if` с чётным числом аргументов может иметь несколько условий `cond` (нечётные аргументы) и соответствующие им значения `expr` (чётные аргументы). При выполнении условия `cond` возвращается соответствующее значение `expr`, а если условие не выполнено, то функция `if` переходит к вычислению следующего условия. Функция `if` с чётным количеством аргументов работает как функция `IFS` в MS Excel.

2.6.7 if($cond_0$, $expr_0$, ... , $cond_n$, $expr_n$, other) – Функция `if` с нечетным числом аргументов

Функция `if` с нечётным числом аргументов работает так же, как и функция `if` с чётным числом аргументов. Однако, если ни одно из указанных условий `cond` не выполнено, то возвращается значение `other` — последний аргумент функции `if`.

Примеры:

```
if(min(price) >= 102, max(sales_amount), min(sales_amount)):ma
if(min(price) >= 102, if(min(sales_amount) < 10, max(sales_amount),
  min(sales_amount)), avg(sales_amount)):ma
```



Для установки нескольких условий можно использовать вложенные if конструкции.

2.6.8 Оператор `not`

Выполняет логическое отрицание.

Пример:

```
not col1 > 1 and not col1 < 0
```

2.6.9 Оператор `or`

Выполняет логическое соединение двух выражений по условию логического ИЛИ.

Пример:

```
(col1 + col2) > 0 or (col1/col2) < 1
```

2.6.10 Регулярные выражения. Операторы для поиска подстрок в SQL.**

- **Оператор** ~ используется для регистрозависимого поиска.

Пример:

```
measures: [
     "if(productname~'^[A,z]',productname):pr_n",
     'productname:productname',
   ],
```

В этом случае в столбике будут выведены productname, которые начинаются на ‘A’ и ‘z’, остальные ячейки останутся пустыми (заменятся на null). ^ указывает на начало строки.

```
filters: {
      '': [
        '~',
        [
          'column',
          'productname',
        ],
        '^[A-C,b]',
      ],
      productname: true,
    },
```

Здесь будут выводиться productname, которые начинаются на ‘A’, ‘B’, ‘C’ и ‘b’.

- **Оператор** `*` используется для нерегистрозависимого поиска.

```
filters: {
    '': [
      '~*',
      [
        'column',
        'productname',
      ],
      'k$',
    ],
    productname: true,
  },
```

В примере выше будут показаны productname, которые заканчиваются на ‘k’ и ‘K’. `$` указывает на конец строки.

- **Оператор** `!` используется для регистрозависимого поиска и учитывает результаты, которые не включают указанную подстроку.

```
filters: {
      '': [
        '!~',
        [
          'column',
          'country',
        ],
        'nited',
      ],
      productname: true,
    },
```

В примере выше в результате фильтрации не будут учитываться country, в которых в любом месте строки есть подстрока ‘nited’.

- **Оператор** `! *` используется для нерегистрозависимого поиска и учитывает результаты, которые не включают указанную подстроку.

```
filters: {
     '': [
       '!~*',
       [
         'column',
         'country',
       ],
       'n',
     ],
     productname: true,
   },
```

В примере выше в результате фильтрации не будут учитываться country, в которых в любом месте строки есть подстрока ‘n’, ‘N’.

2.7 Тригонометрические функции

2.7.1 sin(angle) – Синус

Вычисляет синус угла `angle` в радианах.

Примеры:

```
sin(3.1416):sin_180
sin(radians(30)):sin_30
```

Для перевода радиан в градусы используйте функцию `radians()`.

2.7.2 cos(angle) – Косинус

Вычисляет косинус угла `angle` в радианах.

Примеры:

```
cos(0.5236):cos_30
cos(radians(30)):cos_30
```

2.7.3 radians(degrees) – Преобразование градусов в радианы

Функция `radians` преобразует угловую меру `degrees`, выраженную в градусах, в радианы.

Примеры:

```
radians(45.0) // 0.785398163397448
```

2.7.4 tan(angle) – Тангенс

Вычисляет тангенс угла `angle` в радианах.

Примеры:

```
tan(0.7854):tan_45
tan(radians(45)):tan_45
```

2.7.5 pi() – Число Пи

Возвращает число Пи.

Пример:

```
pi()*2:grad_180
```

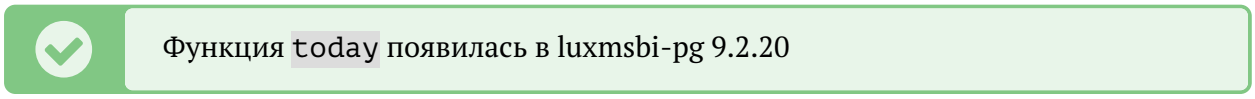
2.8 Календарные функции

2.8.1 today()

Функция `today` возвращает текущую календарную дату на момент вызова функции.

Примеры:

```
today()
```

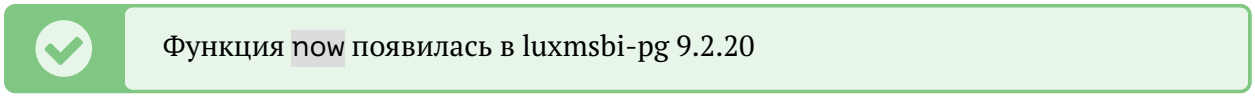

Функция `today` появилась в luxmsbi-pg 9.2.20

2.8.2 now()

Функция `now` возвращает текущее время на момент вызова функции.

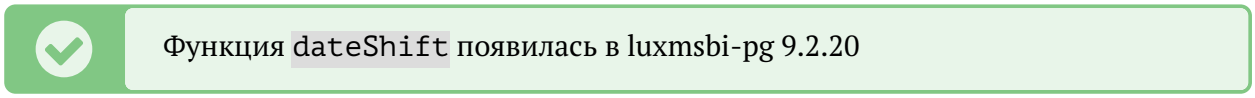
Примеры:

```
now()
```


Функция `now` появилась в luxmsbi-pg 9.2.20

2.8.3 Функция dateShift(delta, unit)

Функция `dateShift` с двумя аргументами работает так же, как и функция `dateShift` с тремя аргументами, но при этом, значение первого аргумента `dt` принимается равным `today()`.



Функция `dateShift` появилась в luxmsbi-pg 9.2.20

2.8.4 Функция dateShift(dt, delta, unit)

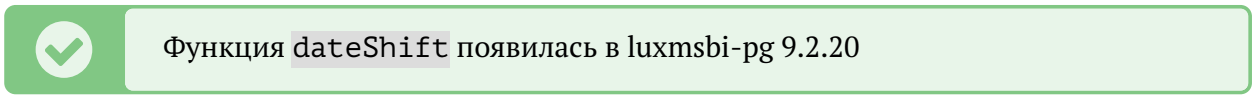
Функция `dateShift` вычисляет новую дату на основе исходной даты `dt`, смещения `delta` и типа смещения `unit`. Значение `delta` может быть как положительным целым, так и отрицательным целым числом. Положительные значения `delta` дают результирующую дату в будущем, а отрицательные - в прошлом, относительно исходной даты `dt`. Значения `unit` задаются строкой и могут быть равны:

- `d` или `day`
- `w` или `week`
- `m` или `month`
- `q` или `quarter`
- `y` или `year`

Значением аргумента `dt` может быть как дата, так и интервал. В случае интервала и начало и конец интервала получают одинаковое смещение и функция `dateShift` возвращает новый интервал.

Примеры:

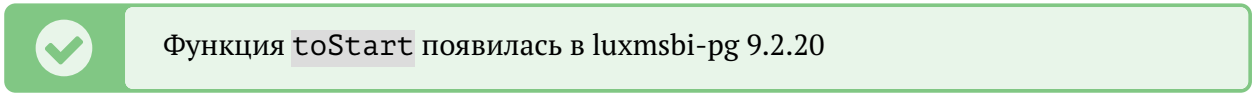
```
dateShift('2020-01-01', -2, m)       // 2019-11-01
dateShift('2020-01-01', -2, 'm')     // 2019-11-01
dateShift('2020-01-01', -2, month)   // 2019-11-01
dateShift('2020-01-01', -2, 'month') // 2019-11-01

dateShift('2020-01-01', 2, 'q')      // 2020-07-01
```



Функция `dateShift` появилась в luxmsbi-pg 9.2.20

2.8.5 Функция toStart(unit)

Функция `toStart` с одним аргументом работает так же, как и функция `toStart` с двумя аргументами, но при этом, значение первого аргумента `dt` принимается равным `today()`.



Функция `toStart` появилась в luxmsbi-pg 9.2.20

2.8.6 Функция toStart(dt, unit)

Функция `toStart` вычисляет новую дату на основе исходной даты `dt`. Функция возвращает дату начала интервала указанного с помощью аргумента `unit`, при этом сам интервал определяется на основе даты `dt`.

С помощью функции `toEnd` легко вычислять даты начала месяца, квартала, года.

Значения `unit` задаются строкой и могут быть равны:

- `d` или `day`
- `w` или `week`
- `m` или `month`
- `q` или `quarter`
- `y` или `year`

Примеры:

```
toStart(q) // возвращает начало текущего квартала
toStart('2024-09-23', 'y') // 2024-01-01 - начало года в который попадает 23 сен
  тября
```

Функция `toStart` появилась в luxmsbi-pg 9.2.20

2.8.7 Функция toEnd(unit)

Функция `toEnd` с одним аргументом работает так же, как и функция `toEnd` с двумя аргументами, но при этом, значение первого аргумента `dt` принимается равным `today()`.

Функция `toEnd` появилась в luxmsbi-pg 9.2.20

2.8.8 Функция toEnd(dt, unit)

Функция `toEnd` вычисляет новую дату на основе исходной даты `dt`. Функция возвращает дату окончания интервала указанного с помощью аргумента `unit`, при этом сам интервал определяется на основе даты `dt`.

С помощью функции `toEnd` легко вычислять даты конца месяца, квартала, года.

Значения `unit` задаются строкой и могут быть равны:

- `d` или `day`
- `w` или `week`
- `m` или `month`
- `q` или `quarter`
- `y` или `year`

Примеры:

```
toEnd(q) // возвращает конец текущего квартала
toEnd('2024-09-23', 'y') // 2024-12-31 - конец года в который попадает 23 сентяб
  ря
```

Функция `toEnd` появилась в luxmsbi-pg 9.2.20

2.8.9 Функция bound(unit)

Функция `bound` с одним аргументом работает так же, как и функция `bound` с двумя аргументами, но при этом, значение первого аргумента `dt` принимается равным `today()`.

Функция `bound` появилась в luxmsbi-pg 9.2.20

2.8.10 Функция bound(dt, unit)

Функция `bound` возвращает *интервал*, вычисленный на основе исходной даты `dt`, длительность которого соответствует аргументу `unit`, при этом дата `dt` входит в возвращаемый *интервал*.

Значения `unit` задаются строкой и могут быть равны:

- `d` или `day`
- `w` или `week`
- `m` или `month`
- `q` или `quarter`
- `y` или `year`

Примеры:

```
bound(q) // возвращает интервал для текущего квартала
bound('2024-09-23', 'y') // ['2024-01-01', '2024-12-31']
```

Функция `bound` появилась в luxmsbi-pg 9.2.20

2.8.11 Функция extend(delta, unit)

Функция `extend` с двумя аргументами работает так же, как и функция `extend` с тремя аргументами, но при этом, значение первого аргумента `dt` принимается равным `today()`.

Функция `extend` появилась в luxmsbi-pg 9.2.20

2.8.12 Функция extend(dt, delta, unit)

Функция `extend` возвращает *интервал*, начало которого равно дате `dt`, а конец вычисляется относительно даты `dt` как смещение `delta` измеренное в еденицах `unit`. Смещение может быть как положительным, так и отрицательным.

Значения `unit` задаются строкой и могут быть равны:

- `d` или `day`
- `w` или `week`
- `m` или `month`
- `q` или `quarter`
- `y` или `year`

Первый параметр `dt` может быть интервалом, в этом случае функция `extend` оставит начало интервала без изменений, а окончание интервала вычислит как указанное смещение относительно начала интервала.

Примеры:

```
extend(1, q) // возвращает интервал с началом сегодня и окончанием через 1 квартал от сегодняшней даты.

extend('2024-09-23', 1, 'y') // ['2024-09-23', '2025-09-23']

extend('2024-09-23', 300, 'day') // ['2024-09-23', '2025-07-20']
```

Функция `extend` появилась в luxmsbi-pg 9.2.20

2.8.13 Функция doty(dt)

Функция `doty` возвращает день года в виде целого числа. Название функции является сокращением от *Day Of The Year*. Параметр `dt` является датой или именем столбца типа дата.

Примеры:

```
doty('2024-01-01') // 1
doty('2024-05-09') // 130
```

Функция `doty` появилась в luxmsbi-pg 9.2.20

2.8.14 Функция woty(dt)

Функция `woty` возвращает неделю года в виде целого числа. Название функции является сокращением от *Week Of The Year*. Параметр `dt` является датой или именем столбца типа дата.

Примеры:

```
woty('2024-01-01') // 1
woty('2021-01-01') // 53
woty('2024-05-09') // 19
```

Функция `woty` появилась в luxmsbi-pg 9.2.20

2.8.15 Функция moty(dt)

Функция `moty` возвращает месяц года в виде целого числа. Название функции является сокращением от *Month Of The Year*. Параметр `dt` является датой или именем столбца типа дата.

Примеры:

```
moty('2024-01-01') // 1
moty('2024-05-09') // 5
```

Функция `moty` появилась в luxmsbi-pg 9.2.20

2.8.16 Функция qoty(dt)

Функция `qoty` возвращает квартал года в виде целого числа. Название функции является сокращением от *Quarter Of The Year*. Параметр `dt` является датой или именем столбца типа дата.

Примеры:

```
qoty('2024-01-01') // 1
qoty('2024-12-09') // 4
```

Функция `qoty` появилась в luxmsbi-pg 9.2.20

2.8.17 Функция hoty(dt)

Функция `hoty` возвращает полугодие года в виде целого числа. Название функции является сокращением от *Half a Year Of The Year*. Параметр `dt` является датой или именем столбца типа дата.

Примеры:

```
hoty('2024-01-01') // 1
hoty('2024-08-09') // 2
```

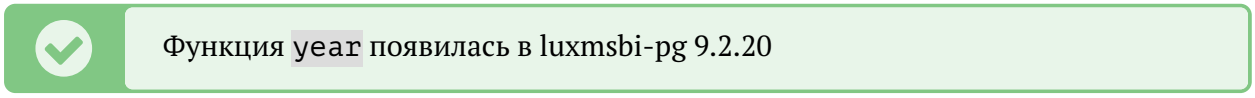
Функция `hoty` появилась в luxmsbi-pg 9.2.20

2.8.18 Функция year(dt)

Функция `year` возвращает год в виде целого числа. Параметр `dt` является датой или именем столбца типа дата.

Примеры:

```
year('2024-01-01') // 2024
year('2023-05-09') // 2023
```



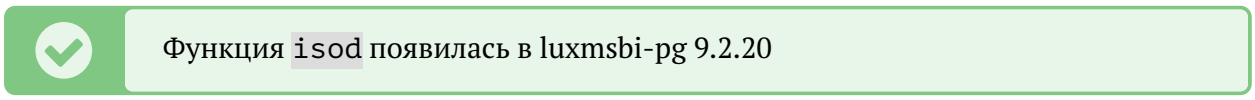
Функция `year` появилась в luxmsbi-pg 9.2.20

2.8.19 Функция isod(dt)

Функция `isod` возвращает день года по стандарту ISO 8601 в виде строки. Параметр `dt` является датой или именем столбца типа дата. Ответ содержит и год, и день в этом году.

Примеры:

```
isod('2024-01-01') // 2024-001
isod('2024-05-09') // 2024-130
```



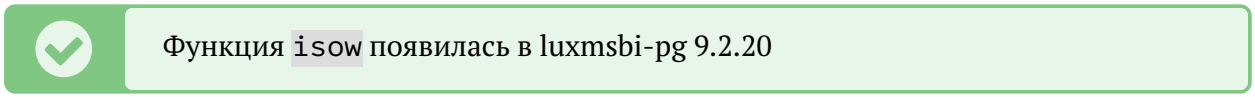
Функция `isod` появилась в luxmsbi-pg 9.2.20

2.8.20 Функция isow(dt)

Функция `isow` возвращает неделю года по стандарту ISO 8601 в виде строки. Параметр `dt` является датой или именем столбца типа дата. Ответ содержит и год, и неделю в этом году.

Примеры:

```
isow('2024-01-01') // 2024-W01
isow('2021-01-01') // 2020-W53
isow('2024-05-09') // 2024-W19
```



Функция `isow` появилась в luxmsbi-pg 9.2.20

2.8.21 Функция isoq(dt)

Функция `isoq` возвращает квартал года по стандарту ISO 8601 в виде строки. Параметр `dt` является датой или именем столбца типа дата. Ответ содержит и год, и квартал в этом году.

Примеры:

```
isoq('2024-01-01') // 2024-Q1
isoq('2024-05-09') // 2024-Q2
```

Функция `isoq` появилась в luxmsbi-pg 9.2.20

2.8.22 Функция isom(dt)

Функция `isom` возвращает месяц года по стандарту ISO 8601 в виде строки. Параметр `dt` является датой или именем столбца типа дата. Ответ содержит и год, и месяц в этом году.

Примеры:

```
isom('2024-01-01') // 2024-01
isom('2024-05-09') // 2024-05
```

Функция `isom` появилась в luxmsbi-pg 9.2.20

2.8.23 Функция isoy(dt)

Функция `isoy` возвращает год по стандарту ISO 8601 в виде строки. Параметр `dt` является датой или именем столбца типа дата.

Примеры:

```
isoy('2024-01-01') // 2024
isoy('2023-05-09') // 2023
```

Функция `isoy` появилась в luxmsbi-pg 9.2.20

2.9 Оконные функции

Оконные функции вычисляются аналогично агрегатным, но не объединяют несколько записей в одну, сохраняя их независимость. Они позволяют получить дополнительную информацию об исходной выборке, например, вычислить нарастающий итог, скользящее среднее или ранжировать значения. Реализация оконных функций в LPE выполнена с использованием принципов функционального программирования, что даёт удобную и относительно компактную форму записи параметров.

Для использования оконных функций требуется обёртка `window(...)`, в аргументах которой указываются все необходимые параметры.

Примеры:

```
window(sum(c1)) // оконная функция - эквивалент sum(c1) OVER ()
window(sum(c1), partition(dt)) // sum(c1) OVER (partition by dt)
window(sum(c1), order(-dt)) // sum(c1) OVER (order by dt DESC)
```

2.9.1 Функция window(fn, partition, order)

- Функция `window` принимает обязательный первый аргумент - функцию `fn` и выполняет функцию `fn` в качестве оконной SQL функции.
- `fn(column_name)`/`fn()` в зависимости от типа функции может требовать аргумент или нет
- `partition(column_name)` определяет, как данные будут разделены на группы для выполнения оконных операций
- `order(column_name)` определяет порядок строк в каждой группе данных. Для корректной работы ранжирующих функций требуется указывать для них `order(column_name)`

Примеры:

Вызов стандартных агрегационных функций: sum(), count(), min(), max(), avg()

```
measures: [
     'window(min(price),partition(date)):w_sum',
     'window(max(price),partition(date)):w_sum',
     'window(sum(price),partition(date),order(price)):w_sum',
     'window(count(productname),partition(date),order(price)):w_count',
     'window(avg(price),partition(date),order(-price)):w_avg',
   ],
```

Оконная функция row_number() присваивает уникальный последовательный номер каждой строке в рамках определенной группы или окна. Этот номер обычно начинается с 1 и увеличивается на 1 для каждой последующей строки.

```
measures: [
     'window(row_number(),partition(country),order(date)):row_number'
     'window(row_number(),order(date)):row_number'
  ],
```

Оконная функция rank() также присваивает уникальный номер каждой строке в рамках определенной группы или окна, но если есть несколько строк с одинаковым значением по порядку сортировки (например, одинаковая зарплата), то функция RANK() присваивает им один и тот же ранг, пропуская следующий ранг. Например, если двум сотрудникам присвоен ранг 1, следующий сотрудник получит ранг 3, а не 2.

```
measures: [
     'window(rank(),partition(country,date),order(date,productname)):rank',
     'window(rank(),order(date,productname)):rank',
  ],
```

Оконная функция dense_rank() возвращает ранг каждой строки в секции результирующего набора без промежутков в значениях ранжирования. Ранг определенной строки равен

количеству различных значений рангов, предшествующих строке, увеличенному на единицу. В этом случае, если двум сотрудникам присвоен ранг 1, следующий сотрудник получит ранг 2.

```
measures: [
      'window(dense_rank(),partition(country,date),order(date,productname)):rank',
      'window(dense_rank(),order(date,productname)):rank',
  ],
```

Оконная функция ntile(cnt) используется для разделения упорядоченного набора строк на указанное количество равных групп (частей). Функция NTILE принимает один аргумент - количество групп, на которые нужно разделить строки, и присваивает каждой строке номер группы, к которой она относится.

```
measures: [
      'window(ntile(4),partition(country,date),order(date,productname)):ntile_4',
      'window(ntile(4),order(date,productname)):ntile_4',
  ],
```

Пример вычисления нарастающего итога:

```
measures: [
      'window(sum(amount),order(date)):cumulative_sum',
  ],
```

Функция `window` появилась в luxmsbi-pg 9.3.0

2.10 Геометрические и картографические функции

2.10.1 pointInCircle – Проверка вхождения точки в окружность

Функция `pointInCircle` определяет, входит ли точка с указанными координатами в окружность.

Функция `pointInCircle` поддерживается только для Clickhouse и PostgreSQL.

2.10.2 pointInEllipses(x, y, x_0, y_0, a_0, b_0, ... ,x_n, y_n, a_n, b_n) – Проверка вхождения точки в указанные эллипсы

Функция `pointInEllipses` определяет, входит ли точка с указанными координатами хотя бы в один из указанных эллипсов.

Функция `pointInEllipses` поддерживается только для Clickhouse.

2.10.3 pointInPolygon – Проверка вхождения точки в полигон

Функция `pointInCircle` определяет входит ли точка с указанными координатами в указанный полигон.

Функция `pointInPolygon` поддерживается только для Clickhouse и PostgreSQL.

2.11 Функции lpe

Иногда возникает необходимость вычислить какое-либо выражение и результат вычисления использовать при построении SQL запроса. В этом случае возникают неоднозначности, так как в LPE есть своя реализация функций `sum()`, `min()`, `count()` и других. Чтобы разрешить эту неопределённость, используйте функцию `lpe()`, которая вычисляет свой аргумент в контексте LPE и возвращает результат вычисления.

2.11.1 lpe(expr)

Функция `lpe` вычисляет свой аргумент `expr` в контексте LPE. Это позволяет выполнить вычисления до генерации SQL запроса и подставлять вычисленные значения в нужное место в SQL запрос.

Пример:

```
lpe(rand())
```

При каждом запросе будет вычислятся новое случайное значение, и оно будет подставлено в SQL.

Функция `lpe` появилась в luxmsbi-pg 9.3.4

2.11.2 ql(arg)

Функция `ql` экранирует свой аргумент как текстовое значение SQL, то есть берёт значение в одинарные кавычки. Одинарные кавычки внутри аргумента `arg` экранируются по правилам SQL, то есть удваиваются.

Пример:

```
// переменная a ссылается на строку quote'me

ql(a) // 'quote''me'
```

```
col1 = ql(a) // col1 = 'quote''me'
```

2.11.3 get_in(key, p_0, ... p_n)

Функция `get_in` возвращает различные значения, доступные в момент запроса. Аргумент `key` задаёт имя объекта, в котором через остальные аргументы можно найти значение. Аргументы p_0, ... p_n являются именами ключей в объектах или индексами массива для навигации в сложной структуре JSON.

Начиная с версии 9.2.12 доступны следующие ключи `key`:

- `user`: объект с информацией о текущем пользователе Luxms BI:
  - `id`: id пользователя (int)
  - `username`: имя пользователя (text)
  - `email`: email полльзователя (text)
  - `sys_config`: JSON объект с информацией из IAM систем (json)

Начиная с версии 9.3.5 доступны следующие ключи `key`:

- `koob`
  - `query`: объект с информацией о запросе клиента Luxms BI через API
    * `columns`: список столбцов
    * `filters`: хэш фильтров
    * `sort`: список сортировок
    * `with`: имя куба

Примеры:

Предположим, что Luxms BI имеет такую информацию о пользователе:

```
{
 "id": 123,
 "username": "user123",
 "email": "user@luxms.bi",
 "sys_config": {
   "ext_domain": "customer.local",
    "ext_groups": ["g1", "g2", "g3"],
    "customval": "key1;key2;key3"
 }
}
```

```
get_in('user', 'sys_config', 'ext_domain') // возвращает текст customer.local
```

Если вы хотите дополнительно обработать значения, возвращаемые get_in, перед отправкой их в SQL, используйте функцию `lpe()`, чтобы стали доступны функци обработки строк, математические операторы и т.д.

```
column = ANY( lpe(get_in('user', 'sys_config', 'customval').split(';').map(ql) ))

/* получится column = ANY('key1','key2','key3') */
```

Предположим, что koob запрос (объект `koob.query`) имеет такую структуру:

```
{
  "with": "mssql.orders_full_ytd",
  "columns": [
    "productname",
    "count(quantity):q"
  ],
  "filters": {
    "productname": [
      "=",
      "Детская одежда",
      "Женская обувь"
    ]
  },
  "limit": 128,
  "sort": [
    "+productname"
  ]
}
```

Также предположим, что в столбце `productname` хранятся имена товаров через запятую. Тогда мы могли бы написать вот такое выражение для фильтров:

```
lpe( let( [foo, get_in(koob,'query','filters','categoryname')],
    str( if(foo.0 = '!=' and foo.count() > 1 , 'not ' , ''),
         match(productname, foo.slice(1).join('|').ql()  ))))
```

Так как в lpe нет функции `match`, то она не будет выполняться в LPE, а перейдёт в SQL запрос. Функции `slice`, `join`, `ql`, `count`, `if`, `get_in`, `let` выполнятся в LPE.

2.11.4 let(names, expr)

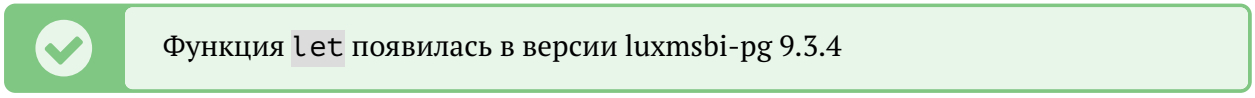
Функция `let` выполняет выражение `expr`, при этом выражению `expr` доступны переменные, объявленные в списке `names`.

Примеры:

```
let([a,2], a+a) // 4
```

Если нужно объявить несколько переменных, то каждое объявление делается в отдельном списке:

```
let([[a,2],[b,8]], a+b) // 10
```



Функция `let` появилась в версии luxmsbi-pg 9.3.4

3 Использование LPE в кубе

В кубе можно использовать выражение`${filters}`. Оно используется для вставки пользовательских фильтров из управляющего дэша или `onClickDataPoint` в тело запроса.

> Чтобы перечисленные ниже функции отрабатывали в кубе, нужно в конфиг куба вставить выражение `is_template: 1`.

3.1 Функция filters()

Синтаксис:

```
<...>
where ${filters()}
```

Это выражение подставляет в куб условия, пришедшие из запроса дэша. Запрос дэша в Luxms BI представляет собой JSON объект следующего формата:

```
{
  "with":"pokaz22.pokaz22",
  "columns":["test","name","sum(v_main):smv"],
  "filters":{
    "test":["=","Доктора наук+Профессор"],
    "name":["=","Женщины","Мужчины"]
  }
}
```

В поле `columns` перечисляются measures и dimensions, запрашиваемые дэшем. Поле `filters` содержит объект, где ключами являются наименования dimensions, а их значениями - массивы. В этих массивах первым элементом идёт знак сравнения, а последующими - перечень значений.

На основе приведённого выше JSON объекта выражение `${filters()}` составит следующий запрос:

```
SELECT
 sum(v_main) as smv,
  degree as test,
  name as name
FROM
 (
   <куб>
```

```
    WHERE degree = 'Доктора наук+Профессор'
    AND name in ('Женщины','Мужчины')
   ) as pokaz22
```

Обратите внимание, что в случае, когда в фильтрах размерности выбрано несколько значений, автоматически используется условие `in`.

Вы можете увидеть получившийся запрос, добавив параметр `?meta` к ссылке запроса дэша, скопированного в виде cURL.

3.1.1 Параметры функции filters()

В качестве параметров этой функции могут перечисляться названия одного или нескольких полей.

Необходимо использовать названия полей из куба, а не из источника

Синтаксис:

```
<запрос>
WHERE ${filters([column1, column2, ...])}
```

Перечисление полей определяет, какие именно фильтры передаются в куб. Написав `${filters(name, test)}`, мы получим результат, аналогичный тому, что описан выше.

Приведём пример с JSON объектом, описанным выше:

Фрагмент куба:

```
<...>
WHERE ${filters(name)}
```

Запрос:

```
SELECT
 sum(v_main) as smv,
  degree as test,
  name as name
FROM
 (
   <куб>
    WHERE name in ('Женщины','Мужчины')
   ) as pokaz22
```

3.1.2 Функция except()

Синтаксис:

```
<запрос>
WHERE ${filters(except(<column1> [, column2, ...]))}
```

Данная функция в качестве параметров принимает одно или несколько ID полей в кубе. Перечисленные поля исключаются из перечня подставляемых в запрос.

То есть, если в случае испольования функции `filters()` мы рассчитываем, что в запрос **подставятся только выбранные значения**, то функция `except()` **исключит из запроса** все перечисленные в ней значения.

4 Книга Рецептов LPE

В данном разделе описаны примеры использования LPE-выражений для вычисления различных фактов в разделе “Размерности” виджета.

4.1 Математические функции

Математические функции могут использоваться как для работы с единичными значениями, так и со столбцами. Например, ниже показано вычисление EBIDA-показателя.

```
sum(revenue_without_nds)-sum(price_without_nds*sales_amount):ebida
```

Здесь вычитание применяется к единичным значениям, а умножение к столбцам.

В тригонометрических функциях по умолчанию используются радианы, а не углы, однако, специальные функции позволяют преобразовывать радианы и градусы между собой. В примере ниже вычисляется расстояние от Санкт-Петербурга до различных объектов при помощи их координат.

```
acos(sin(radians(59.9386))*sin(radians(latitude)) + ↩
  cos(radians(59.9386))*cos(radians(latitude)) * cos(radians(30.3141 - ↩
  (longitude)))) * 6371:distance
```

Ниже приведен пример использования функции `power()` при расчёте сложных процентов.

```
deposit*power(1+percents/100, years):summa
```

Эту же формулу можно использовать для вычисления количества лет, через которые сумма перевалит через определенный порог

```
log(1+percents/100, 2):years_num
```

При расчёте величин, которые могут оказаться не целыми числами рекомендуем использовать функцию `round()` и её аналоги (`floor()` и `ceiling()`).

Важно заметить, что в интерфейсе Luxms BI есть другой инструмент для округления чисел - в настройках виджета вы можете задать формат числа (снимок экрана ниже), если вы используете его, то в функции `round()` нет необходимости.

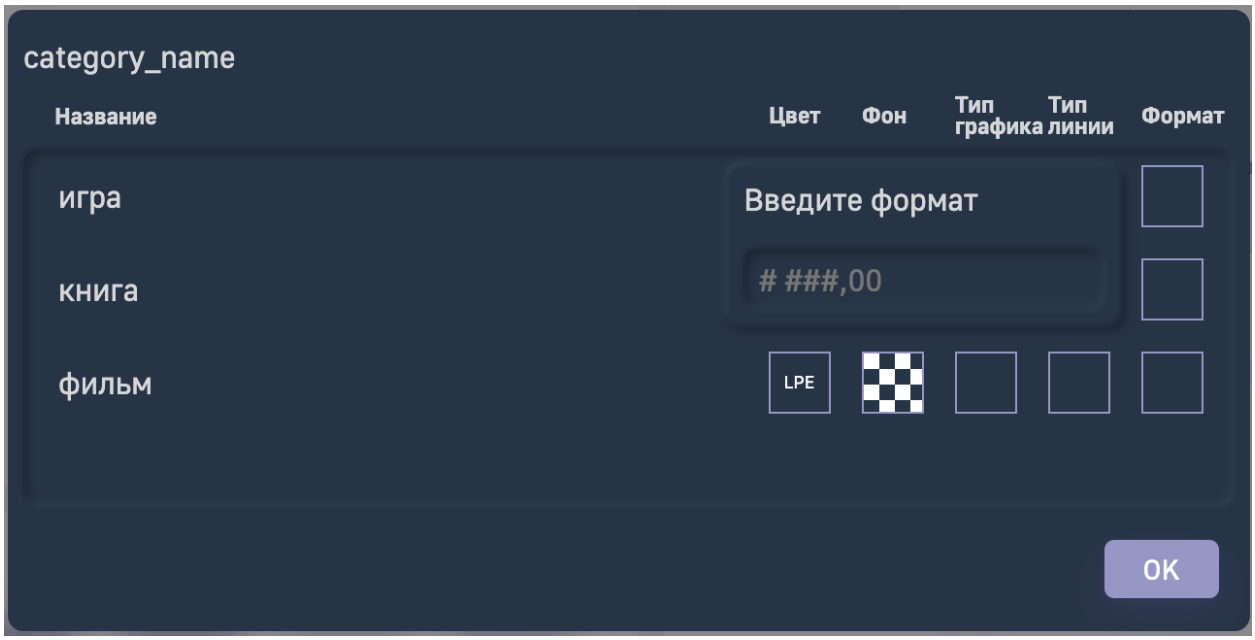


Рис. 4.1 **format_numbers.png**

Пример использования функции `round()` для округления результата деления:

```
round(sum(revenue_without_nds)/1000, 2):revenu
```

4.2 Строковые функции

Чаще всего строковые функции используются для создания или редактирования размерностей, а не фактов. Например, с помощью LPE можно создать столбец “ФИО” из отдельных “Имя”, “Отчество”, “Фамилия”:

```
concat(surname, name, fathername):full_fio
```

Или убрать из названия федеральных округов словосочетание “федеральный округ” для сокращения

```
trim(lower(fo_name), 'федеральный округ'):fo_name_short
```

4.3 Преобразование типов данных с помощью LPE

Функция `age()` вычисляет разницу между двумя датами и возвращает её в виде типа данных `interval`. Так как этот тип данных не отображается виджетами Luxms BI, необходимо преобразовать его.

```
to_char(age(now(), '2022-03-15'), 'Y MM DD HH24:MI:SS'):aa
```