ООО Авторапорт
ПО SDM

Инструкция по работе ПО SDM:

Направление запросов через DB Tools

Оглавление

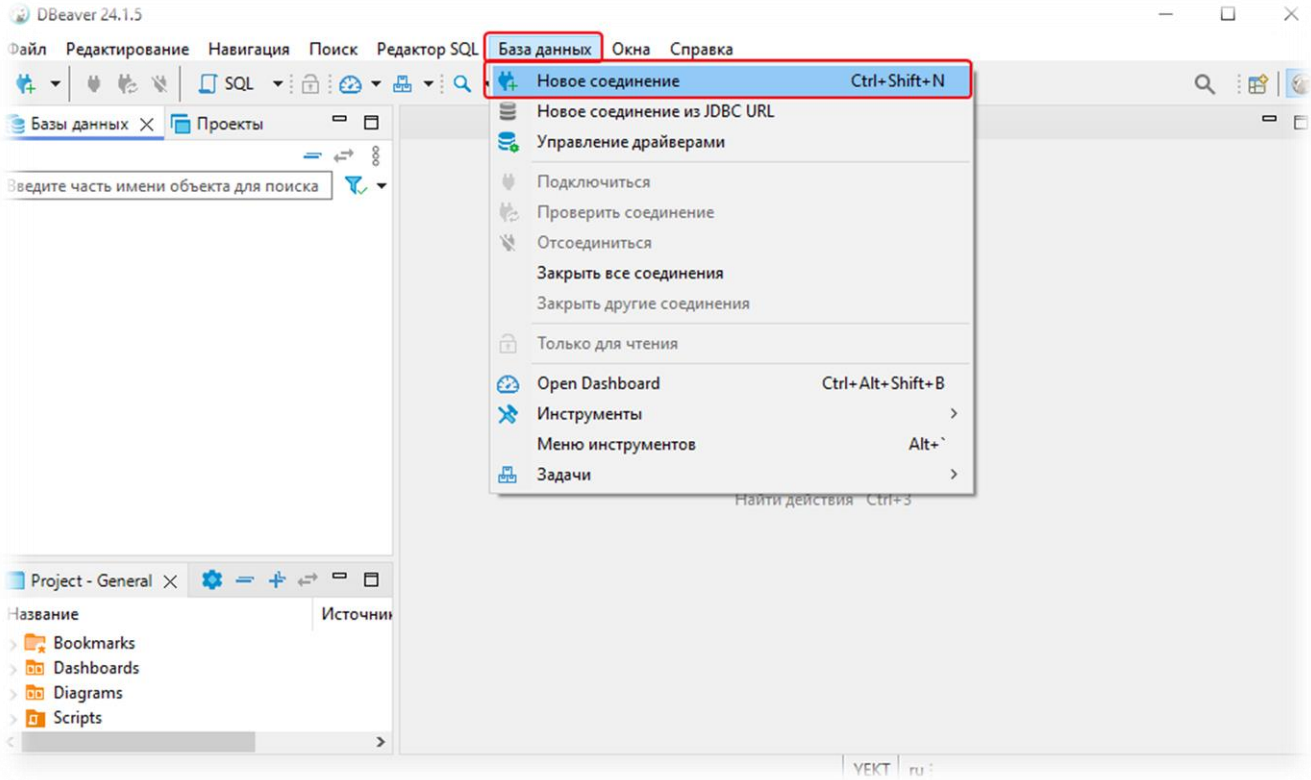
Подготовка к взаимодействию

Требования для взаимодействия:

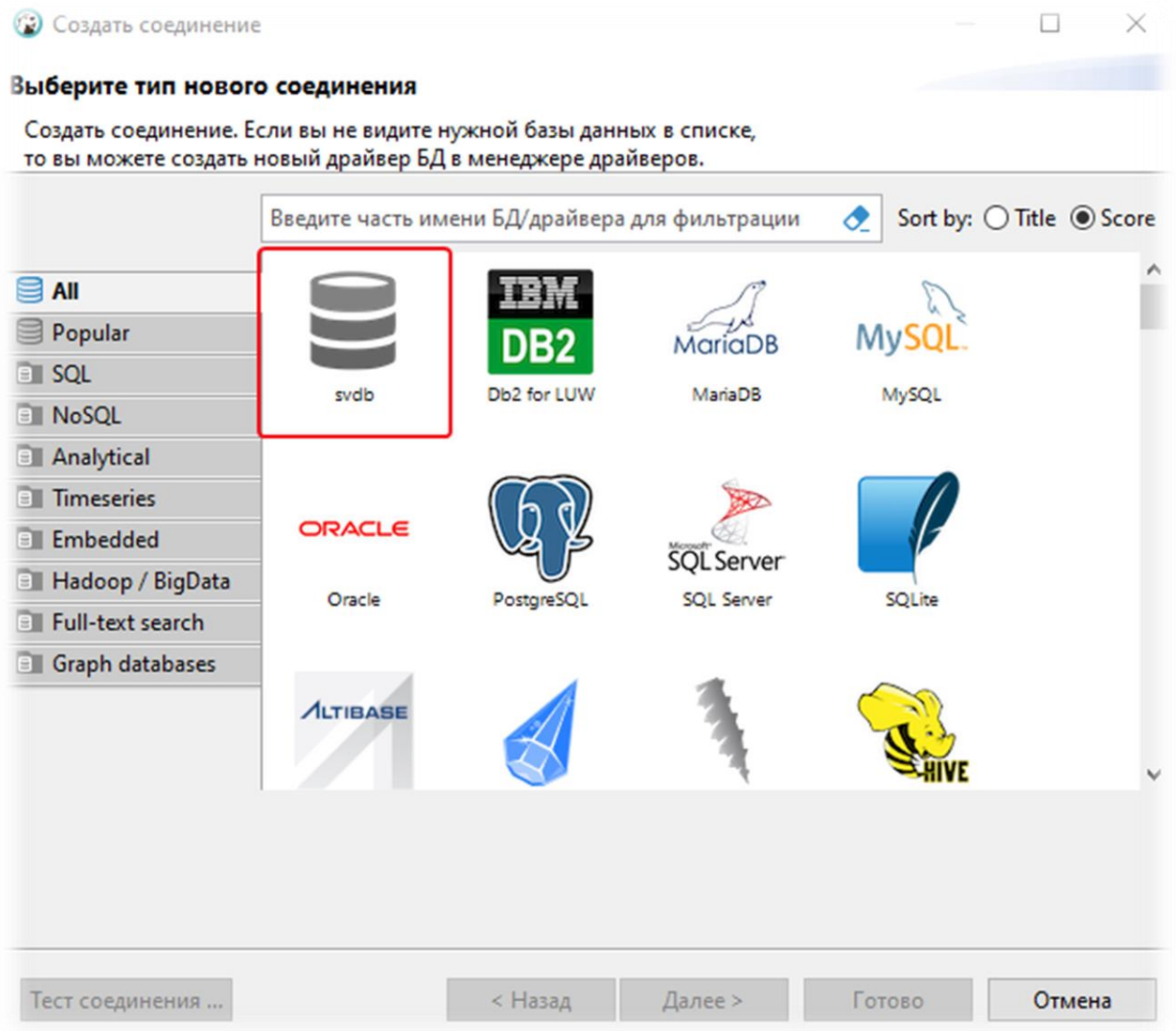
- наличие заключенного договора;
- наличие Username (имя_пользователя_keycloak);
- наличие Password (пароль_для авторизации_в_keycloack);
- наличие на компьютере DBeaver (универсального клиента БД);
- наличие драйвера для Spectrum Data Gate.

Порядок создания нового подключения с использованием уже настроенного драйвера

1. Выбрать в верхнем меню пункт «База данных» -> «Новое соединение»

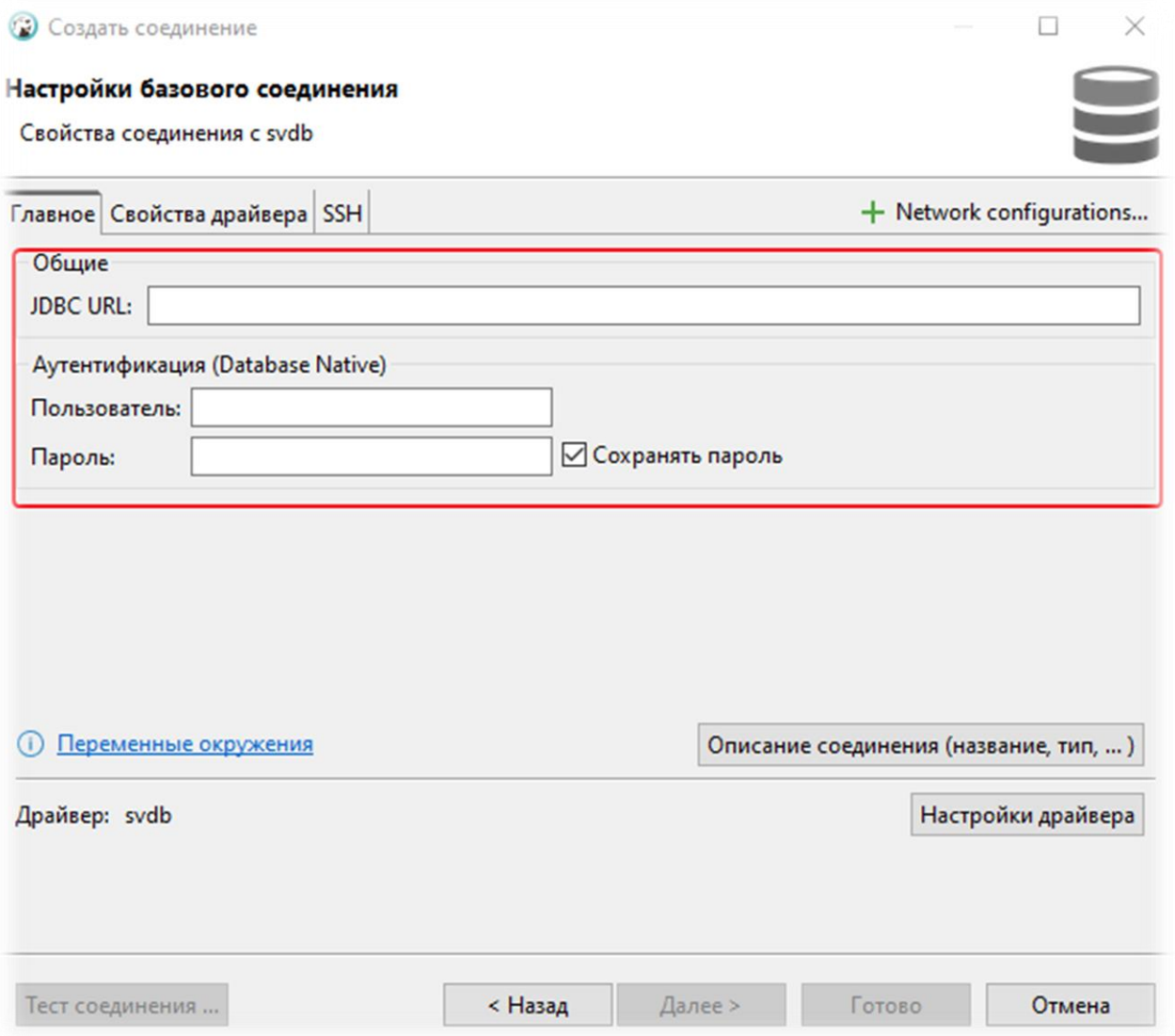


2. Выбрать тип нового соединения «svdb»

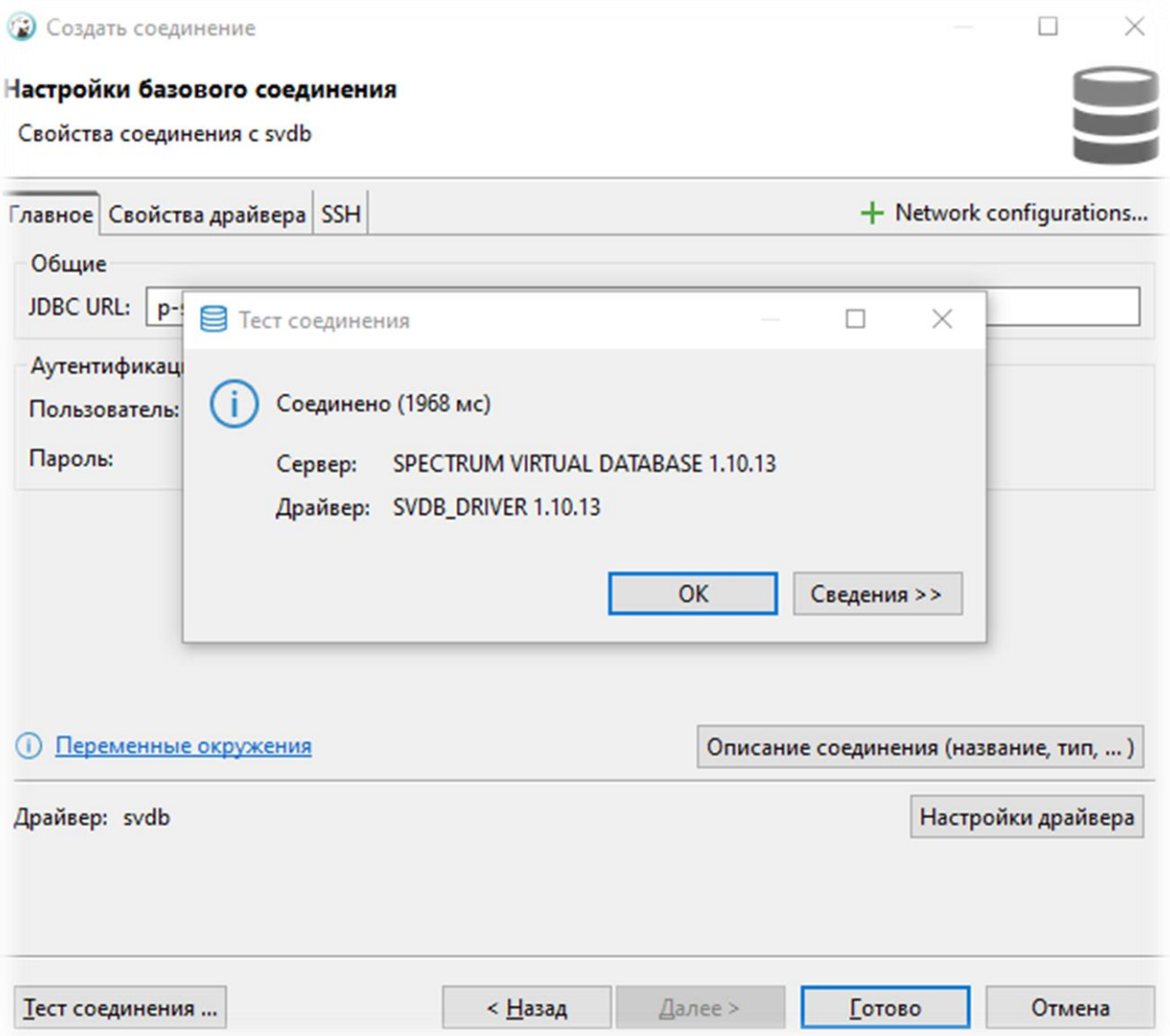


3. Нажать на кнопку «Далее»

4. Заполнить настройки базового соединения:
    a. JDBC URL: p-svdb-@8.spectrumdata.tech:443
    b. Пользователь: имя_пользователя_keycloak
    c. Пароль: пароль_ для авторизации_в_keycloack



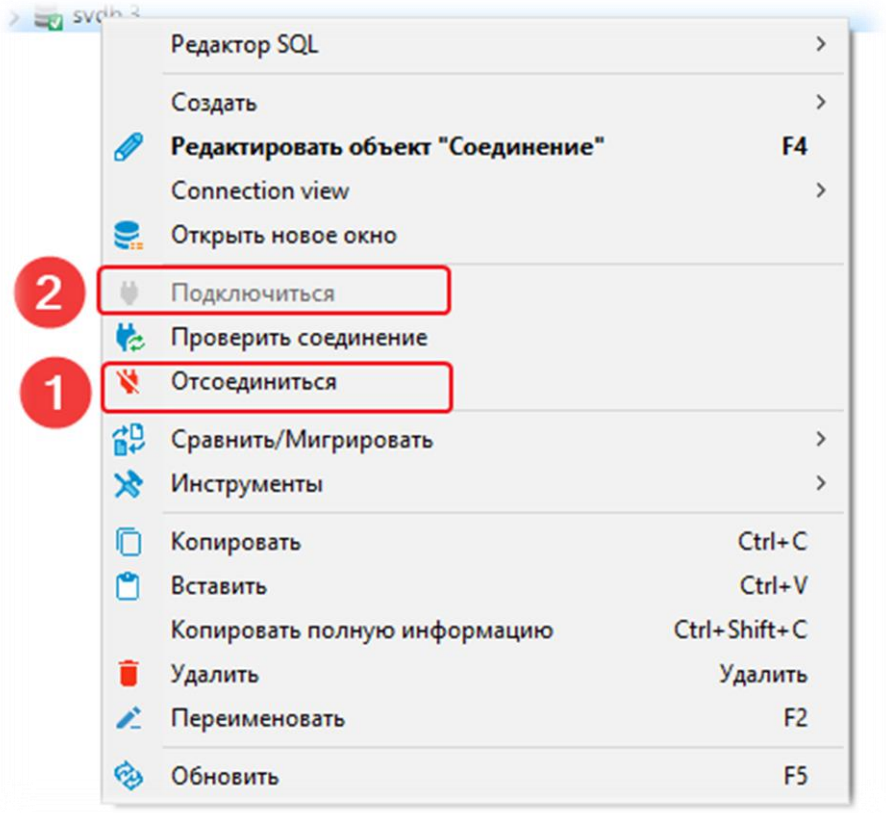
5. Нажать на кнопку "Тест соединения" для проверки подключения.
   Должно вернуться «Соединено», а так же версия драйвера.



6. Нажать на кнопку «Готово».

Порядок действий при смене прав доступа

1. Выбрать подключенную базу данных.
2. Правой кнопкой мыши вызвать окно, в котором выбрать действие «отсоединиться».
3. Правой кнопкой мыши вызвать окно, в котором выбрать действие «подключиться».



Синтаксис SDQL

Выражение на языке запросов SDQL может содержать следующие элементы:

```
[ PREPARE ]
[ EXPLAIN ]
SELECT
  [ DISTINCT ]
  [ * | <expression> [ [ AS ] <alias> ] [, ...] ]
  [ [ APPEND | REWRITE ] INTO [<target>]::<table>  ]
  [ FROM <table | subquery> [ [ AS ] <alias> ] ]
  [ WHERE <expression> ]
  [ GROUP BY <expression> [, ...]  ]
  [ HAVING <expression> ]
  [ EXPAND BY <expression> [, ...]  ]
  [ ORDER BY <expression> [ASC | DESC] [, ...] [NULLS (FIRST | LAST)] ]
  [ OFFSET <start> ]
  [ LIMIT <count> ]
  [ WITH <hint> [...] ]
  [ @P (p1, p2, ...) | (n1 = p1, n2 = p2,...) ]
  [ @F [F | A | N] [S <fetch-size:int>] [M <fetch-min-size:int>] [T
<fetch-time:int(ms)> ] ]
  [ @R <roles> ]
  [ @D <driver> ]

<condition> ::=
  [ true | false | <expression> [ AND <expression>] ]

<hint> ::=
  [
    LOCAL_NODE |
    SEQ_FULL_SCAN |
    RND_FULL_SCAN |
    EXPLICIT_FIELDS |
    PARALLEL [<n>]
  ]
INSERT INTO
  [ <table> ]
  [ (<expression> [, ...]) ]
  VALUES
  [ (<expression> [, ...]) ]
  [ @P (p1, p2, ...) | (n1 = p1, n2 = p2,...) ]
  [ @F [F | A | N] [S <fetch-size:int>] [M <fetch-min-size:int>] [T
<fetch-time:int(ms)> ] ]
  [ @R <roles> ]
  [ @D <driver> ]
```

Для выполнения ранее сохраненных планов запросов

```
EXECUTE '<uid>' [ @P (p1, p2, ...) | (n1 = p1, n2 = p2,...) ]
```

Также для пользовательских таблиц, созданных по команде `SELECT ... INTO ...` имеется команда удаления:

```
DROP TABLE [<target>]::<table>
```

Порядок предложений в инструкции SELECT имеет значение. Любое из необязательных предложений может быть опущено; но если необязательные предложения используются, они должны следовать в определенном порядке.

Все ключевые слова SDQL в соответствии с ANSI SQL являются регистр независимыми, то есть `SELECT 1` это то же самое, что и `SelecT 1`

PREPARE

Инструкция PREPARE вместо выполнения запроса формирует и кэширует его план в сессии и возвращает его идентификаторов.

Затем этот сохраненный план может использоваться посредством вызова EXECUTE.

Основная задача PREPARE - кэшировать часто повторяемые запросы с параметрами для экономии времени и большей чистоты запросов (нет риска инъекций или неправильного синтаксиса)

EXPLAIN

Инструкция EXPLAIN позволяет вывести план выполнения запроса без его непосредственного исполнения.

Результат будет представлен в виде таблицы, каждая строка которой - это шаг выполнения, например:

description	id	parent	type
`SELECT: fld:[full_name<in:name>]`	1001	0	*operations.Select
`-> PAGING: offset 2, limit 10`	2002	1001	*operations.Paging
`-> PRIMARY SELECT: tbl:[dict.regions] -> SELECT fld:[name<out:full_name>] WHERE ep_number = '123' LIMIT 12`	3003	2002	*operations.PrimarySelect

Также план выполнения запроса выводится в серверный лог в виде дерева:

```
[SDQL0200](200)
--------- Query plan ---------
SELECT: fld:[full_name<in:name>]
┗═PAGING: offset 2, limit 10
  ┗═PRIMARY SELECT: tbl:[dict.regions] ->
    ╠┈┈┈SELECT fld:[name<out:full_name>]
    ╠┈┈┈  WHERE ep_number = '123'
    ┗┈┈┈  LIMIT 12
```

Пример использования

```
EXPLAIN SELECT * FROM dict.regions
```

SELECT

На текущий момент синтаксис SDQL поддерживает только один тип запроса - SELECT, который позволяет делать выборку одной или нескольких строк и столбцов из одной таблицы в Spectrum Data Gate.

`<expression>` - название выбираемого столбца в формате `[table.]field` или константа.

`<alias>` - псевдоним столбца. Определяет название столбца в результирующем наборе данных.

> *Псевдоним может быть русскоязычным и с пробелами при использовании в двойных кавычках, например SELECT id as "Мой ИД"*

При анализе запроса - SDQL следит за тем, чтобы не было дубликатов итоговых имен полей в возвращаемых данных.

Это поведение **отличается** от ANSI SQL баз.

Так, в Spectrum Data Gate нельзя выполнить запрос вида `select x,x,x from my.table` или `select a as x, b as x from my.table` - так как при таких запросах появляется несколько полей с одинаковым итоговым именем.

Это сделано для исключения двусмысленности при трактовке данных и их маппинга в структуры типа JSON.

Из-за этого требования **значение символа * также отличается от ANSI SQL**. Обычно это означает “все поля из таблицы”, в случае Spectrum Data Gate это означает “все поля, которых еще нет в списке выбранных”.

Таким образом, если в таблице x есть поля a,b,c,d,e, то выборка `select *, *, * from x` вернет не 3 комплекта полей, а только один.

В случае выборки `select e, d, *, a from x` в итоге вернется e,d,b,c,a, то есть поля, явно указанные по именам окажутся на своих местах, а остальные не включенные поля добавятся вместо *.

Учитываются именно **итоговые имена**, если сделать запрос `select a as b, b as c, * from x`, то вернутся поля b (реальный контент из a), c (реальный контент из b), a, d, e (то есть поле a будет также возвращено, так как поля с таким именем пока не было в SELECT)

Пример использования

```
SELECT name AS region, 'constant' AS cnst FROM dict.regions

SELECT name, type, * FROM dict.regions -- поля name, type переместятся
вперед
```

Начиная с версии 0.5.0 в SELECT можно использовать связанные подзапросы из других таблиц

```
select x.a, x.b, (select fields c from t where d = x.a)
from t2 x
with parallel 6
```

DISTINCT

Указывается перед списком полей в SELECT, чтобы указать, что нужны только уникальные (различные) записи в ответе. Полные дубликаты будут удалены.

```
SELECT DISTINCT region_code FROM (SELECT region_code FROM fssp.direct
LIMIT 50 WITH RND_FULL_SCAN)
```

INTO

Позволяет создавать таблицы из результатов запросов и хранить их в связанном с Spectrum Data Gate хранилищем (сейчас используется Postgres)

Имя таблицы целевой обязательно включает в себя `::`, что означает “внешнюю, временную таблицу”, а также может включать в себя указание целевого хранилища.

Сейчас к системе подключено два хранилища:

1. `session` в памяти объемом до 10000 записей, хранящиеся только на время сессии, например `session::x`, при хранении в сессии, таблицы **не должны** иметь схем в названии `session::a.x` - нельзя
2. `default` в подключенном Postgres, так как это умолчание, можно опускать название `default::my.x` это то же самое, что `::my.x`, таблицы **обязаны** иметь схему в названии, например `::a.x`, `::my.y`, имя схемы `my` - особое - оно создает таблицы в специальном

личном пространстве базы данных, доступном только текущему пользователю, при использовании любой другой схемы - таблица создается в общем пространстве

Поддерживается несколько режимов команды

3. обычный `SELECT 1 as i INTO ::my.x` - если таблицы нет - создает и записывает в нее результаты запроса, если уже есть - то ошибка
4. добавление `SELECT 1 as i APPEND INTO ::my.x` - если таблицы нет, то создает, если есть, то дописывает данные в таблицу
5. пересоздание `SELECT 1 as i REWRITE INTO ::my.x` - сначала уничтожает имеющуюся таблицу, затем заново создает и заполняет

Сам запрос вернет журнал как происходила загрузка, чтобы получить загруженные данные надо выполнить SELECT из созданной таблицы, например `select * from ::my.x`

Ограничение! SELECT INTO не может использоваться в подзапросах, так нельзя `select * from (select 1 into ::my.x)`, потому что не до конца понятна семантика такого вызова

Пример использования:

```
SELECT code, name INTO ::my.example1 FROM dict.regions limit 5 -- создадим таблицу и прокачаем ее 5 записями
SELECT 'test-code', 'test-name' APPEND INTO ::my.example1  -- добавим туда еще данные
SELECT * FROM ::my.example1 -- прочитаем данные из нашей таблицы
SELECT 'new-code', 'new-name' REWRITE INTO ::my.example1 -- перетрем таблицу новыми данными
SELECT * FROM ::my.example1 -- снова перечитаем
DROP TABLE ::my.example1 -- удаление пользовательской таблицы
```

FROM

Инструкция FROM определяет список таблиц Spectrum Data Gate, из которых будет производиться выборка данных.

`<table>` - название таблицы Spectrum Data Gate в формате `schema.table`.

`<alias>` - алиас таблицы. Позволяет обращаться к столбцам таблицы по алиасу.

Вместо таблицы можно использовать любой подзапрос:

```
select c, n from (select code as c, name as n from dict.regions limit 1)
```

Пример использования

```
SELECT * FROM dict.regions
```

WHERE

Инструкция WHERE задает условия, по которым будут отобраны записи в целевой таблице.

<expression> - произвольное выражение в формате, которое будет приведено к bool в соответствии с динамической типизацией.

Внимание! В отличие от SELECT и от HAVING в WHERE запрещено использование агрегирующих функций, так как их поведение на этой фазе не определенное

Пример использования

```
SELECT * FROM dict.regions WHERE normalized_code > 10 AND normalized_name
like 'Республика%'
```

GROUP BY

Стандартный синтаксис для указания группировки SQL, например

```
select gen, max(power) from dict.model_reference
where _raw = 'Mitsubishi Padjero Sport'
group by gen
```

В агрегатах очень удобно использовать ссылки во избежание ненужного дублирования

```
select year(some_date) as y, value from some.table
group by &y
-- вместо
select year(some_date) as y, value from some.table
group by year(some_date)
```

HAVING

Инструкция HAVING задает условия, которые должны выполняться уже после составления итоговой записи в SELECT. Все имена полей уже относятся не к таблице, а к результатам.

В отличие от ANSI SQL, для использования HAVING не обязательно использование группировки, или агрегирующих функций. Это просто некий “поздний фильтр”, однако его использование без агрегатов или поточных агрегатов обычно не имеет смысла и преимуществ перед WHERE

Например:

```
SELECT id AS code, code AS c FROM some.table
   WHERE code = 1   -- тут это относится к some.table.code
   HAVING code = 2  -- а тут это &code, то есть some.table.id в исходном
запросе
```

Использование с GROUP BY - это достаточно понятная и обычная вещь. Но HAVING можно применять и со скользящими агрегатами.

Пример: мы хотим ответить на вопрос, что вообще считать “большой суммой долга в ФССП”. Может это 100000? А может 1000000? И методика экспресс оценки которая выполняется за секунду с небольшим:

1. Будем использовать RND_FULL_SCAN
2. Возьмем сразу планку 10000, чтобы отсечь мелкие штрафы
3. Будем оставлять только те записи, которые больше всех предыдущих в потоке, то есть из ряда `(1, 2, 10, 4, 6, 8, 2, 12, 100, 2,3,4,5,9)` мы оставим только `(1,2,10,12,100)`
4. Подберем число LIMIT такое, чтобы примерно взять 80..95 перцентиль, например 10-11

Нужно выполнить на комбинации `cr_max` и `having`

```
select rownumber(), ep_number , fio , cr_max(debt_sum) maxds, debt_sum
from fssp.direct
where debt_sum  > 10000
having maxds = debt_sum
limit 10
with rnd_full_scan
```

EXPAND BY

Позволяет “развернуть” массивы значений из полей таблицы в строки. Каждый элемент массива из поля, указанного в выражении EXPAND BY, породит новую строку в результирующем наборе.

При этом возможны два варианта ссылки на поле:

1. Обычное выражение, содержащее названия полей, вызов функций, литералы и т.п. Результат такого выражения будет использован для разворачивания строк, но сами значения не будут отображены в результирующем наборе.

2. Ссылка на поле или его алиас из выражения SELECT (через &). В этом случае указанное поле будет использоваться для разворачивания строк, а в результирующем наборе исходное значение выражения будет заменено на элемент массива, соответствующий текущей строке.

Если в выражении EXPAND BY указано несколько полей или выражений, в результате будет построено декартово произведение по всем этим полям.

Например, есть следующая таблица `t`:

a	b	c
hello	(1, 2)	('X')
world	(3, 4)	('Y', 'Z')

В результате выполнения запроса:

```
SELECT a, b, c AS code
  FROM t
  EXPAND BY &b, &code
```

получим следующий набор строк:

a	b	code
hello	1	'X'
hello	2	'X'
world	3	'Y'
world	3	'Z'
world	4	'Y'
world	4	'Z'

Если ссылку на алиас заменить выражением:

```
SELECT a, b, c AS code
  FROM t
  EXPAND BY &b, c
```

результат будет следующим (поле `code` будет содержать исходные значения в каждой строке)

a	b	code
hello	1	('X')
hello	2	('X')
world	3	('Y', 'Z')
world	3	('Y', 'Z')

a	b	code
world	4	(‘Y’, ‘Z’)
world	4	(‘Y’, ‘Z’)

ORDER BY

Управляет упорядочением результатов. Ссылается на имена полей в итоговом запросе, а не исходном. Ключевые слова ASC и DESC управляют упорядочением по возрастанию или убыванию соответственно. Подвыражение `NULLS FIRST`, `NULLS LAST` управляют упорядочением `NULL`, если стоит `NULLS FIRST` (по умолчанию), то `NULL` оказываются выше остальных значений, при `NULLS LAST` - ниже.

```
SELECT code, name from dict.regions order by name desc
```

OFFSET

Инструкция `OFFSET` задает количество записей, которое надо пропустить перед началом выдачи результатов. Позволяет реализовать сценарии постраничного запроса к таблице.

Пример использования

```
SELECT * FROM dict.regions OFFSET 20
```

LIMIT

Инструкция `LIMIT` задает максимальное количество записей, которое может быть возвращено в результате запроса.

Пример использования

```
SELECT * FROM dict.regions OFFSET 20 LIMIT 10
```

WITH

Инструкция `WITH` позволяет указать дополнительные параметры выполнения запроса.

Поддерживаются следующие подсказки:

- `LOCAL_NODE` - режим сбора данных только с одной ноды Spectrum Data Gate - с которой установлено текущее соединение. Используется для запросов к системным таблицам, таким как `sys.session` и `sys.nodes`.
- `SEQ_FULL_SCAN` - режим последовательного сканирования таблицы Cassandra. Форсирует полный обход таблицы в один поток, что позволяет получить воспроизводимую последовательность токенов. Может быть удобно для запросов с необходимостью восстановления после ошибки.

- RND_FULL_SCAN - режим сканирования таблицы Cassandra в несколько потоков. Не гарантирует порядок обхода строк. Результирующий набор данных может содержать записи в произвольном порядке. Этот режим работает намного быстрее последовательного сканирования.
- EXPLICIT_FIELDS - позволяет вывести в результатах запроса полный набор полей таблицы, включая технические поля и поля, требующие явного указания в запросе.
- PARALLEL [<n>] - позволяет распараллелить выполнение команды SELECT к которой относится, можно указать число от 2 до 32, если не указать - используется значение по умолчанию 4 - позволяет существенно ускорить выполнение запросов с подзапросами (например обращение к другим источникам)
- FULL_SCAN - в текущей версии позволяет совершить запрос к Posgtres с пустым WHERE. Необходимо понимать, что использование этой опции может сгенерировать большой трафик и повлиять на работоспособность инфраструктуры. Не рекомендуется использовать без ясного понимания, зачем это делается. Механика работы: позволяет отправлять PRIMARY SELECT в Postgres в котором нет ни одного аргумента в WHERE.

Пример использования

```
SELECT * FROM sys.tables WITH EXPLICIT_FIELDS
```

INSERT

Поддерживается только для NATS таблиц

SDQL поддерживает возможность вставки значений в таблицу

Запрос вставки имеет схожий синтаксис со стандартным запросом вставки PG

```
INSERT INTO sem.a (subject, "objects")
VALUES ('configs', ('key_1', 'key_2'));
```

Правила составления INSERT запроса:

```
INSERT INTO schema.table ([FIELD, ...]) VALUES ([VALUE...])
```

FIELD - Список полей таблицы, в которые будут ставлены значения:

1. Допустимы только поля с меткой `ins`

2. Все поля с меткой `ins` обязательны для использования

VALUE - константные значения, которые будут записаны в поля. При вставке массива, поддерживаются только массивы строк (`'STRING'`)

Общие правила:

1. Количество полей должно быть равно количеству записываемых значений

2. Значения записываются в том же порядке, в котором перечислены поля

По аналогии с PG в случае успеха возвращает таблицу с указанием кол-ва записанных рядов

Название	Тип	f()	Комментарий	Теги
count	INT	``	Количество записанных рядов	

При использовании INSERT вместе с PREPARE в качестве аргументов можно передавать значения, но не поля

```
PREPARE
INSERT INTO sem.a (subject, "objects") VALUES ($1, $2);

EXECUTE 'UID подготовленного запроса' @P 'configs', ('key_1', 'key_2')
```

@P - передача параметров

В Spectrum Data Gate параметры можно передать как на уровне протокола, так и в самом SDQL:

Самая простая форма - именованные параметры

```
SELECT * FROM my.table WHERE tp = $tp and value = $n @P tp=3, n='hello'
```

Также поддерживаются позиционные параметры с номерами, начинающимися с 0

```
SELECT * FROM my.table WHERE tp = $0 and value = $1 @P 3, 'hello'
```

Особый смысл конструкция обретает в сочетании с PREPARE, тогда это значения параметров по умолчанию

```
PREPARE SELECT * FROM my.table WHERE tp = $tp and value = $n @P tp=3,
n='hello' --uid
EXECUTE '<uid>' -- 3, hello
EXECUTE '<uid>' @P tp=10 -- 10, hello
EXECUTE '<uid>' @P n='world' -- 3, 'world'
EXECUTE '<uid>' @P n='world', tp=10 -- 10, 'world'
```

Это же работает и с позиционными параметрами:

```
PREPARE SELECT * FROM my.table WHERE tp = $1 and value = $2 @P 3, 'hello'
--uid
EXECUTE '<uid>' -- 3, hello
EXECUTE '<uid>' @P 10 -- 10, hello
EXECUTE '<uid>' @P ?, 'world' -- 3, 'world'
```

Знак вопроса ? в позиционных параметрах означает “оставить значение по умолчанию”

@R - ограничение ролей

Системная инструкция @R позволяет ограничить набор ролей, с которыми выполняется запрос. Роли могут быть ограничены только тем набором, который уже предоставлен пользователю в явном или неявном виде - т.е. расширить свои роли с помощью этой команды нельзя.

Пример использования

```
SELECT * FROM dict.regions @R 'public, sales'
```

@D - информация об использованном драйвере

Системная инструкция @D позволяет указать тип и версию драйвера Spectrum Data Gate, который используется для запроса. Это позволяет выводить клиенту предупреждения о неподдерживаемых функциях и обеспечивать обратную совместимость сервера с различными версиями драйверов.

Принимает параметры:

- T- тип драйвера, например, golang или jdbc
- V- версия драйвера, например, 0.2.131
- O - дополнительные опции

Пример использования

```
SELECT * FROM dict.regions @D T 'jdbc' V '0.2.131'
```

Динамическая типизация в Spectrum Data Gate

В Spectrum Data Gate во всех выражениях используется **динамическая типизация**.

Это означает, что тип всех переменных в выражении определяется прежде всего не исходным типом их значений, а **теми действиями, которые над ними производятся**.

Функции явного приведения типа

1. `bool` - `select 1, bool(1)` - приводит значение как `bool`
2. `string` - `select , string(1)` - приводит значение к строке (обычно ни на
3. `int` - `select 2.0, 2.1, int(2.0), int(2.1), int(null)` - приводит к `int` любые валидные числа с округлением, `null` рассматривается как `0`
4. `int_s` - строгое приведение к `int`, с проверкой дробной части `int(2.0)` - ОК, `int(2.1)` - ошибка
5. `float` - `select 2, float(2), float('233.133334444891234'), float(null)` - приводит числа к float
6. `float_s` - строгое приведение к `float`, влияет на преобразование `decimal`, например `float_s('233.133334444891234')`
7. `decimal` - приведение к decimal, при ошибках - `'0.0'`
8. `decimal_s` - строгое приведение к decimal, возвращает ошибки
9. `array` - плоский конструктор массивов `array(1,(2,(3,4)))` формирует `(1,2,3,4)`

Выражения

В Spectrum Data Gate и в SELECT и в WHERE можно использовать произвольные выражения, при этом разрешены и скобки для группировки и правильной приоритизации действий.

Выражения бывают:

1. одноместные: `some_field`, `1` ,`null` - в основном это константы и обращения к полям
2. бинарные `1 + 2`, `some_field > 2`, `some_field[1]` - сравнения, математика, индексаторы и т.п.
3. N-арные `(1,2,3,4)`, `some_func(1, 2, 3)`, `1.some_func(2,3)` - конструкторы массивов и вызовы функций

Значения

1. имена полей в таблицах - `code` , `t1.code` - при вычислениях замещаются значениями соответствующих полей
2. фиксированные константы:
   - `null` - значение `null`
   - `true` - булево `true`
   - `false` - булево `false`
3. литералы, типизированные константы:
   - `'any text'` - строки (в одинарных кавычках), могут быть многострочные
   - `'any ''text'''` - двойные апострофы (ANSI SQL) для передачи в строке апострофов
   - `1`, `-1234` - целые числа (int)
   - `1.1`, `-1234.3` - числа с плавающей запятой (float)

- `'123123132132131232132 1.3232'` - `decimal` - понимаются к строки, приводимые к числу, могут быть вне диапазонов int, float

4. ссылки на выражения в SELECT (&name), например `select (some_field+2) as c, &c * 2 from some.table where &c > 100`

Проверка на NULL и проверка на не-NULL

---

1. `a is null`
2. `x is not null`

---

1. `'','  ','null', 'nil', '<nil>'` - пустые строки и строки с явным упоминанием NULL
2. `[],{}` - пустые массивы и объекты
3. `1800-01-01`, `3000-01-01` - наши особенные внутренние константы для NULL –дат

Внимание! При проверке на NULL используется эффективное приведение к NULL, так например как NULL трактуются

Общие логические операторы

4. Отрицание NOT (not) - `select a from t where NOT ( a > 1 AND b < 2 )`
5. Конъюнкция AND (and) - `select a, b from t where c > 1 AND b < 2 AND c > 4`
6. Дизъюнкция OR (or) - `select a, b from t where c > 1 OR b < 2 OR a > 100`

Бинарные выражения и операторы

При описании типов NUMBER - означает “любое число” - int, float, decimal, будет произведена попытка их приведения ANY обозначает - любой тип

Название	Оператор	Типы на вход	Тип результата	Пример	Примечание
Равенство	=	ANY	BOOL	`SELECT 23 = '23'`	**приведение типов, не учитывает регистр строк**
Неравенство	!=,<>	ANY	BOOL	`SELECT 23 != '24'`	**приведение типов, не учитывает регистр строк**
Больше	>	ANY	BOOL	`SELECT 'bac' > 'abc'`	**приведение типов, не учитывает регистр строк**
Больше или равно	>=	ANY	BOOL	`SELECT 'bac' >= 'abc'`	**приведение типов, не учитывает регистр строк**
Меньше	<	ANY	BOOL	`SELECT 'abc' < 'bac'`	**приведение типов, не учитывает регистр строк**
Меньше или равно	<=	ANY	BOOL	`SELECT 'abc' <= 'bac'`	**приведение типов, не учитывает регистр строк**
В диапазоне (включая границы)	BETWEEN	ANY	BOOL	`SELECT 2 BETWEEN (1 AND 3)`	внутри замещается на `(2 >= 1 and 2 <= 3)`, стандартное поведение
Соответствие паттерну LIKE	LIKE, ~~	STRING	BOOL	`SELECT 'abcde' like '%C%'`	**приведение типов, не учитывает регистр строк**, поддерживается стандартная опция ESCAPE (ANSI SQL)
Не соответствие паттерну LIKE	NOT LIKE, !~~	STRING	BOOL	`SELECT 'abcde' not like '%x%'`	**приведение типов, не учитывает регистр строк**, поддерживается стандартная опция ESCAPE (ANSI SQL)
Соответствие паттерну REGEX	~	STRING	BOOL	`SELECT 'abcde' ~ '^.*c.*$'`	**приведение типов, не учитывает регистр строк**
Не соответствие паттерну REGEX	!~	STRING	BOOL	`SELECT 'abcde' !~ '^.*x.*$'`	**приведение типов, не учитывает регистр строк**
Нахождение значения в массиве	IN	ANY	ARRAY	`select 1 in (2,1,3)`	

Название	Оператор	Типы на вход	Тип результата	Пример	Примечание
Отсутствие значения в массиве	`NOT IN`	`ANY`	`ARRAY`	`select 5 not in (2,1,3)`	
Нахождение значения в массиве (левый)	`HAS`	`ARRAY`	`ANY`	`select (2,1,3) HAS 1`	
Наличие любого из значений в массиве	`HAS ANY`	`ARRAY`	`ARRAY`	`select (2,1,3) HAS ANY (1,5,6)`	
Наличие всех значение в массиве	`HAS ALL`	`ARRAY`	`ARRAY`	`select (1,5,6,3,2) HAS ALL (2,1,3)`	
Отсутствие любого из значений	`HAS NOT`	`ARRAY`	`ARRAY`	`select (1,5,6,3,2) HAS NOT (4, 9)`	
Сложение	`+`	`NUMBER`	`NUMBER`	`SELECT 1 + '2.3'`	
Вычитание	`-`	`NUMBER`	`NUMBER`	`SELECT 4.5 - '2.3'`	
Умножение	`*`	`NUMBER`	`NUMBER`	`SELECT 5 * 6`	
Деление	`/`	`NUMBER`	`NUMBER`	`SELECT 5 / 6`	**см. ниже замечание про деление на 0**
Деление по модулю (остаток)	`%`	`INT`	`INT`	`SELECT 19.0 % 10`	
Возведение в степень	`^`	`NUMBER`	`NUMBER`	`SELECT 2 ^ 3`	

По поводу деления на 0 - так как при делении производится приведение int к float64 (чтобы 1/2 было 0.5, а не 0), то соответственно деление на 0 не является ошибкой, результатом будет `Infinity`, бесконечность, если требуется, однако при приведении к int это будет ошибкой то есть `select 1/0` - нет ошибки, `select int(1/0)` - будет ошибка

Примечания по математическим операциям

1. если на вход были `int`, `float`, `string` то будет использоваться прямое и обратное приведение к “лучшему” типу
   - то есть `SELECT 1.0 + 2.0 -> int(3)`, `SELECT 1.1 + '2' -> float(3.1)`
2. но если на вход был хотя бы один `decimal` (`money`) - то именно этот тип и сохранится - то есть это защита вычислений с большими или денежными величинами от случайного приведения
   - то есть `SELECT decimal(1.0) + 2.0 -> decimal(3)`
3. деление на 0
   - если использовались `int`, `float`, `string`, то деление на 0 это `float(Infinity)`
     - `SELECT 1 / 0 -> float(Infinity)`
   - если нужно произвести деление обычных типов, но обеспечить отсутствия неопределенностей (NaN,+Inf,-Inf), то можно воспользоваться функциями `if_defined(value)` или `must_defined(value)`
     - `select if_defined(1/0) -> null`
     - `select if_defined(1/2) -> 0.5`
     - `select must_defined(1/0) -> error`
     - `select must_defined(1/2) -> 0.5`
   - если использовались `decimal` - то деление на 0 - ошибка (не определено)
     - `SELECT decimal(1) / 0 -> error`

Индексаторы

Особым видом выражений является “индексатор”, который в общем виде выглядит `<expression>[<index>]`

Результат будет зависеть от типа `<expression>` и от типа `<index>` :

Тип выражения	Тип индексатора	Результат	Пример
`STRING`	`INT`	буква в заданной позиции или null если строка короче	`'hello'[1] -> 'e'`
`ARRAY`	`INT`	элемент с указанным номером или null если массив короче	`(1,2,3)[1] -> 2`
`OBJECT`	`STRING`	поле в JSON объекте по указанному пути	допустим `obj = {x:{a:1}}`, тогда `obj['x.a'] -> 1`
`ARRAY<OBJECT>`	`STRING`	массив результатов индексации по пути для объектов внутри массива	допустим `a = [{x:{a:1}},{x:{a:3}}]`, тогда `a['x.a'] -> [1,3]`

Остальные варианты вызовов приведут к ошибке

Массивы, наборы значений

Конструирование массива `(1, 2, 3)`. Массивы используются как

1. значения возврата `select (1,2,3) as codes`
2. как аргументы некоторых операций и функций `x in (1,2,3)`

Вызовы функций

Вызов функции состоит в передаче имени и аргументов функции, например:

```
select upper('hello')

select if_null(x, 1)
```

Следовательно, можно по разному вкладывать функции:

```
select concat(upper(x), ' ', upper(y))
```

Что примечательно - SDQL поддерживает оба стиля вызова функций как классический C/SQL `upper(trim(replace('hello','l','r')))` так и “объектный” `'hello'.replace('l','r').trim().upper()` вы можете сами выбирать стиль, в котором Вам удобнее писать

Ссылки на выражения в SELECT

В отличие от ANSI SQL все имена и псевдонимы полей в SELECT должны быть уникальны.

Например нельзя написать `select a as n, b as n`, так как будет два выражения с псевдонимом n.

На первый взгляд это некое ограничение, возможно страхующее от случайных copy-paste ошибок.

Однако это открывает эксклюзивную возможность SDQL - **ссылки на выражения**

Например в ANSI SQL мы можем написать так:

```
select (a + b + c + d + e) as m from my.tab
```

Если потребуется ограничение , или производное поле, то можно его написать так:

```
select (a + b + c + d + e) as m,  (a + b + c + d + e) * 2 as m2
from my.tab where  (a + b + c + d + e) > 100 and (a + b + c + d + e) < 200
```

В SDQL из-за свойства уникальности имен полей это решается следующим образом:

```
select (a + b + c + d + e) as m, &m * 2 as m2
from my.tab where &m > 100 and &m < 200
```

Внутри при этом выражение раскладывается в

```
select (a + b + c + d + e) as m,  (a + b + c + d + e) * 2 as m2
from my.tab where  (a + b + c + d + e) > 100 and (a + b + c + d + e) < 200
```

Таким образом, это совместимое с обычным SQL поведением, но с куда более защищенным от copy-paste ошибок поведением и с более лаконичным синтаксисом.

Стандартная библиотека функций

В данном разделе приведем реестр встроенных функций

Примечание, многие функции возвращают неопределенные типы, например `number`, `any`. При чтении сигнатур обращайте внимание на скобки вокруг аргументов. Например `sum(<number>) <number>` и `sin(number) float`, в первом случае это означает, что в ответ будет именно тот тип числа на входе, то есть если тип поля `a: INT sum(a)` это тоже `int`, в случае же с `sin` - тип number никак особо не обрабатывается и не влияет на возвращаемое значение.

Например `if_null_s(value:<any>,default:<any>) <any>` означает, что тип результата будет самым подходящим общим из типов двух аргументов, например если и `value` и `default` будут `int` то и результат будет `int`

Spectrum Data Gate имеет автоприведение типов, соответственно если в сигнатуре указано `sin(float) float` - это значит, что можно вызывать и `sin('1')` так как строка `1` без потерь приводима к `float.`

У некоторых параметров стоит символ `?` - это значит он не обязательный, можно не указывать

Общие функции

Название	Сигнатура	Примечание
Длина строки, массива	`len(any...) int`	`len(null)->0`, `len('hello')->5`,`len(123)->1`, `len((1,2,3))->3`
Тип выражения и значения	`type_of(any) array`	`type_of(if ('a' = 'b', 1, null )) -> ['INT','<nil>']` - первый элемент массива - формальный тип по выводу, второй - актуальный тип значения

Явное преобразование типов

В SDQL используется динамическая типизация, но иногда полезно явно привести значение к нужному типу. В SDQL это делается через функции с названием типа. Суффикс `_s` указывает на `strict`, более строгое приведение.

Название	Сигнатура	Примечание
Явное приведение к `int`	`int(any) int`	`int(null) -> 0`, `int(2.3) -> 2`, `'hello' -> 0`
Строгое приведение к `int`	`int_s(any) int`	`int_s(null) -> 0`, `int_s(2.0) -> 2`, но `int_s(2.3)`, `int_s('hello')` - **ошибки**
Явное приведение к `float`	`float(any) float`	`float(null) -> 0`, `float('2.3') -> 2.0`, `float('hello') -> 0.0`
Строгое приведение к `float`	`float_s(any) float`	`float_s(null) -> 0`, `float_s('hello')` - **ошибка**
Явное приведение к `decimal`	`decimal(any) decimal`	`decimal(null) -> 0`, `decimal('2.3') -> 2.3`, `float('hello') -> 0.0`
Строгое приведение к `decimal`	`decimal_s(any) decimal`	`decimal_s(null) -> 0`, `decimal_s('hello')` - **ошибка**
Явное приведение к `bool`	`bool(any) bool`	`bool(null)-> false`, `bool(1)-> true`
Явное приведение к `string`	`string(any) bool`	`bool(null)-> false`, `bool(1)-> true`
Приведение к строке с датой без времени	`date(any) string (date)`	`date('1979-01-23T12:14:15.000+03:00') -> '1979-01-23`
Приведение к строке с датой, только если нет части со временем	`date_s(any) string (date)`	`date_s('1979-01-23T12:14:15.000+03:00') -> error`
Приведение к полному времени (таймстампу, в виде строки)	`datetime(any) string (datetime)`	
Парсинг JSON (произвольного)	`json(string) any`	`json('1')`, `json('[1,2]')`, `json('{"a":2}')`
Парсинг JSON (только объекты)	`json_obj(string) object`	`json_obj('{"a":2}')`
Построение объектов	`build_obj((string,any), ...)`	`build_obj(('a',2),('b',3))`

Математические функции

Название	Сигнатура	Примечание
Константа pi	`pi() float`	
Константа e	`e() float`	
Модуль, абсолютное значение	`abs(float) float`	
Квадратный корень	`sqrt(float) float`	
Экспонента	`exp(float) float`	
Логарифм натуральный	`ln(float) float`	
Логарифм по основанию 2	`log2(float) float`	
Логарифм по основанию 10	`log10(float) float`	
Ближайшее целое	`round(float) int`	`round(2.1)->2`, `round(2.9)->3`
Ближайшее целое (в меньшую сторону)	`floor(float) int`	`floor(2.9) -> 2`
Ближайшее целое (в большую сторону)	`ceil(float) int`	`ceil(2.1) -> 3`
Модуль, абсолютное значение	`abs(float) float`	
Синус (от угла в рад.)	`sin(float) float`	
Косинус (от угла в рад.)	`cos(float) float`	
Тангенс (от угла в рад.)	`tan(float) float`	
Арксинус (в рад.)	`asin(float) float`	
Арккосинус (в рад.)	`acos(float) float`	
Арктангенс (в рад.)	`atan(float) float`	
Перевод градусов в радианы	`rad(float) float`	
Перевод радиан в градусы	`grad(float) float`	
Проверка на NaN	`isnan(float) bool`	`isnan(0/0)` -> `true`
Проверка на Inf	`isinf(float, sign int?) bool`	`isinf(1/0)` -> `true`, sign - это 1 - проверка на +Inf, -1 - проверка на -Inf

Строковые функции

Название	Сигнатура	Примечание
Перевод в верхний регистр	`upper(string) string`	
Перевод в нижний регистр	`lower(string) string`	
Обрезка пробельных символов с краев	`trim(string) string`	
Соединение строк	`concat(string,...) string`	`concat('a','-',1) -> 'a-1`
Форматирование строк (golang - вариант)	`fmt(string,args...) string`	`fmt('%d - %d (%d)',1,2,3) -> '1 - 2 (3)`
Замена вхождения подстроки на другую	`replace(src:string,find:string,rep:string) string`	`replace('world','or','ou') -> 'would'`
Замена вхождения подстроки на другую (regex)	`rx_replace(src:string,find:regex,rep:string) string`	`replace('123','^(\d)\d+$','$1...') -> '1...'`
Извлечение подстроки при разбиении на поля	`str_field(src:string,delimiter:string,num:int) string`	`str_field('1;2;3',';',1) -> '2'` (от 0)
Получение подстроки	`substring(src:string, from int, to int?) string`	начало с 0, верхняя граница не включается `substring('hello',2,4) -> 'll'`, если указать в to `null`, то это “до конца строки”
Разделение строки на массив	`split(src:string, delimiter:string) array`	`split('1:2:3', ':') -> ['1','2','3']`
Слияние массива в строку	`str_join(src:array, delimiter:string) string`	`str_join((1,2,3), ':') -> '1:2:3'`

Функции для даты и времени

Название	Сигнатура	Примечание
Год	`year(datetime) int`	
Месяц	`month(datetime) int`	
День	`day(datetime) int`	
Таймстамп миллисекунды	`timestamp_ms(datetime) int`	
Таймстамп наносекунды	`timestamp_ns(datetime) int`	

Логические функции

Название	Сигнатура	Примечание
Выбор первого не-NULL элемента	`coalesce(<any...>) <any>`	`coalesce(null,'',2,3) -> 2` NULL понимается в логическом смысле (пустые строки например тоже null)
Альтернатива для логического null	`if_null(value:<any>,default:<any>) <any>`	`if_null(null, 1) -> 1`, `if_null(1,2)->1`
Альтернатива для null	`if_null_s(value:<any>,default:<any>) <any>`	`if_null_s(null, 1)->1`, но в отличе от `if_null`: `if_null_s('',1) -> ''`
Тернарный if-then-else	`if(cond:bool,then:<any>,else:<any>) <any>`	`if(1 > 2, 'x', 'y') -> 'y'`, `if(2 > 1, 'x','y') > 'x'`

Работа с массивами

Некоторые функции используют выражения в качестве параметра - на данный момент выражения поддерживают два параметра:

1. `it` - значение текущего элемента

2. `acc` - аккумулятор для метода `fold_of`

Название	Сигнатура	Примечание
Первый элемент массива	`first_of(array) anyfirst_of(array, expression) any`	`first_of((1, 2, 3)) => 1first_of((1, 2, 3), it >= 2) => 2`
Последний элемент массива	`last_of(array) anylast_of(array, expression) any`	`last_of((1, 2, 3)) => 3last_of((3, 1, 2), it > 2) = > 3`
Выборка элементов массива по диапазону	`slice_of(array, from int, to int?) array`	`slice_of((1, 2, 3), 1, 2) => (2)slice_of((1, 2, 3), null, 1) => (1)slice_of((1, 2, 3), 1, null) => (2,3)`
Исключение дубликатов из массива	`set_of(array) array`	`set_of((1, 2, 2, 3, 1)) => (1, 2, 3)`
Объединение двух массивов	`union_of(array, array) array`	`union_of((1, 2, 3), (3, 4)) => (1, 2, 3, 3, 4)`
Пересечение двух массивов	`intersect_of(array, array) array`	`intersect_of((1, 2, 3), (3, 4)) => (3)`
Исключение элементов другого массива	`subtract_of(array, array) array`	`subtract_of((1, 2, 3), (3, 4)) => (1, 2)`
Развернуть массив	`reverse_of(array) array`	`reverse_of((2, 1, 3)) => (3, 1, 2)`

Название	Сигнатура	Примечание
Отсортировать массив в порядке возрастания	`sort_of(array) arraysort_of(array, direction) arraysort_of(array, expression) arraysort_of(array, direction, expression) array`	`direction` - поярдок сортировки `asc`, `descsort_of((1, 3, 4)) => (1, 3, 4)sort_of((1, 3, 4), 'asc') => (1, 3, 4)sort_of((1, 3, 4), 'desc') => (4, 3, 1)sort_of((1, 2, 3), it * -1) => (3, 2, 1)sort_of((1, 2, 3), 'desc', it % 2) => (1, 3, 2)`
Отсортировать массив в порядке убывания	`sort_desc_of(array) array`	`sort_desc_of((2, 1, 3)) => (3, 2, 1)`
Минимальный элемент массива	`min_of(array) anymin_of(array, expression) any`	`min_of((3, 4, -1, 2)) => -1min_of((3, 4, -1, 2), it * -1) => 4`
Максимальный элемент массива	`max_of(array) anymax_of(array, expression)`	`max_of((3, 4, -1, 2)) => 4max_of((3, 4, -1, 2), it % 3) => 2`
Сумма элементов массива	`sum_of(array) number`	`sum_of((2, 1, 3)) => 6`
Среднее значение элементов массива	`avg_of(array) number`	`avg_of((2, 1, 3)) => 2`
Фильтрация массива по выражению	`filter_of(array, expression) array`	`filter_of((1, 2, 3), it >= 2) => (2, 3)`
Преобразование значений массива по выражению	`transform_of(array, expression) array`	`transform_of((1, 2, 3), it * 2) => (2, 4, 6)`
Свернуть массив	`fold_of(array, initial any, expression) any`	`fold_of((1, 2, 3), 10, acc + it * 2) => 22`
Конкатенация элементов массива	`join_of(array) stringjoin_of(array, delimiter string) stringjoin_of(array, expression) stringjoin_of(array, delimiter string, expression) string`	`join_of((1,2,3)) => 1, 2, 3join_of((1,2,3), '.') => 1.2.3join_of((1,2,3), it * 2) => 2, 4, 6join_of((1,2,3), '.', it * 2) => 2.4.6`
Расчет медианы	`median_of(array) float`	`median_of((3, 100)) => 51.5 median_of((100, NaN)) => NaN`

Название	Сигнатура	Примечание
Расчет персентиля	`percentile_of(array, float) float`	`percentile_of((3, 100), 0.5) => 51.5` `percentile_of((100, NaN), 0.5) => NaN`
Расчет стандартного отклонения	`std_of(array) float`	`std_of((11.74, 50.00, 69.72)) => 29.4799` `std_of((11.74, NaN, 63.90)) => NaN`
Расчет дисперсии	`disp_of(array) float`	`disp_of((11.74, 50.00, 69.72)) => 869.0644` `disp_of((11.74, NaN, 63.90)) => NaN`
Расчет ковариации	`cov_of(array, array) float`	`cov_of((11.74, 50.00), (90.93, 3.77)) => -1667.37` `cov_of((11.74, 50.00,), (90.93, NaN)) => NaN`
Расчет корреляции	`corr_of(array, array) float`	`corr_of((11.74, 50.00), (90.93, 3.77)) => -1` `corr_of((11.74, 50.00), (90.93, NaN)) => NaN`

Агрегаты

Агрегирующие функции применяются в сочетании с GROUP BY, HAVING и вычисляют значения для всего набора данных или группы.

Название	Сигнатура	Примечание
Подсчет (не-NULL) значений	`count(any) int`	`count(code)`
Подсчет уникальных (не-NULL) значений	`count_distinct(any) int`	`count_distinct(code)`
Минимум	`min(<any>) <any>min(<any>, expression) <any>`	`min(debt)min(debt, it % 10000)`
Максимум	`max(<any>) <any>max(<any>, expression) <any>`	`max(debt)max(debt, it % 10000)`
Сумма	`sum(<number>) <number>`	`sum(price)`
Среднее	`avg(number) float`	`avg(price)`
Первый элемент группы	`first(<any>) <any>`	`count_distinct(code)`
Последний элемент группы	`last(<any>) <any>`	`count_distinct(code)`

Название	Сигнатура	Примечание
Сбор массива значений	`array_agg(any) array`	`([1],[2],[null],[1]) -> [1,2,1]`
Сбор массива значений (включая NULL)	`array_agg_n(any) array`	`([1],[2],[null],[1]) -> [1,2,null,1]`
Сбор уникальных значений	`set_agg(any) array`	`([1],[2],[null],[1]) -> [1,2]`
Сбор уникальных значений (включая NULL)	`set_agg_n(any) array`	`([1],[2],[null],[1]) -> [1,2,null]`
Свертывание значений	`fold(any, initial any, expression) any`	`fold(value, 10, acc + int(it) * 2)`
Конкатенация значений	`join(any) stringjoin(any, delimiter string) stringjoin(any, expression) stringcr_join(any, delimiter string, expression) string`	`join(value)join(value, '.')join(value, it * 2)join(value, '.', it * 2)`
Расчет медианы	`median(float) float`	Возвращает NaN при некорректном значении в наборе `median(price)`
Расчет персентиля	`percentile(float, float) float`	Возвращает NaN при некорректном значении в наборе `percentile(price, 0.95) percentile(price, 95)`
Расчет стандартного отклонения	`std(float) float`	Возвращает NaN при некорректном значении в наборе `std(price)`
Расчет дисперсии	`disp(float) float`	Возвращает NaN при некорректном значении в наборе `disp(price)`
Расчет ковариации	`cov(float, float) float`	Возвращает NaN при некорректном значении в наборе `cov(price, income)`
Расчет корреляции	`corr(float, float) float`	Возвращает NaN при некорректном значении в наборе `corr(price, income)`

Скользящие агрегаты

Под “скользящими” понимаются агрегирующие функции, которые рассчитываются на текущей строке в рамках уже переданного ранее, а не всего датасета (собственно поэтому это скользящие агрегаты).

Большинство функций этой группы имеют префикс `cr_*` , чтобы отличать от сгруппированных агрегатов в будущем

Название	Сигнатура	Примечание
Номер строки	`rownumber() int`	`rownumber()`
Подсчет (не-null) значений	`cr_count(any) int`	`cr_count(code)`
Подсчет уникальных (не-null) значений	`cr_count_distinct(any) int`	`cr_count_dictinct(code)`
Сбор массива значений	`cr_array(any) array`	`([1],[2],[null],[1]) -> [1,2,1]`
Сбор массива значений (включая NULL)	`cr_array_n(any) array`	`([1],[2],[null],[1]) -> [1,2,null,1]`
Сбор уникальных значений	`cr_set(any) array`	`([1],[2],[null],[1]) -> [1,2]`
Сбор уникальных значений (включая NULL)	`cr_set_n(any) array`	`([1],[2],[null],[1]) -> [1,2,null]`
Минимум (текущий локальный)	`cr_min(<any>) <any>cr_min(<any>, expression) <any>`	`cr_min(debt)cr_min(debt, it % 10000)`
Максимум (текущий локальный)	`cr_max(<any>) <any>cr_max(<any>, expression) <any>`	`cr_max(debt)cr_max(debt, it % 10000)`
Сумма (текущего и предыдущих значений)	`cr_sum(<number>) <number>`	`cr_sum(price)`
Среднее (текущее)	`cr_avg(number) float`	`cr_avg(price)`

Название	Сигнатура	Примечание
Предыдущее (текущее)	`cr_prev(<any>) <any>`	например `cr_prev(_token)`, `_token` - вывод предыдущего токена в наборе
Свертывание значений	`cr_fold(any, initial any, expression) any`	`cr_fold(value, 10, acc + int(it) * 2)`
Конкатенация значений	`cr_join(any) stringcr_join(any, delimiter string) stringcr_join(any, expression) stringcr_join(any, delimiter string, expression) string`	`cr_join(value)cr_join(value, '.')cr_join(value, it * 2)cr_join(value, '.', it * 2)`

Функция для генерации хеша

Название	Сигнатура	Примечание
Алгоримт генерации `md5` хеша	`md5(string) string`	
Алгоримт генерации `sha1` хеша	`sha1(string) string`	
Алгоримт генерации `sha256` хеша	`sha256(string) string`	
Алгоримт генерации `murmur64` хеша	`murmur64(string) string`	
Алгоримт генерации `murmur64` хеша	`murmur64i(string) int`	В виде целочисленного числа (аналогично `cassandra token(...)`)

Функции для нормализации ФИО

Название	Сигнатура	Примечание
Объединение ФИО	`fio_join(string, string, string) string`	`fio_join('Иванов', 'Иван', 'Иванович') -> 'Иванов Иван Иванович'`, 3 аргумент опциональный
Извлечение части ФИО	`fio_extract(string, string) string`	`fio_extract('o', 'Иванов Иван Иванович') -> 'Иванович'`, если нет какой-либо части ФИО, вернёт пустую строку, если частей больше трёх, вернёт всё что есть в часть с отчеством

Синтаксис

```
build_table (
   (
        ('<column-name-1>', '<column-type-1>'), -- например ('id','int'),
        ('<column-name-2>', '<column-type-2>'), -- например
('name','string')
        -- ....
   ),
   (
        (val11, val12), -- первая запись, например (1,'hello'),
        (val21, val22), -- вторая запись, например (2,'world'),
   ),
)
```

Пример простого запроса

```
SELECT id, name FROM build_table (
   (
        ('id', 'int'),
        ('name', 'string'),
   ),
   (
        (1, 'hello'),
        (2, 'world'),
   ),
)
```

Вернет таблицу

id	name
1	hello
2	world

Предполагаемое практическое использование - в составе CROSS APPLY для нескольких запросов в составе одного

```
SELECT q.query_id, q.fio, q.birth,
    (SELECT FIELDS src.passport, src.inn, src.address FROM super.source
src WHERE src.fio = q.fio AND src.birth = q.birth) -- вот тут собственно
смысл, что иначе как CROSS APPLY на дополнительную таблицу такого
поведения в один
     -- запрос не добиться
```

```
 FROM build_table (
   (
        ('query_id', 'int'),
        ('fio', 'string'),
        ('birth', 'date'),
   ),
   (
        (1, 'Иванов Петр Семенович', '1980-01-03'),
        (2, 'Севастьянов Сергей Пименович','1975-04-23'),
        (3, 'Леонтьев Эдуард Корнилович','1993-03-05'),
   ),
) as q
```

Обогащение запросов одних источников через другие (cross-apply)

В Spectrum Data Gate реализован вариант многотабличных (и соответственно комплексных) запросов в варианте с условным названием CROSS APPLY, который среди реляционных баз используется в MS SQL.

В Spectrum Data Gate он реализован посредством подзапросов.

Рассмотрим вызов:

```
SELECT f.fio, f.birth, f.ep_number,
(SELECT FIELDS inn, snils FROM some.base WHERE fio=f.fio AND birth =
f.birth LIMIT 1)
FROM fssp.direct f
LIMIT 100
WITH RND_FULL_SCAN PARALLEL 4
```

В этом случае происходит:

1. Spectrum Data Gate начнет формировать поток записей из fssp.direct с полями fio, birth
2. Для каждой записи из fssp.direct будет выполнен запрос в some.base и оттуда будет извлечены соответствующие `inn` и `snils` (**такой базы нет, это для примера синтаксиса**)
3. Затем инструкция `fields` указывает, что `inn` и `snils` надо сделать полями наружного запроса
4. Все это делатеся в паралель по 4 запроса одновременно к источнику `some.base`

На выходе будут поля `fio`, `birth`, `inn`, `snils`

Допустим нужно, чтобы остались только записи, где `inn` не пустой, тогда нужно добавить наружный надзапрос:

```
SELECT * FROM (
        SELECT f.fio, f.birth, f.ep_number,
        (SELECT FIELDS inn, snils FROM some.base WHERE fio=f.fio AND birth
= f.birth LIMIT 1)
```

```
        FROM fssp.direct f
        WITH RND_FULL_SCAN PARALLEL 4
) WHERE inn IS NOT NULL LIMIT 100
```

Обратите внимание LIMITвынесен наружу, так как нужно получить 100 записей.

Рассмотрим все поддерживаемые варианты.

Представим, что в основной таблице есть одна строка `a:1, b:2` А во второй таблице есть 3 строки `c:3, d:4`, `c:null, d:6`, `c:3, d:null`

Ключевое слово	Описание	Множественный	Пример входа	Что на выходе
FIELDS	Экспорт полей как есть	нет	`SELECT a, b, (SELECT FIELDS c, d ... LIMIT 1)`	`a:1,b:2,c:3,d:4`
ARRAYS	Каждое поле представлено массивом своих значений, включая NULL	да	`SELECT a, b, (SELECT ARRAYS c, d ...)`	`a:1,b:2,c:[3,NULL,3],d:[4,6,NULL]`
SETS	Подобно ARRAYS, но только уникальные значения без NULL	да	`SELECT a, b, (SELECT SETS c, d ...)`	`a:1,b:2,c:[3],d:[4,6]`
OBJECT	Представить как один объект JSON	нет	`SELECT a, b, (SELECT OBJECT c, d ... LIMIT 1) as f`	`a:1,b:2,f:{c:3,d:4}`
OBJECTS	Представить как массив объектов JSON	да	`SELECT a, b, (SELECT OBJECTS c, d ... LIMIT 1) as f`	`a:1,b:2,f:[{c:3,d:4},{c:null,d:6},{c:3,d:null}]`

Внимание! Нужно осторожно пользоваться многотабличными запросами, так как очень легко сделать ошибку, которая повлечет затраты в платных источниках или слишком долгое время ожидания консультируйтесь с экспертами.

Управление режимом фетчинга в SDQL

---

*Внимание! Для большинства задач прямое управление режимом фетчинга **НЕ требуется**!*

Системная инструкция `@F` предназначена для перекрытия действующего по умолчанию режима фетча (считывания данных с сервера и подготовки их к отправки на сервере).

Синтаксис в стиле `@<LETTER> ...` в SDQL всегда указывает, что это нечто, имеющее очень специфическое техническое назначение, не влияющее на сами данные и не предназначенное для использования без точного понимания целей, опции для более широкого профиля заносятся в конструкцию `WITH ...`

По умолчанию для полностью кэшированных в памяти таблиц используется простая стратегия `All` которая сразу возвращает все записи в момент поступления запроса (синхронно)

Для больших и “ленивых” таблиц используется возврат чанками по min-fetch-size .. fetch-size записей с шагом не более fetch-time на ожидание записей сверх минимума (1).

При этом различается оптимистичное и пессимистичное поведение. При оптимистическом поведении производится ожидание первого чанка и минимальный размер чанка устанавливается в 1 При пессимистической оценке - первый чанк сразу возвращается пустым, чтобы подтвердить прием запроса, при этом минимальный чанк 0, чтобы при выходе на таймаут выдавать пустой батч

С точки зрения производительности - чем меньше коммуникации с сервером тем лучше, то есть передача по одной записи - не эффективна, в то же время передача всего сразу одним куском невозможна, если данных много, также хотелось бы видеть данные по мере их поступления даже, если не все данные еще готовы. Поэтому и выстраивается сложная логика фетчинга, которая старается отдать побыстрее, но при этом как можно более значительными кусками.

То есть регулировка по умолчанию (оптимистичная):

1. `first-mode`: `F` (first)
2. `fetch-time`: `10000` (ms)
3. `fetch-size`: `200`
4. `min-fetch-size`: `1`

Регулировка для заведомо медленных запросов (пессимистическая)

5. `first-mode`: `N` (first empty)
6. `fetch-time`: `30000` (ms)
7. `fetch-size`: `20`
8. `min-fetch-size`: `1`

Допустим кассандра отдает первую запись через 2 секунды, вторую через 3 и затем еще через секунду еще 200 записей примерно с разницей в 20 ms - в этом случае при выполнении запроса будет пауза 2 секунды и отдача 1 записи (работают fetch-first, min-fetch-size, fetch-time), затем будет прочитана еще 1 запись (сработает min-fetch-size, fetch-time) затем вернется 100 записей (попадут в одно окно fetch-time и fetch-size) затем остаток

Данные настройки можно перекрыть опцией @F

Опция @F оставляет в неприкосновенности дефолты и при этом выставляет нужные параметры:

1. `SELECT ... @F N` -> `fetch-first: none` - первый батч пустой
2. `SELECT ... @F F` -> `fetch-first: first` - ждет первого не пустого батча
3. `SELECT ... @F A` -> `fetch-first: all` - чтение всех записей - **внимание! может положить сервер, не использовать, если нет понимания зачем.**
4. `SELECT ... @F S 200` -> `fetch-size: 200` (меняет дефолт 100 на 200, так как в рамках 1000 - то допустимо)
5. `SELECT ... @F S 200 M 50` -> `fetch-size: 200, min-fetch-size: 50` (не взирая на таймаут будет минимум по 50 в пачке возвращать)
6. `SELECT ... @F T 5000` -> `fetch-time:5000 (5s)` - разрешает подольше (1s->5s) ждать следующую пачку
7. `SELECT ... @F S 1000 M 0 T 50000` -> `fetch-time:50000 (50s), fetch-size: 1000, min-fetch-size: 0` - позволяет дольше ждать отдельной пачки, пачка максимального размера, но при этом если данных нет на 50 секунде, но еще будут - вернуть просто пустой чанк

Считаем, что это требуется при исследовательских задачах и выгрузках при тюнингах особо массивных запросов на больших таблицах при потребности одновременно в максимизации быстродействия. Также это позволяет настроить режим экспресс оценки запросов без их полного “выгребания” до конца.

Отличия от ANSI SQL

Здесь сведены принципиальные отличия языка SDQL от стандартов, принятых в SQL.

Естественно, что SDQL реализует только очень небольшую часть всех возможностей SQL, здесь описаны именно отличия при которых функция SQL реализована, но с изменением в каких-то основных принципах.

В этом документе описываются не все фичи, а именно те, которые похожи или прямо совпадают по синтаксису или идеям с SQL, при этом реализованы с принципиальными отличиями.

Так например, индексаторы не являются каким-то отклонением или расширением, так как ANSI не регламентирует как организован синтаксис для работы с объектами JSON или массивами.

Соглашения

1. В ANSI SQL нет однозначного указания о приведении типов, в SDQL принято решение использовать динамическую типизацию, тут есть резон сравнить с PG
   - в Postgres - иногда типы приводятся сами и выглядят как динамические, а иногда требуют явного каста в нужный тип, тип сложного значения определяется прежде всего параметрами и вторично примененным оператором, в MSSQL вообще + может означать как сложение, так и конкатенацию
   - в SDQL тип выражения **всегда** кроме редких исключений определяется **оператором** и Spectrum Data Gate пытается привести к нужным типы параметров, так + это всегда “сложение”, “число” - `'2' + 2 -> 4`, `'2' + '2' -> 4`, `'h' + '1' -> error`
   - благодаря этому мы можем использовать legacy данные в мешанине, но при этом четко выражать какие мы ожидаем действия и результаты
2. Исходя из JS-подобной типизации у нас все сводимо к `bool` поэтому такой запрос `select * from t where code` означает примерно `select * from t where code is not null and code != ''`
3. по традиции SpectrumData у нас также есть понятие “эффективного null” - например `is null` вернет `true` для пустых строк, пробельных строк и строки с надписью `null` - ну то есть если что-то в реальной жизни всеми понимается как `null` и его выражает, то Spectrum Data Gate это также считает `null`

Расширения

1. Поддерживается вызов функций в объектной нотации, любых
   - в ANSI SQL можно только `len(x)`
   - в SDQL можно и `len(x)` и `x.len()`, это касается и функций с несколькими параметрами, `if_null(x, y)` то же самое, что `x.if_null(y)`
2. Поддерживается подстановка выражений из SELECT в другие части запросов:
   - в ANSI SQL можно только так `SELECT some_field as f, (x + y) as sum WHERE some_field > 1 and (x + y ) < 100`
   - в SDQL это же можно так `SELECT some_field as f, (x + y) as sum WHERE &f > 1 AND &sum < 100`
3. SDQL рассматривает результат своей работы как “поток записей”, а не как множество, соответственно мы предполагаем, что есть понятие “номера записи”, “предыдущей записи” и т.п., поэтому в базовом наборе функций есть “поточные агрегаторы” - это агрегирующие функции без группировки, но использующие историю всего запроса, например:
   - `rownumber()` (номер строки)
   - `cr_sum(value)` (текущая накопленная сумма по value)
   - `cr_prev(value)` (значение value в предыдущей записи потока)

Уникальные концепции

В SDQL прямо на уровень языка было вынесено несколько концепций не характерных для SQL.

Большинство этих новшеств связаны с функциями, которые в других диалектах требуют использования на уровне протокола или отдельными командами и так как это носит достаточно специализированный и технический характер, такие конструкции имеют префикс @ в своем ключевом слове и обычно “не-SQL” синтаксис внутри

1. @D информация об используемом драйвере - для обеспечения прямой и обратной совместимости и корректного логирования ошибок
2. @R установка ограничений на роль для конкретного запроса - для отладки ACL с ограниченными правами
3. @F настройка поведения `fetcher` - влияет на интенсивность опроса СУБД и на объем данных в пакете

Также к особой концепции можно отнести “явные” и “защищенные” поля, которые влияют на поведение * - по сути эти управление видимостью полей

1. Явные - видны только если их указать явно в `select` или используя `WITH EXPLICIT_FIELDS`
2. Защищенные - видны только если есть особые права на доступ к ним

Ограничения

1. Строгое ограничение на уникальность полей
   - в ANSI SQL можно `select x, x, a as b, c as b`
   - в SDQL так нельзя, все поля должны быть уникальны `select x, x as x1, a as b, c as b2` - вот так можно
   - **Именно это свойство позволяет использовать ссылки на формулы &xxx**
2. Из этого следует измененное поведение *, например для таблицы с полями `a, b, c`
   - в ANSI SQL запрос `select a, *` вернет `a,a,b,c`
   - в SDQL запрос `select a, *` вернет `a,b,c` (по альясам, то есть `select a as x, *` вернет `x, a, b, c`)

Специфика работы с ключами

Одной из ключевых особенностей Spectrum Data Gate является возможность обвязки в качестве REST сервисов, включая источники SpectrumData

Однако очевидно, что это возможно сделать, только с некоторыми ограничениями, которые присущи конкретному сервису.

Это касается прежде всего того - какие поля являются ключевыми для данного REST сервиса.

Для этого в Spectrum Data Gate есть понятие “логического ключа сервиса”.

Например, сервис http://someservice подключен к Spectrum Data Gate как таблица `services.some`.

Известно, что сервис всегда требует на вход `vin` и использует его в формате:

```
http://someservice/api/getdata?vin={vin}
```

то это значит, что запросы **обязаны** включать в себя WHERE с полем `vin` и только с оператором = (равенство), так как иначе отсутствует ключевая информация для вызова сервиса.

то есть `vin` это **логический ключ** в рамках данного сервиса.

Соответственно такой запрос будет обработан:

```
select * from services.some where vin = 'Z94CB41AAGR323319'
```

А такие запросы нет

```
-- тут проблема в том, что оператор `>` сервисом не поддерживается
select * from services.some where vin > 'Z94CB41AAGR323319'

-- тут проблема в том, что поле `year` не является ключевым для сервиса,
-- а нужного ключа не предоставлено
select * from services.some where year = 2020
```

Например:

```
-- этих полей явно не достаточно для выполнения запроса на источник ФССП
SELECT * FROM fssp.source WHERE
    last_name = 'Примеров' AND
    debt_sum  > 100000
```

приведет к ошибке:

```
SQL Error [22000]: UNKNOWN: (400)[ERDS0010] required key not found -
[fssp.source]
```

В случае работы с кассандрой есть понятия ключ, который нужно использовать в запросе, есть возможность перебрать в ней все данные при при помощи опций: RND_FULL_SCAN, SEQ_FULL_SCAN. Однако использование данных опций будет не у всех пользователей Spectrum Data Gate, у кого права нет обязаны использовать тот или иной ключ.

Пример:

```
SELECT *
FROM xf.bfo AS b
WHERE b.financial_results_report_obj['revenue.on_report_year'] > 100000000
limit 10
WITH rnd_full_scan;
```

После выполнения запроса с данной опцией будет произведет полный обход таблицы кассандры, и результат будет получен.

У многих источников бывает не один а несколько ключевых наборов, например:

1. по `fio + birth`
2. по `pass`
3. по `inn`

В этом случае будет использоваться тот ключ, для которого есть условия, если есть условия для нескольких ключей - будет взят первый (как по порядку приоритетности)

Ключевые и не ключевые условия

Spectrum Data Gate умеет анализировать используемые условия и правильно определить, какие из них должны быть обработаны прямо на REST (например в составе ключа), а какие применены уже затем, для фильтрации результата.

Например, рассмотрим план запроса:

```
EXPLAIN
SELECT * FROM fssp.source WHERE
    last_name = 'Примреов' AND
    first_name = 'Пример' AND
    patronymic = 'Примерович' AND
    birth = '09.12.1984' AND
    debt_sum  > 100000

[SDQL0200](200)
--------- Query plan ---------
SELECT: fld:[ep_number<in:epNumber>],
╠·····fld:[last_name<in:lastName>],
```

```
╠┈┈┈.................................
╚═FILTER: debtSum<out:debt_sum> > 100000
  ╚═PRIMARY SELECT: tbl:[fssp.source] ->
    ╠┈┈┈SELECT *
    ╠┈┈┈  WHERE f<q:lastName<out:last_name>> = 'Примреов' AND
    ╠┈┈┈    i<q:firstName<out:first_name>> = 'Пример' AND
    ╠┈┈┈    o<q:patronymic> = 'Примерович' AND
    ╚┈┈┈    bDate<q:entBdate<out:birth>> = '09.12.1984'
```

Сочетание `f + i + o + bDate` является ключевым для источника ФССП. Запрос будет в итоге разделен на 2 фазы:

1. На `PRIMARY SELECT` будут использоваться поля для ФИО и ДР
2. И уже потом на фазе `FILTER` будет применен фильтр по сумме долга

Явные и защищаемые поля

В Spectrum Data Gate есть механизм по ограничению видимости полей в Spectrum Data Gate.

Один механизм - “явные поля” (EXPLICIT FIELDS), а второй - “защищаемые поля”.

Явные поля сделаны в эргономических целях, а защищаемые понятно в целях безопасности

Явные поля

Концепция “явных полей” - проста - если вы просто сделаете запрос с *:

```
select * from sys.tables
```

там не будет технического поля `er_bank_code`, чтобы оно появилось в запросе, надо или **явно** указать в запросе:

```
select catalog, er_bank_code from sys.tables
```

Внимание! Синтаксис `select *, er_bank_code` пока не поддерживается!

Также есть опция `WITH EXPLICIT_FIELDS` которая включит все явные поля в *

```
select * from sys.tables with explicit_fields
```

Защищаемые поля

Обычно защищаемые поля еще и EXPLICIT и содержат какую-то особо «чувствительную» информацию.

Доступ к ним раздается через ACL и по умолчанию только роль `admin` имеет к ним доступ.

Пользователь без таких полномочий не увидит этих полей в списке, при попытке все же выполнить SELECT по этому полю будет логируемая ошибка `403` кода.

В таблице полей отмечены тегом `sec`

Реестр ошибок Spectrum Data Gate

Общая таблица ошибок

Код	HTTP	Название	Сообщение	Базовая	Конструктор
AUTH0010	500	auth.InternalError	*internal auth error*	☑	`auth.NewInternalError(obj string, err error, message string, args ...any)`
AUTH0020	401	auth.NoCredentialsError	*no credentials*	☐	
AUTH0030	400	auth.InvalidHeaderError	*invalid Authorization format*	☑	`auth.NewInvalidHeaderError(err error, pattern string, args ...any)`
AUTH0040	401	auth.NotAuthenticatedError	*user not authenticated*	☑	`auth.NewNotAuthenticatedError(obj string, pattern string, args ...any)`
AUTH0050	403	auth.ForbiddenError	*deny*	☑	`auth.NewForbiddenError(obj string, user string, target string, permissions string)`
AUTH0060	403	auth.WrongSessionUserError	*session principal diffs from incoming.*	☑	`auth.NewWrongSessionUserError(incomeUser string, sessionUser string)`
CERR0010	500	errs.NotSupportedError	*not supported feature*	☑	`errs.NewNotSupportedError(obj string, src string, feature string, cause NotSupportedCause, pattern string, args ...any)`
CERR0020	500	errs.InternalError	*internal error*	☑	`errs.NewInternalError(err error, component string, pattern string, args ...any)`
CERR0030	500	errs.TypeConvertError	*type convert error*	☑	`errs.NewTypeConvertError(value any, targetType string, pattern string, args ...any)`

Код	HTTP	Название	Сообщение	Базовая	Конструктор
ERDS0010	400	datasets.NoKeyMatchError	*required key not found*	☑	`datasets.NewNoKeyMatchError(table string, pattern string, args ...any)`
ERDS0020	404	datasets.NoFieldMatchError	*required field not found*	☑	`datasets.NewNoFieldMatchError(table string, field string, pattern string, args ...any)`
ERDS0030	500	datasets.InvalidDatasetPlatform	*invalid dataset platform*	☑	`datasets.NewInvalidDatasetPlatform(entityCode string, datasetCode string, platform &{v1 Bank_Platform})`
ERDS0040	500	datasets.InvalidCassandraKeyConfiguration	*invalid cassandra key configuration*	☑	`datasets.NewInvalidCassandraKeyConfiguration(entityCode string, datasetCode string, message string)`
EREP0000	500	entreader.InvalidRootError	*invalid root folder for er*	☑	`entreader.NewInvalidRootError(root string, external error)`
EREP0001	500	entreader.GeneralFileError	*general file error*	☑	`entreader.NewGeneralFileError(external error)`
EREP0002	500	entreader.GeneralYamlError	*general yaml error*	☑	`entreader.NewGeneralYamlError(external error)`
EREP0003	500	entreader.EmptyFileError	*empty er file error*	☐	
EREP0111	500	entreader.InvalidPackageError	*invalid package name*	☑	`entreader.NewInvalidPackageError(actual string, expected string)`
EREP0112	500	entreader.InvalidCodeError	*invalid entity code*	☑	`entreader.NewInvalidCodeError(code string)`
EREP0113	500	entreader.InvalidLinkTargetError	*invalid link target*	☑	`entreader.NewInvalidLinkTargetError(linkIndex int, linkTarget string)`
EREP0114	500	entreader.MissingPackageError	*package is null or blank*	☐	

Код	HTTP	Название	Сообщение	Базовая	Конструктор
EREP0115	500	entreader.MissingLinkTargetError	*missing link target*	☑	`entreader.NewMissingLinkTargetError(linkIndex int)`
SDQL0010	400	sdql.ParsingError	*invalid SDQL query*	☑	`sdql.NewParsingError(err error, pattern string, args ...any)`
SDQL0050	500	sdql.CompileError	*SDQL query compilation error*	☑	`sdql.NewCompileError(err error, pattern string, args ...any)`
SDQL0060	500	sdql.EvalError	*SDQL expression eval error*	☑	`sdql.NewEvalError(err error, pattern string, args ...any)`
SDQL0070	500	sdql.CastError	*cannot cast value*	☑	`sdql.NewCastError(err error, from any, toType string, castContext string)`
SVDB0099	400	svdb.UidIsNullError	*uid is null.*	☑	`svdb.NewUidIsNullError(err error, pattern string, args ...any)`
SVDB0100	404	svdb.EntityNotFoundError	*entity not exist on server.*	☐	
SVDB0101	500	model.NotSupportedValueTypeError	*not supported type*	☑	`model.NewNotSupportedValueTypeError(value any, pattern string, args ...any)`
SVDB0102	500	model.InvalidArrayItemError	*invalid array item*	☑	`model.NewInvalidArrayItemError(err error, idx int, pattern string, args ...any)`
SVDB0103	500	model.InvalidObjItemError	*invalid obj item*	☑	`model.NewInvalidObjItemError(err error, field string, pattern string, args ...any)`
SVDB0110	500	model.InvalidFieldError	*invalid field*	☑	`model.NewInvalidFieldError(err error, name string, pattern string, args ...any)`
SVDB0211	404	planner.TableNotFoundError	*table not found*	☑	`planner.NewTableNotFoundError(tableName string)`

Код	HTTP	Название	Сообщение	Базовая	Конструктор
SVDB0212	404	planner.FieldNotFoundError	*field not found*	☑	`planner.NewFieldNotFoundError(tableName string, fieldName string)`
SVDB0213	400	planner.InvalidKeyError	*invalid key*	☑	`planner.NewInvalidKeyError(pattern string, args ...any)`
SVDB0216	404	planner.FunctionNotFoundError	*function not found*	☑	`planner.NewFunctionNotFoundError(e *sdql.Expression)`
SVDB0240	403	planner.DenyError	*deny by acl*	☑	`planner.NewDenyError(subjectType string, subject string, permission string, pattern string, args ...any)`
SVDB0250	400	planner.NonConsistentQueryError	*inconsistent error*	☑	`planner.NewNonConsistentQueryError(pattern string, args ...any)`
SVDB2001	500	executor.InvalidOperationError	*invalid operation*	☑	`executor.NewInvalidOperationError(op &{coreIOperation}, pattern string, args ...any)`
SVDB4001	500	tables.NoCredentialsError	*absent credentials*	☑	`tables.NewNoCredentialsError(table string)`
SVDB4002	500	tables.NoConnectionURLError	*absent connection URL*	☑	`tables.NewNoConnectionURLError(table string)`
SVDB4003	500	tables.InvalidConnectionURLError	*invalid connection URL*	☑	`tables.NewInvalidConnectionURLError(table string)`
SVDB4010	500	tables.FieldMatchError	*error in predicate evaluation*	☑	`tables.NewFieldMatchError(err error, fld string, exp *sdql.Expression, pattern string, args ...any)`
SVDB4020	400	tables.UserDataError	*error caused by invalid user data*	☑	`tables.NewUserDataError(err error, pattern string, args ...any)`

Код	HTTP	Название	Сообщение	Базовая	Конструктор
SVDB6000	404	core.BackStoreNotFoundError	*local session table not found*	☑	`core.NewBackStoreNotFoundError(tableName string)`
SVDB6010	400	core.BackStoreAlreadyExistsError	*local table already exists*	☑	`core.NewBackStoreAlreadyExistsError(tableName string)`
SVDB6020	400	core.BackStoreCountLimitError	*try append local table when limit riched*	☑	`core.NewBackStoreCountLimitError(limit int)`
SVDB6040	400	core.BackStoreInvalidSchemaError	*invalid record for schema*	☑	`core.NewBackStoreInvalidSchemaError(message string, args ...any)`
SVDB6050	500	core.BackStoreExternalError	*error in external system*	☑	`core.NewBackStoreExternalError(err error, message string, args ...any)`
SVDB7000	500	server.FatalGrpcStartError	*fatal gRPC start error*	☑	`server.NewFatalGrpcStartError(err error, pattern string, args ...any)`
SVDB7001	500	server.PrincipalNotSetInMetadataError	*principal not set in context*	☐	
SVDB7002	400	server.FetchWithoutCursor	*try to fetch without cursor (may be without /ExecuteQuery or after failed /ExecuteQuery)*	☐	
SVDB7003	500	server.TLSError	*invalid TLS*	☑	`server.NewTLSError(err error, pattern string, args ...any)`

Код	HTTP	Название	Сообщение	Базовая	Конструктор
SVDB7004	500	bus.FatalBusStartError	*fatal bus start error*	☑	`bus.NewFatalBusStartError(err error, pattern string, args ...any)`
SVDB7005	500	bus.GeneralBusError	*bus error*	☑	`bus.NewGeneralBusError(err error, pattern string, args ...any)`
SVDB7006	500	bus.ProcessBusMsgError	*bus message processing error*	☑	`bus.NewProcessBusMsgError(err error, pattern string, args ...any)`
SVDB7007	500	bus.HandleBusMsgError	*bus message handling finished with error*	☑	`bus.NewHandleBusMsgError(err error, pattern string, args ...any)`
SVDB7008	500	bus.GetStreamInfoError	*can't get info about/from stream*	☑	`bus.NewGetStreamInfoError(err error, pattern string, args ...any)`
SVDB7009	400	systables.NotClusterModeJSFetch	*can't get nats channel in SINGLENODE mode*	☐	
SVDB7010	500	server.InvalidSessionUIDError	*invalid session uid*	☑	`server.NewInvalidSessionUIDError(sessionUID string, pattern string, args ...any)`
SVDB7011	500	server.RestCallProcessingError	*rest request processing error*	☑	`server.NewRestCallProcessingError(err error, pattern string, args ...any)`
SVDB8000	500	driver.GeneralDriverError	*general native client error*	☑	`driver.NewGeneralDriverError(err error, pattern string, args ...any)`

Код	HTTP	Название	Сообщение	Базовая	Конструктор
SVDB8001	500	driver.NoSessionUIDInServerHeadersError	*no sessionuid in server headers*	☐	

AUTH0010 auth.InternalError - internal auth error

Код: AUTH0010, Http: 500

внутренняя ошибка системы авторизации (обертка над внешней)

Является базовой ошибкой, должна использоваться через конструктор:

обертка над переданной ошибкой [err] при аутентификации в контексте работы объекта с условным именем [obj] можно передать как [pattern] + args, если [err] == nil, то это будет просто уточненное сообщение

```
auth.NewInternalError(obj string, err error, message string, args ...any)
```

AUTH0020 auth.NoCredentialsError - no credentials

Код: AUTH0020, Http: 401

ошибка формируется, если в контексте требующем креденций они не были переданы в запросе

Является SINGLTON ошибкой, используется прямо как переменная

AUTH0030 auth.InvalidHeaderError - invalid Authorization format

Код: AUTH0030, Http: 400

ошибка формата хидера Authorization

Является базовой ошибкой, должна использоваться через конструктор:

обертка над переданной ошибкой [err] можно передать как [pattern] + , если [err] == nil, то это будет просто уточненное сообщение

```
auth.NewInvalidHeaderError(err error, pattern string, args ...any)
```

AUTH0040 auth.NotAuthenticatedError - user not authenticated

Код: AUTH0040, Http: 401

ошибка формируется если

1. креденции были переданы в запросе (иначе `auth.NoCredentialsError`)

2. кри этом не удалось с их помощью распознать и аутентифицировать пользователя

Является базовой ошибкой, должна использоваться через конструктор:

в контексте работы объекта с условным именем [obj] и с каким внутренним сообщением об ошибке [pattern] + [args]

```
auth.NewNotAuthenticatedError(obj string, pattern string, args ...any)
```

AUTH0050 auth.ForbiddenError - deny

Код: AUTH0050, Http: 403

ошибка формируется если:

1. креденции были переданы в запросе и аутентифицированы

2. при этом пользователю не хватает каких-то полномочий

Является базовой ошибкой, должна использоваться через конструктор:

в контексте работы объекта с условным именем [obj] с каким-то защищаемым объектом [target] и каким-то описанием пермиссий [permissions] для пользователя [user]

```
auth.NewForbiddenError(obj string, user string, target string, permissions string)
```

AUTH0060 auth.WrongSessionUserError - session principal diffs from incoming.

Код: AUTH0060, Http: 403

ошибка формируется когда пользователь сессии не совпадает с пользователем, переданным в запросе

Является базовой ошибкой, должна использоваться через конструктор:

где [incomeUser] в качестве входящего пользователя и [sessionUser] в качестве пользователя сессии

```
auth.NewWrongSessionUserError(incomeUser string, sessionUser string)
```

CERR0010 errs.NotSupportedError - not supported feature

Код: CERR0010, Http: 500

Возникает при DEPRECATE, FUTURE и иных вариантах вызова некорректных в данном контексте фичей

Является базовой ошибкой, должна использоваться через конструктор:

С дополнительными параметрами:

1. obj“ - объект, функция или иной компонент, который пытался **использовать** фичу

2. src- объект, у которого пытались запросить фичу

3. feature- название фичи

4. cause- тип причины отказа в поддержке

5. pattern`+args - дополнительное сообщение

```
errs.NewNotSupportedError(obj string, src string, feature string, cause NotSupportedCause, pattern string, args ...any)
```

CERR0020 errs.InternalError - internal error

Код: CERR0020, Http: 500

обертки над внешними (своими или чужими) ошибками, когда надо явно выделить, что это ошибка считается именно внутренним сбоем, в NewInternalError можно отметить какой именно компонент

Является базовой ошибкой, должна использоваться через конструктор:

с указанием

1. err - собственно внешняя ошибка, опциональная

2. component - при работе какого компонента возникла ошибка

3. pattern + args - дополнительное сообщение к ошибке (опциональное)

```
errs.NewInternalError(err error, component string, pattern string, args ...any)
```

CERR0030 errs.TypeConvertError - type convert error

Код: CERR0030, Http: 500

ошибка преобразования типов

Является базовой ошибкой, должна использоваться через конструктор:

с указанием

1. value - значение, которое пытались преобразовать

2. targetType - целевой тип

3. pattern + args - сообщение к ошибке

```
errs.NewTypeConvertError(value any, targetType string, pattern string, args ...any)
```

ERDS0010 datasets.NoKeyMatchError - required key not found

Код: ERDS0010, Http: 400

в случае если dataset.GetMatchedKey не нашел ключа для таблицы, которая этого требует. в случае, если был выполнен запрос на таблицу касандры, причина отсутствия ключевого поля в условии запроса решением будет являться использование опций RND_FULL_SCAN, SEQ_FULL_SCAN.

Является базовой ошибкой, должна использоваться через конструктор:

для указанной таблицы [table] и с дополнительным пояснением

```
datasets.NewNoKeyMatchError(table string, pattern string, args ...any)
```

ERDS0020 datasets.NoFieldMatchError - required field not found

Код: ERDS0020, Http: 404

не найдено запрошенное поле

Является базовой ошибкой, должна использоваться через конструктор:

для указанной таблицы [table], поля [field] и с дополнительным пояснением

```
datasets.NewNoFieldMatchError(table string, field string, pattern string,
args ...any)
```

ERDS0030 datasets.InvalidDatasetPlatform - invalid dataset platform

Код: ERDS0030, Http: 500

ошибка противоречия переданного соединения и указанной платформы, в норме используется через [NewInvalidDatasetPlatform], вкладывается в описатель сущности

Является базовой ошибкой, должна использоваться через конструктор:

с указанием кода датасета и ожидаемой платформы

```
datasets.NewInvalidDatasetPlatform(entityCode string, datasetCode string,
platform &{v1 Bank_Platform})
```

ERDS0040 datasets.InvalidCassandraKeyConfiguration - invalid cassandra key configuration

Код: ERDS0040, Http: 500

ошибка конфигурации ключа набора данных, в норме используется через [NewInvalidCassandraKeyConfiguration], вкладывается в описатель сущности

Является базовой ошибкой, должна использоваться через конструктор:

с указанием кода датасета и ожидаемой платформы

`datasets.NewInvalidCassandraKeyConfiguration(entityCode string, datasetCode string, message string)`

EREP0000 entreader.InvalidRootError - invalid root folder for er

Код: `EREP0000`, Http: `500`

ошибка, возникает при попытке использовать не корректную корневую директорию

Является базовой ошибкой, должна использоваться через конструктор:

Оборачивая имя переданной корневой директории и внешнюю ошибку

`entreader.NewInvalidRootError(root string, external error)`

EREP0001 entreader.GeneralFileError - general file error

Код: `EREP0001`, Http: `500`

базовая общая ошибка чтения файла ER, вкладывается в описатель файла, в норме используется через [NewGeneralFileError]

Является базовой ошибкой, должна использоваться через конструктор:

оборачивая внешнюю ошибку

`entreader.NewGeneralFileError(external error)`

EREP0002 entreader.GeneralYamlError - general yaml error

Код: `EREP0002`, Http: `500`

базовая общая ошибка при чтении YAML (неформатный YAML), вкладывается в описатель файла, в норме используется через [NewGeneralYamlError]

Является базовой ошибкой, должна использоваться через конструктор:

оборачивая внешнюю ошибку

`entreader.NewGeneralYamlError(external error)`

EREP0003 entreader.EmptyFileError - empty er file error

Код: `EREP0003`, Http: `500`

пустой файл в ER, вкладывается в описатель файла

Является SINGLTON ошибкой, используется прямо как переменная

EREP0111 entreader.InvalidPackageError - invalid package name

Код: EREP0111, Http: 500

базовая ошибка неправильного имени пакета, в норме используется через [NewInvalidPackage], вкладывается в описатель файла

Является базовой ошибкой, должна использоваться через конструктор:

с подсказкой на ожидание

```
entreader.NewInvalidPackageError(actual string, expected string)
```

EREP0112 entreader.InvalidCodeError - invalid entity code

Код: EREP0112, Http: 500

базовая ошибка неправильного кода сущности, в норме используется через [NewInvalidCode], вкладывается в описатель сущности

Является базовой ошибкой, должна использоваться через конструктор:

с подсказкой на ожидание

```
entreader.NewInvalidCodeError(code string)
```

EREP0113 entreader.InvalidLinkTargetError - invalid link target

Код: EREP0113, Http: 500

базовая ошибка неправильного кода связи, в норме используется через [NewInvalidLinkTarget], вкладывается в описатель сущности

Является базовой ошибкой, должна использоваться через конструктор:

с точным указанием какая связь ошибочная

```
entreader.NewInvalidLinkTargetError(linkIndex int, linkTarget string)
```

EREP0114 entreader.MissingPackageError - package is null or blank

Код: EREP0114, Http: 500

ошибка полного отсутствия настройки пакета, синглтон, вкладывается в описатель сущности

Является SINGLTON ошибкой, используется прямо как переменная

EREP0115 entreader.MissingLinkTargetError - missing link target

Код: EREP0115, Http: 500

базовая ошибка отсутствующего кода связи, в норме используется через [NewMissingLinkTarget], вкладывается в описатель сущности

Является базовой ошибкой, должна использоваться через конструктор:

с точным указанием какая связь ошибочная

```
entreader.NewMissingLinkTargetError(linkIndex int)
```

SDQL0010 sdql.ParsingError - invalid SDQL query

Код: SDQL0010, Http: 400

возникает при ошибке парсинга запроса SDQL

Является базовой ошибкой, должна использоваться через конструктор:

обертка над переданной ошибкой [err] можно передать как [pattern] + , если [err] == nil, то это будет просто уточненное сообщение

```
sdql.NewParsingError(err error, pattern string, args ...any)
```

SDQL0050 sdql.CompileError - SDQL query compilation error

Код: SDQL0050, Http: 500

возникает при ошибке компиляции запроса SDQL

Является базовой ошибкой, должна использоваться через конструктор:

обертка над переданной ошибкой [err] можно передать как [pattern] + , если [err] == nil, то это будет просто уточненное сообщение

```
sdql.NewCompileError(err error, pattern string, args ...any)
```

SDQL0060 sdql.EvalError - SDQL expression eval error

Код: SDQL0060, Http: 500

возникает при ошибке вычислений условий SDQL

Является базовой ошибкой, должна использоваться через конструктор:

обертка над переданной ошибкой [err] можно передать как [pattern] + , если [err] == nil, то это будет просто уточненное сообщение

`sdql.NewEvalError(err error, pattern string, args ...any)`

SDQL0070 sdql.CastError - cannot cast value

Код: SDQL0070, Http: 500

все ошибки приведения типов значений

Является базовой ошибкой, должна использоваться через конструктор:

с указанием

1. исходной ошибки [err]
2. исходного значения [from]
3. целевого типа [toType]
4. контекста каста [castContext]

`sdql.NewCastError(err error, from any, toType string, castContext string)`

SVDB0099 svdb.UidIsNullError - uid is null.

Код: SVDB0099, Http: 400

Uid имеет значение nil

Является базовой ошибкой, должна использоваться через конструктор:

оборачивает внешний [err] с дополнительным сообщение pattern + args

`svdb.NewUidIsNullError(err error, pattern string, args ...any)`

SVDB0100 svdb.EntityNotFoundError - entity not exist on server.

Код: SVDB0100, Http: 404

сбой протокола, отсутствие идентификатора сессии в ответе сервера

Является SINGLTON ошибкой, используется прямо как переменная

SVDB0101 model.NotSupportedValueTypeError - not supported type

Код: SVDB0101, Http: 500

ошибка при формировании v1.Record - использование не поддерживаемого типа данных

Является базовой ошибкой, должна использоваться через конструктор:

с указанием того какой тип не поддерживается (для [value]) и с опциональным дополнительным сообщением (pattern + args)

`model.NewNotSupportedValueTypeError(value any, pattern string, args ...any)`

SVDB0102 model.InvalidArrayItemError - invalid array item

Код: SVDB0102, Http: 500

ошибка при формировании Array полей v1.Record

Является базовой ошибкой, должна использоваться через конструктор:

с указанием причины (err), индекса элемента массива (idx) и дополнительного сообщения (pattern+args)

`model.NewInvalidArrayItemError(err error, idx int, pattern string, args ...any)`

SVDB0103 model.InvalidObjItemError - invalid obj item

Код: SVDB0103, Http: 500

ошибка при формировании Object полей v1.Record

Является базовой ошибкой, должна использоваться через конструктор:

с указанием причины (err), имени поля (field) и дополнительного сообщения (pattern+args)

`model.NewInvalidObjItemError(err error, field string, pattern string, args ...any)`

SVDB0110 model.InvalidFieldError - invalid field

Код: SVDB0110, Http: 500

ошибка уровня именованного поля

Является базовой ошибкой, должна использоваться через конструктор:

с указанием имени поля [name], опциональной внешней ошибкой [err] и дополнительным комментарием (pattern + args)

`model.NewInvalidFieldError(err error, name string, pattern string, args ...any)`

SVDB0211 planner.TableNotFoundError - table not found

Код: SVDB0211, Http: 404

ошибка при попытке получить данные из таблицы, которой нет в реестре таблиц

Является базовой ошибкой, должна использоваться через конструктор:

с указанием имени таблицы `tableName`

```
planner.NewTableNotFoundError(tableName string)
```

SVDB0212 planner.FieldNotFoundError - field not found

Код: SVDB0212, Http: 404

ошибка при попытке использования несуществующего поля

Является базовой ошибкой, должна использоваться через конструктор:

с указанием: 1. `tableName` - имя таблицы 2. `fieldName` - имя поля

```
planner.NewFieldNotFoundError(tableName string, fieldName string)
```

SVDB0213 planner.InvalidKeyError - invalid key

Код: SVDB0213, Http: 400

ошибка связанная с некорректным ключом запроса SdQL

Является базовой ошибкой, должна использоваться через конструктор:

```
planner.NewInvalidKeyError(pattern string, args ...any)
```

SVDB0216 planner.FunctionNotFoundError - function not found

Код: SVDB0216, Http: 404

ошибка при попытке использовать не существующие функции

Является базовой ошибкой, должна использоваться через конструктор:

с указанием контекстного выраженияе

```
planner.NewFunctionNotFoundError(e *sdql.Expression)
```

SVDB0240 planner.DenyError - deny by acl

Код: SVDB0240, Http: 403

ошибки, связанные с запретами со строны ACL и других средств безопасности

Является базовой ошибкой, должна использоваться через конструктор:

с указанием:

1. subjectType - тип предмета проверки
2. subject - предмет проверки полномочий
3. permission - требуемые полномочия
4. pattern + args - дополнительные комментарии

```
planner.NewDenyError(subjectType string, subject string, permission string, pattern string, args ...any)
```

SVDB0250 planner.NonConsistentQueryError - inconsistent error

Код: SVDB0250, Http: 400

различные проблемы функциональной некорректности запросов

Является базовой ошибкой, должна использоваться через конструктор:

с дополнительным сообщением

```
planner.NewNonConsistentQueryError(pattern string, args ...any)
```

SVDB2001 executor.InvalidOperationError - invalid operation

Код: SVDB2001, Http: 500

возникает при рассогласовании переданного плана с ожидаемым на стороне executor

---

Является базовой ошибкой, должна использоваться через конструктор:

с указанием

1. операции [op]
2. дополнительным сообщением pattern+args

```
executor.NewInvalidOperationError(op &{core IOperation}, pattern string, args ...any)
```

SVDB4001 tables.NoCredentialsError - absent credentials

Код: SVDB4001, Http: 500

ошибка при обращении к таблице без обязательных креденций

Является базовой ошибкой, должна использоваться через конструктор:

с указанием кода таблицы

`tables.NewNoCredentialsError(table string)`

SVDB4002 tables.NoConnectionURLError - absent connection URL

Код: SVDB4002, Http: 500

ошибка при обращении к таблице без обязательного соединения

Является базовой ошибкой, должна использоваться через конструктор:

с указанием кода таблицы

`tables.NewNoConnectionURLError(table string)`

SVDB4003 tables.InvalidConnectionURLError - invalid connection URL

Код: SVDB4003, Http: 500

ошибка при получении невалидного соединения

Является базовой ошибкой, должна использоваться через конструктор:

с указанием кода таблицы

`tables.NewInvalidConnectionURLError(table string)`

SVDB4010 tables.FieldMatchError - error in predicate evaluation

Код: SVDB4010, Http: 500

ошибки при применении предиката к полю

Является базовой ошибкой, должна использоваться через конструктор:

с указанием

1. внешней ошибки err

2. поля, для которого проводилась проверка [fld]

3. проверяемого выражения [exp]

4. дополнительных разъяснений pattern + args

```
tables.NewFieldMatchError(err error, fld string, exp *sdql.Expression,
pattern string, args ...any)
```

SVDB4020 tables.UserDataError - error caused by invalid user data

Код: SVDB4020, Http: 400

ошибки, связанные с неправильными данными пользователя, на его ответственности

Является базовой ошибкой, должна использоваться через конструктор:

с указанием дополнительной ошибки и текста

```
tables.NewUserDataError(err error, pattern string, args ...any)
```

SVDB6000 core.BackStoreNotFoundError - local session table not found

Код: SVDB6000, Http: 404

локальная таблица сессии не найдена

Является базовой ошибкой, должна использоваться через конструктор:

с указанием имени таблицы [tableName]

```
core.NewBackStoreNotFoundError(tableName string)
```

SVDB6010 core.BackStoreAlreadyExistsError - local table already exists

Код: SVDB6010, Http: 400

локальная таблица уже создана и идет попытка ее создания заново

Является базовой ошибкой, должна использоваться через конструктор:

с указанием имени таблицы [tableName]

```
core.NewBackStoreAlreadyExistsError(tableName string)
```

SVDB6020 core.BackStoreCountLimitError - try append local table when limit riched

Код: SVDB6020, Http: 400

попытка добавить в таблицу сверх общего лимита

Является базовой ошибкой, должна использоваться через конструктор:

с указанием имени таблицы [tableName]

```
core.NewBackStoreCountLimitError(limit int)
```

SVDB6040 core.BackStoreInvalidSchemaError - invalid record for schema

Код: SVDB6040, Http: 400

попытка добавить в таблицу запись, не подходящую по схеме

Является базовой ошибкой, должна использоваться через конструктор:

с указанием имени таблицы [tableName]

```
core.NewBackStoreInvalidSchemaError(message string, args ...any)
```

SVDB6050 core.BackStoreExternalError - error in external system

Код: SVDB6050, Http: 500

ошибки back-store во внешних хранилищах

Является базовой ошибкой, должна использоваться через конструктор:

с указанием имени таблицы [tableName]

```
core.NewBackStoreExternalError(err error, message string, args ...any)
```

SVDB7000 server.FatalGrpcStartError - fatal gRPC start error

Код: SVDB7000, Http: 500

обертка для внешних исключений, приводящих к фатальному сбою старта gRPC

Является базовой ошибкой, должна использоваться через конструктор:

оборачивает внешний [err] с дополнительным сообщение pattern + args

```
server.NewFatalGrpcStartError(err error, pattern string, args ...any)
```

SVDB7001 server.PrincipalNotSetInMetadataError - principal not set in context

Код: SVDB7001, Http: 500

ошибка отсутствия принципала в контексте, может быть при каком-то грубом сбое цепочки gRPC

Является SINGLTON ошибкой, используется прямо как переменная

SVDB7002 server.FetchWithoutCursor - try to fetch without cursor (may be without /ExecuteQuery or after failed /ExecuteQuery)

Код: SVDB7002, Http: 400

попытка Fetch из сессии без курсора - 95% неправильное использование API со стороны клиента

Является SINGLTON ошибкой, используется прямо как переменная

SVDB7003 server.TLSError - invalid TLS

Код: SVDB7003, Http: 500

ошибки gRPC, связанные TLS

Является базовой ошибкой, должна использоваться через конструктор:

оборачивает внешний [err] с дополнительным сообщение pattern + args

```
server.NewTLSError(err error, pattern string, args ...any)
```

SVDB7004 bus.FatalBusStartError - fatal bus start error

Код: SVDB7004, Http: 500

обертка для ошибок инициализации шины, приводящих к фатальному сбою старта шины кластера

Является базовой ошибкой, должна использоваться через конструктор:

оборачивает внешний err с дополнительным сообщением pattern + args

```
bus.NewFatalBusStartError(err error, pattern string, args ...any)
```

SVDB7005 bus.GeneralBusError - bus error

Код: SVDB7005, Http: 500

некая ошибка в работе шины, не специфичная - обертка над различными другими ошибками

Является базовой ошибкой, должна использоваться через конструктор:

оборачивает внешний [err] с дополнительным сообщение pattern + args

`bus.NewGeneralBusError(err error, pattern string, args ...any)`

SVDB7006 bus.ProcessBusMsgError - bus message processing error

Код: SVDB7006, Http: 500

ошибка обработки сообщения из шины

Является базовой ошибкой, должна использоваться через конструктор:

`bus.NewProcessBusMsgError(err error, pattern string, args ...any)`

SVDB7007 bus.HandleBusMsgError - bus message handling finished with error

Код: SVDB7007, Http: 500

ошибка обработки сообщения хэндлером шины

Является базовой ошибкой, должна использоваться через конструктор:

с дополнительным сообщение pattern + args

`bus.NewHandleBusMsgError(err error, pattern string, args ...any)`

SVDB7008 bus.GetStreamInfoError - can’t get info about/from stream

Код: SVDB7008, Http: 500

ошибка при получении информации о JetStream

Является базовой ошибкой, должна использоваться через конструктор:

с дополнительным сообщение pattern + args

`bus.NewGetStreamInfoError(err error, pattern string, args ...any)`

SVDB7009 systables.NotClusterModeJSFetch - can’t get nats channel in SINGLENODE mode

Код: SVDB7009, Http: 400

ошибка формируется, при попытке получить LOGS или ERRORS в SINGLENODE режиме

Является SINGLTON ошибкой, используется прямо как переменная

SVDB7010 server.InvalidSessionUIDError - invalid session uid

Код: SVDB7010, Http: 500

ошибки gRPC, связанные с sessionuid

Является базовой ошибкой, должна использоваться через конструктор:

используя дополнительные параметры:

1. `sessionUID` - sessionUID для которого сформирована ошибка (может быть пустым)

2. `pattern` + `args` - стандартное дополнительное сообщение

```
server.NewInvalidSessionUIDError(sessionUID string, pattern string, args ...any)
```

SVDB7011 server.RestCallProcessingError - rest request processing error

Код: SVDB7011, Http: 500

ошибки, связанные с обработкой rest-вызовов

Является базовой ошибкой, должна использоваться через конструктор:

оборачивает внешний [err] с дополнительным сообщение pattern + args

```
server.NewRestCallProcessingError(err error, pattern string, args ...any)
```

SVDB8000 driver.GeneralDriverError - general native client error

Код: SVDB8000, Http: 500

некая ошибка в работе драйвера, не специфичная - обертка над различными другими ошибками

Является базовой ошибкой, должна использоваться через конструктор:

оборачивает внешний [err] с дополнительным сообщение pattern + args

```
driver.NewGeneralDriverError(err error, pattern string, args ...any)
```

SVDB8001 driver.NoSessionUIDInServerHeadersError - no sessionuid in server headers

Код: SVDB8001, Http: 500

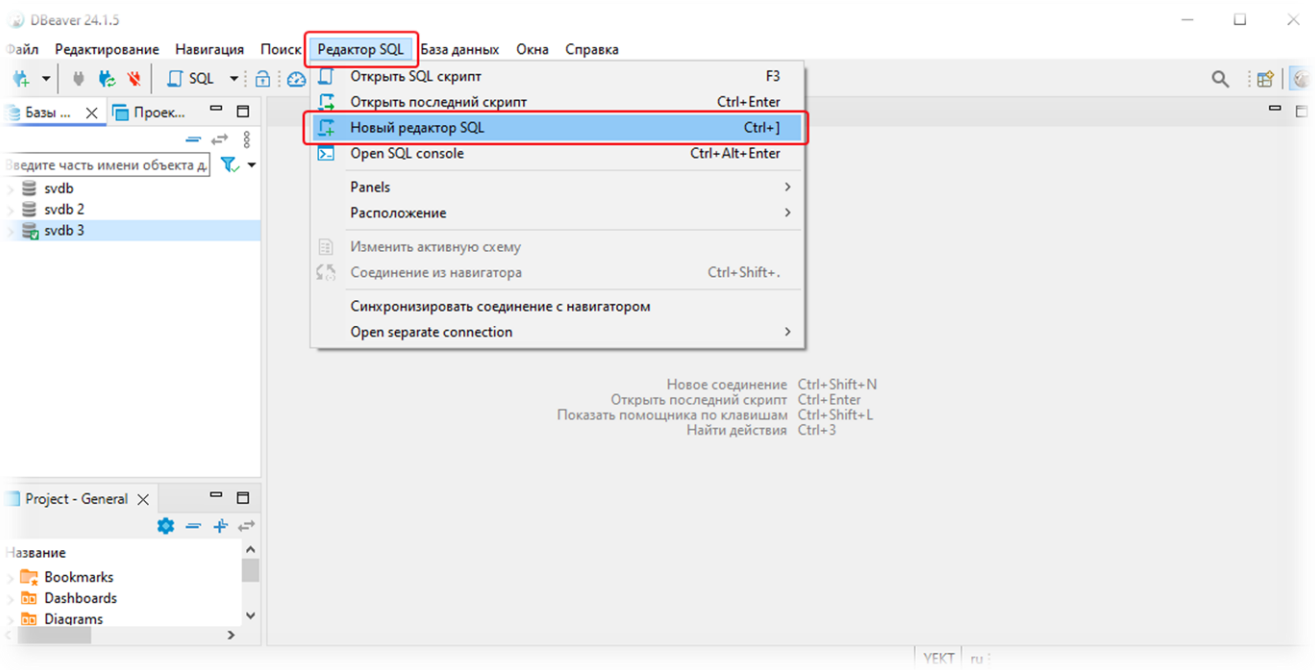
сбой протокола, отсутствие идентификатора сессии в ответе сервера

Является SINGLTON ошибкой, используется прямо как переменная

Приложение 1

Пример: выполнить SQL запрос для получения справочника регионов.

1. Выбрать в верхнем меню пункт «Редактор SQL» -> «Новый редактор SQL»



2. Написать запрос и нажать кнопку «Выполнить SQL запрос»

---

*select * from dict.regions*