ДОКУМЕНТАЦИЯ В РАЗРАБОТКЕ

Версия 20250305-1913

Обзор языка запросов Ecomet

Этот язык предназначен для работы с операциями, такими как **извлечение данных, модификация, управление подписками и удаление**, обеспечивая структурированный и выразительный синтаксис. В следующих разделах документированы доступные типы запросов, их синтаксис и примеры.

---

1. Запросы `GET`

Назначение

Получение данных из одной или нескольких баз данных (или объектов) с использованием **фильтрации, группировки, сортировки, постраничной навигации и агрегации**.

Общий синтаксис

get <список полей> from <список баз данных> where <условия> [опции]

- **Список полей:** Одно или несколько полей для извлечения. Можно использовать вызовы функций (`$concat`, `$term`) и псевдонимы (`AS`).
- **Список баз данных:** Названия баз данных или идентификаторы объектов (например, `root`, `db1`).
- **Условия:** Логическое выражение с операторами `=`, `:=`, `:>`, `:<`, `:LIKE` и логическими связками `AND`, `OR`, `ANDNOT`.
- **Опции (необязательно):** Дополнительные параметры, такие как группировка (`group by`), сортировка (`order by`), постраничная загрузка (`page`), блокировка (`lock`), форматирование (`format`).

Примеры

Простой GET-запрос:

Получить поле `foo` (с псевдонимом `'alias'`) из базы `root`, где `buz` больше 123:

```
get foo AS 'alias' from root where AND ( buz :> 123 )
```

-

Запрос с несколькими базами и полями:
Получить `foo1` и `foo2` из `db1`, `db2`, где `buz` содержит `f` (используется `LIKE`) или `bar` равно -42:

```
get foo1, foo2 from db1,db2 where OR ( buz :LIKE 'f', bar = -42 )
```

-

Запрос с отрицанием условий и сортировкой:
Получить `foo` из базы `'TEST'`, исключая объекты, у которых `buz = 45.5` и `bar < 4`, отсортировав по убыванию `foo3`:

```
get foo from 'TEST' where ANDNOT (buz := 45.5, bar :< 4) order by foo3 DESC
```

-

Постраничная загрузка:
Получить `foo` из `root`, где `buz = 1`, начиная с 5-й записи и ограничив выборку 10 записями:

```
get foo from root where AND (buz := 1) page 5:10
```

-

Блокировка чтения (`lock read`):
Получить `foo` из всех баз (`*`), применяя блокировку на чтение:

```
get foo from * where AND (bar :< 4) page 5:10 lock read
```

-

Группировка и сортировка:
Группировка по `foo2` и `buz2`, сортировка по `foo3` и `buz3` (по убыванию):

```
get foo1 from d1,d2,'D3' where AND (bar :< 4) group by foo2, buz2 order by foo3, buz3 DESC
```

-

Кастомное форматирование:
Использование встроенного JSON-форматтера:

```
get foo1 from root where AND (bar :< 4) format $json
```

Или вызов пользовательской функции форматирования:

get foo1 from root where AND (bar :< 4) format $my_lib:my_fun

-

---

2. Запросы `SUBSCRIBE` и `UNSUBSCRIBE`

Назначение

Подписка на обновления данных и управление подписками.

Синтаксис

Подписка:
subscribe <запрос> [опции]

-

Отмена подписки:
unsubscribe <идентификатор подписки>

-

Пример:

subscribe get foo from root where AND (buz :> 123)

Отмена подписки:

unsubscribe <subscription-id>

(`<subscription-id>` заменяется на идентификатор подписки.)

---

3. Запросы `SET`

Назначение

Обновление значений полей существующих объектов.

Синтаксис

set <назначения полей> in <список баз данных> where <условия> [опции]

Примеры:

Обновление поля:

```
set foo=$hello in root where AND (bar :< 1) lock write
```

-

Обновление с функцией:

```
set foo=$concat($buz,'_test') in * where OR (bar :> 1) lock read
```

-

---

4. Запросы `INSERT`

Назначение

Добавление новых объектов в базу данных.

Синтаксис

insert <назначения полей> [format <форматтер>]

Примеры:

Простое добавление:

```
insert foo1=123, foo2=$term('buz')
```

-

Добавление с форматированием:

```
insert foo1=123, foo2=$term('buz') format $from_string
```

-

---

5. Запросы DELETE

Назначение

Удаление объектов из базы данных.

Синтаксис

delete from <список баз данных> where <условия> [опции]

Примеры:

Удаление с блокировкой:

```
delete from root where AND (bar :< 1) lock write
```

-

Удаление по сложному условию:

```
delete from db1,db2 where ANDNOT (bar :LIKE '1', buz := foo)
```

-

---

6. Управление транзакциями

Назначение

Операции над транзакциями: начало, фиксация, откат.

Синтаксис

Начало транзакции:

```
transaction_start
```

-

Фиксация изменений:

```
transaction_commit
```

-

Откат изменений:

```
transaction_rollback
```

-

Пример транзакции:

transaction_start
set foo=$hello in root where AND (bar :< 1) lock write
transaction_commit

(Откат транзакции при ошибке:)

transaction_start
... (какие-то запросы) ...
transaction_rollback

Итоги

Язык запросов **Ecomet Query Language** поддерживает:

- **Извлечение данных** (`get`) с фильтрацией, сортировкой и группировкой.
- **Изменение данных** (`set`).
- **Добавление данных** (`insert`).
- **Удаление данных** (`delete`).
- **Подписки на изменения** (`subscribe`/`unsubscribe`).
- **Управление транзакциями**.

Каждый запрос может включать **опции блокировки, форматирования, группировки и постраничной загрузки**, что делает его мощным инструментом для работы с распределёнными данными.

Используйте приведённые примеры для создания своих запросов! 🚀

Ecomet Query Language (EQL) - $count Function Usage

Функция $count является агрегатной функцией в Ecomet Query Language (EQL), которая используется для подсчета количества элементов (объектов, записей и т.д.), соответствующих определенному запросу или критериям. Она обычно используется в сочетании с group by для предоставления количества элементов в группах.

1. Basic Usage:
   - Самое простое использование $count - это получение общего количества элементов, соответствующих запросу. Если вы не используете group by, $count возвращает одно число, представляющее все подходящие элементы.
2. Usage with group by:
   - Когда $count используется с group by, она возвращает количество для каждой отдельной группы. Результаты включают идентификатор каждой группы и соответствующее ей количество.
3. Usage with AS
   - Вы можете дать имя результату функции, используя as. Это полезно, когда вы используете результаты в программе.
4. Multiple Aggregate Functions:
   - Вы можете использовать несколько агрегатных функций в одном запросе.
5. Restrictions:
   - $count может использоваться только в get и subscribe.
   - $count должен использоваться как часть списка полей get.
   - $count поддерживает только (). Вы не можете предоставлять никаких аргументов.

Summary:

- $count() предоставляет мощный способ подсчета объектов в Ecomet.
- В сочетании с group by она может генерировать агрегатные счетчики, что делает ее ценным инструментом для аналитики и обобщения данных в Ecomet.

Supported environments:

- get
- subscribe

Not Supported:

- set
- delete
- insert

Ecomet Query Language (EQL) - group by Clause Usage
Предложение group by в Ecomet Query Language (EQL) используется для группировки строк, которые имеют одинаковые значения в одном или нескольких столбцах, в сводные строки, например, "найти количество пользователей в каждой стране". Предложение group by часто используется с агрегатными функциями (такими как $count, $sum, $avg, $max, $min) для вычисления сводной информации для каждой группы.

1. Basic Usage:
   - Базовый синтаксис для использования group by - это указание одного или нескольких полей (столбцов), по которым вы хотите сгруппировать данные.
   - Когда вы используете group by, результаты запроса будут иметь одну строку для каждой уникальной комбинации значений в указанных полях группировки.
2. Usage with Aggregate Functions:
   - group by наиболее эффективен в сочетании с агрегатными функциями.
   - Агрегатные функции применяются к каждой группе отдельно.
3. Grouping by Multiple Fields:
   - Вы можете группировать по нескольким полям. Результаты будут сгруппированы по каждой уникальной комбинации значений в этих полях.
4. group by and where:
   - Вы можете использовать group by с предложением where для фильтрации данных перед их группировкой.
5. group by and order by
   - You can use group by with order by to order result
6. Restrictions:
   - Поля, которые не указаны в предложении group by, должны использоваться внутри агрегатной функции. В противном случае результат будет неопределенным.
   - group by поддерживается только в get и subscribe.

Summary:
- group by - это фундаментальный инструмент для агрегации данных в EQL.
- Он позволяет вам обобщать данные на основе общих характеристик.
- Сочетание с фильтрами where и агрегатными функциями делает его очень мощным.
- Вы можете группировать по одному или нескольким полям для создания многоуровневых сводок данных.

Supported environments:
- get
- subscribe

Not Supported:
- set
- delete
- insert

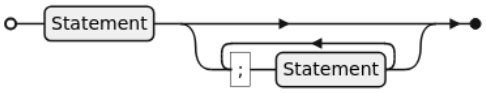
```
@startebnf
' Основное правило – список операторов
StatementList = Statement, { ";", Statement } ;

' Оператор может быть одним из нескольких вариантов
```

```
Statement = Get
          | Subscribe
          | Unsubscribe
          | Set
          | Insert
          | Delete
          | transaction_start
          | transaction_commit
          | transaction_rollback ;

' Правило для оператора GET
Get = "get", GetFieldList, "from", Databases, "where", Condition, [ GetParamList ] ;

' Правило для оператора SUBSCRIBE
Subscribe = "subscribe", text, "get", GetFieldList, "from", Databases, "where", Condition, [
SubParamList ] ;

' Правило для оператора UNSUBSCRIBE
Unsubscribe = "unsubscribe", text ;

' Правило для оператора SET
Set = "set", SetFieldList, "in", Databases, "where", Condition, [ SetParamList ] ;

' Правило для оператора INSERT
Insert = "insert", SetFieldList, [ InsertParamList ] ;

' Правило для оператора DELETE
Delete = "delete", "from", Databases, "where", Condition, [ Lock ] ;

' Правило для описания баз данных
Databases = "*"
          | Database, { ",", Database } ;

' Правило для отдельной базы данных
Database = Atom | text ;

' Правило для списка полей в операторе GET
GetFieldList = "*"
          | GetField, { ",", GetField } ;

' Правило для описания поля выборки
GetField = FieldValue, [ "AS", text ]
          | Function, "(", ")", [ "AS", text ]
          | Function, "(", VariableList, ")", [ "AS", text ] ;

' Правило для значений полей (возможно, цепочка)
FieldValue = Field, [ ">", FieldValue ] ;
```

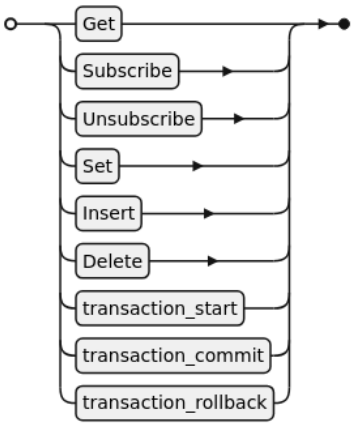
```
' Правило для списка устанавливаемых полей (оператор SET)
SetFieldList = SetField, { ",", SetField } ;

' Правило для описания установки поля
SetField = Field, "=", Variable ;

' Правило для идентификатора (атом)
Atom = "field" | "atom" ;

' Правило для условий
Condition = Field, Operator, Constant
          | Field, "[", Date, ":", Date, "]"
          | "AND", "(", ConditionList, ")"
          | "OR", "(", ConditionList, ")"
          | "ANDNOT", "(", Condition, ",", Condition, ")" ;

' Список условий (для AND/OR)
ConditionList = Condition, { ",", Condition } ;

' Операторы сравнения и логики
Operator = "=" | ":=" | ":>" | ":<" | ":>=" | ":=<" | ":<>" | "LIKE" | ":LIKE" ;

' Правило для даты
Date = integer | text ;

' Дополнительные параметры для GET
GetParamList = GetParam, { GetParam } ;
GetParam = "order by", OrderByList
          | "group by", GroupByList
          | "page", integer, ":", integer
          | Lock
          | Format ;

' Дополнительные параметры для SUBSCRIBE
SubParamList = SubParam, { SubParam } ;
SubParam = "stateless" | "no_feedback" | Format ;

' Дополнительные параметры для SET
SetParamList = SetParam, { SetParam } ;
SetParam = Lock | Format ;

' Дополнительные параметры для INSERT
InsertParamList = InsertParam, { InsertParam } ;
InsertParam = "update" | Lock | Format ;

' Правило для блокировки
Lock = "lock read" | "lock write" ;
```

```
' Параметры сортировки
OrderByList = OrderBy, { ",", OrderBy } ;
OrderBy = Field, OrderDirection
        | integer, OrderDirection
        | Field
        | integer ;
OrderDirection = "ASC" | "DESC" ;

' Параметры группировки
GroupByList = GroupBy, { ",", GroupBy } ;
GroupBy = Field | integer ;

' Форматирование результата
Format = "format", "$", Atom, [ ":", Atom ] ;

' Правило для списка констант
ConstantList = Constant, { ",", Constant } ;

' Константа может быть простым терминалом или выражением
Constant = ConstTerm
        | "[", ConstantList, "]"
        | Function, "(", ConstantList, ")" ;

' Терминальные константы
ConstTerm = integer | float | text | Atom ;

' Список переменных
VariableList = Variable, { ",", Variable } ;

' Правило для переменной
Variable = ConstTerm
        | "$", FieldValue
        | "[", VariableList, "]"
        | Function, "(", ")"
        | Function, "(", VariableList, ")" ;

' Правило для вызова функции
Function = "$", Atom, [ ":", Atom ] ;
@endebnf
```

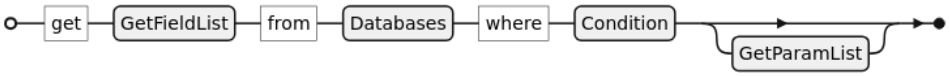
StatementList



Statement

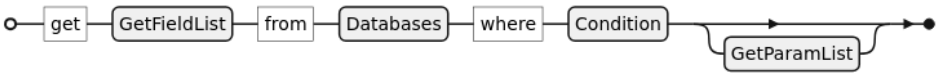


Get



Subscribe

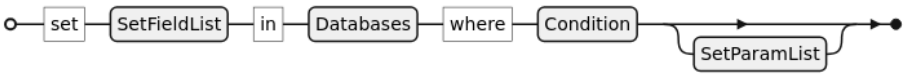
Get



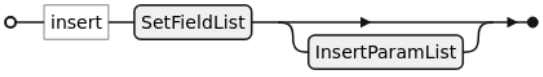
Subscribe

Unsubscribe

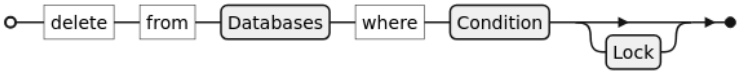
Set



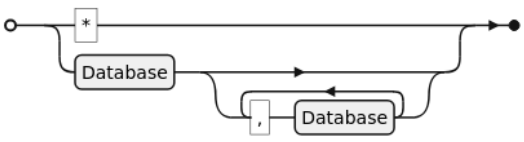
Insert



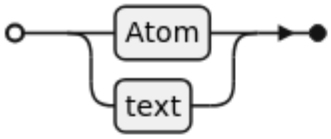
Delete



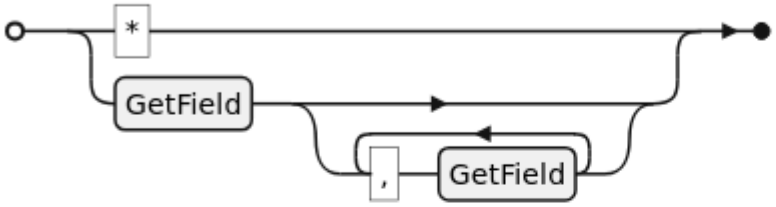
Databases



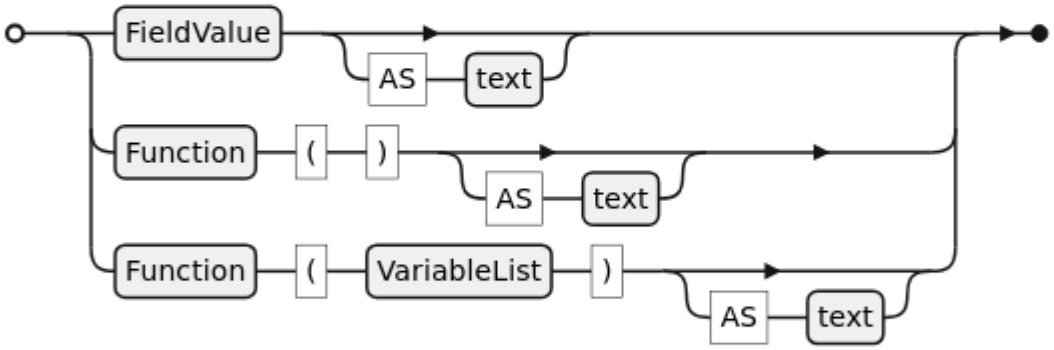
Database



GetFieldList

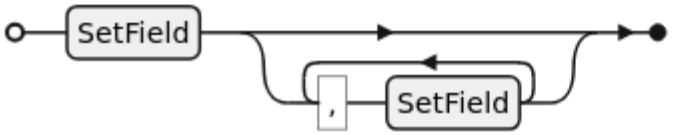


GetField

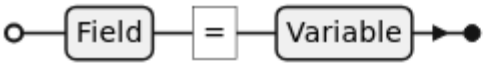


FieldValue

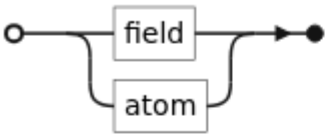
SetFieldList



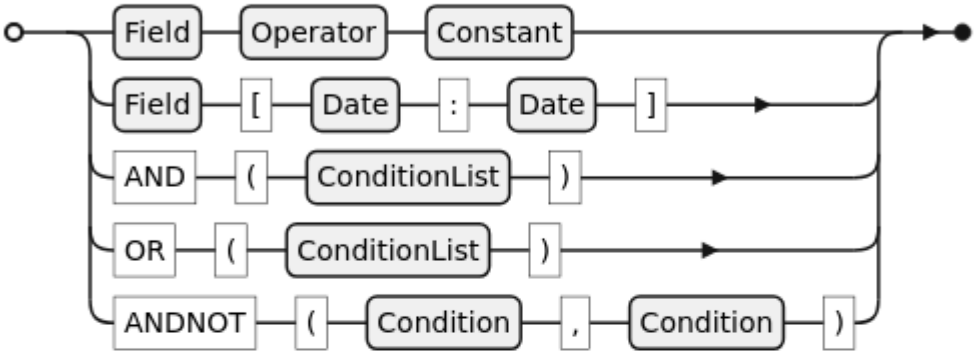
SetField



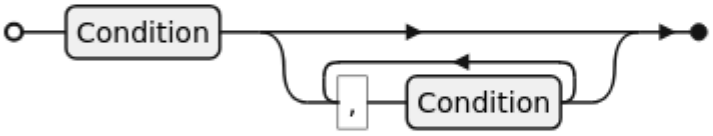
Atom



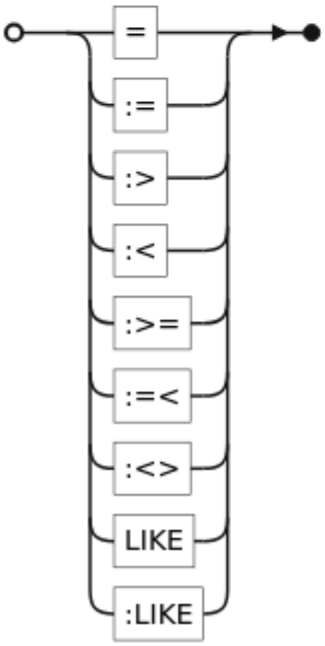
Condition



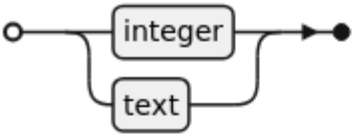
ConditionList



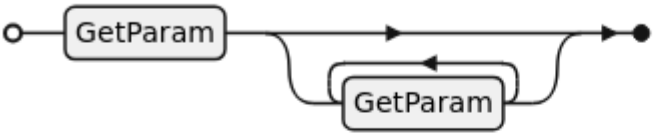
Operator



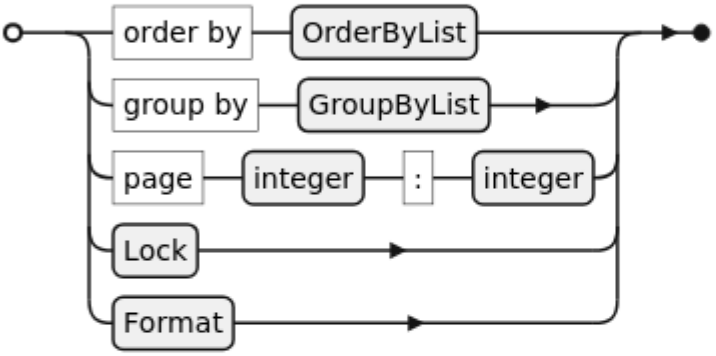
Date



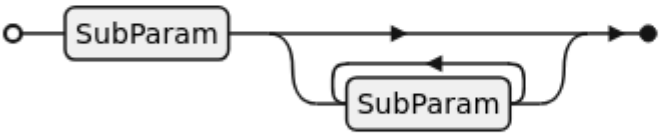
GetParamList



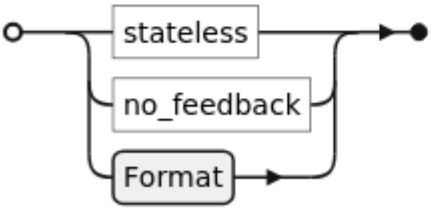
GetParam



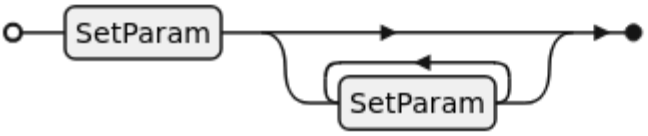
SubParamList



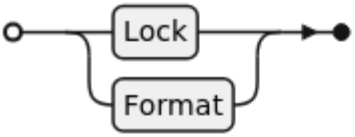
SubParam



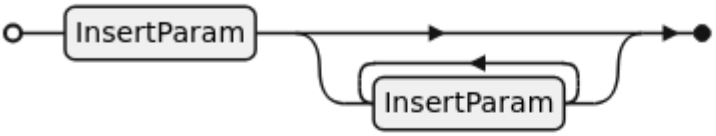
SetParamList



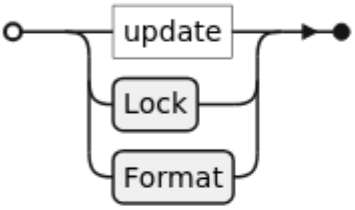
SetParam



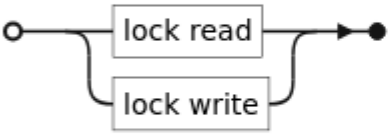
InsertParamList



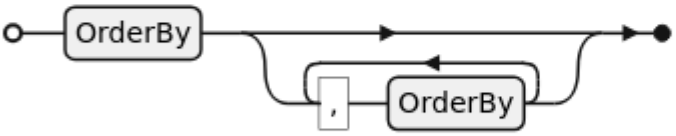
InsertParam



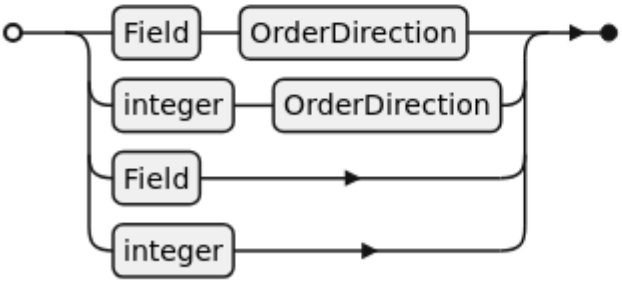
Lock



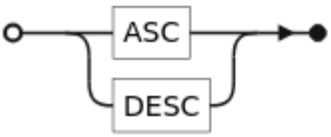
OrderByList



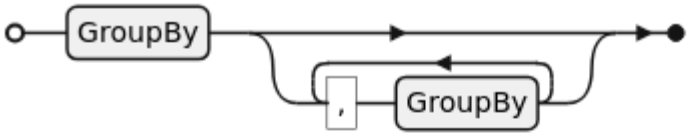
OrderBy



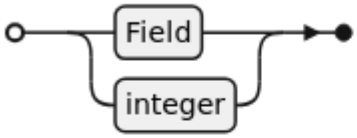
OrderDirection



GroupByList

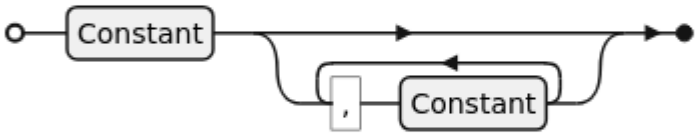


GroupBy

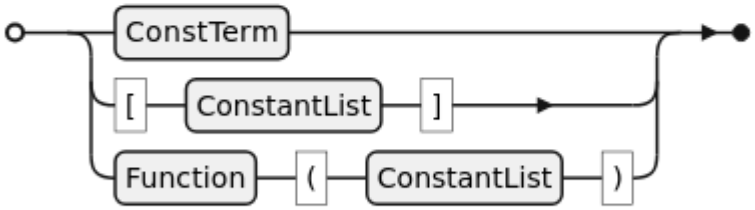


Format

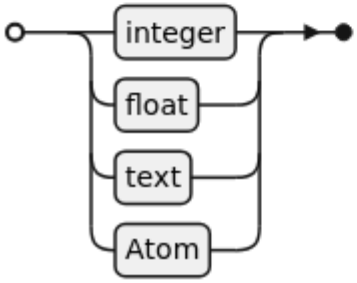
ConstantList



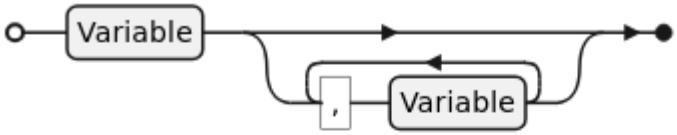
Constant



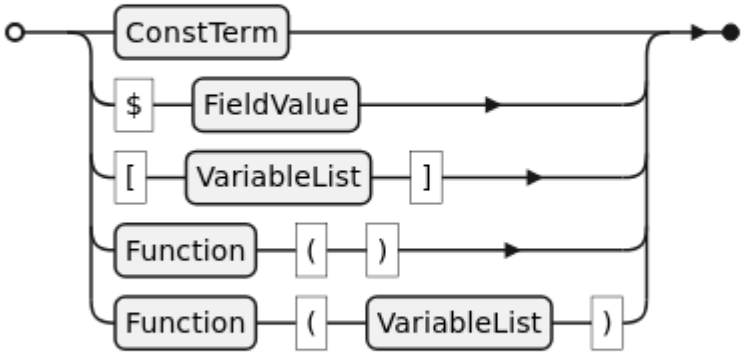
ConstTerm



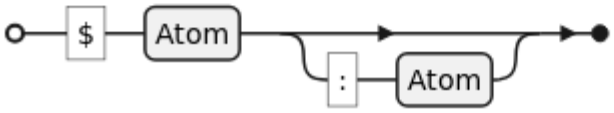
VariableList



Variable



Function



Оператор LIKE в Ecomet Query Language

Назначение:
Оператор **LIKE** используется для сопоставления шаблонов со строковыми полями (конкретно, с полями типа binary() в Erlang). Его основная задача — выбор объектов, у которых значение поля удовлетворяет заданному шаблону.

Поддерживаемые метасимволы:

- **^**
    - **Назначение:** Явно указывает, что сопоставление должно начинаться с указанного шаблона.

Пример:
{<<".fp_path">>, 'LIKE', <<"^", BasePath/binary, "/">>}

    - Это условие выбирает объекты, у которых значение поля `".fp_path"` начинается с `BasePath/` (например, если `BasePath` равен `"/root/FP/PROJECT/my_folder"`, будут выбраны пути, начинающиеся с `"/root/FP/PROJECT/my_folder/"`).
- **_**
    - **Назначение:** Соответствует любому одному символу (аналог оператора `.` в регулярных выражениях).
    - **Примечание:** В предоставленных примерах этот символ не используется, но его можно предположить для обозначения одного символа.
- **%**
    - **Назначение:** Соответствует любой последовательности из нуля или более символов (аналог `.*` в регулярных выражениях).
    - **Примечание:** В представленных примерах не используется, однако его поведение можно предположить как стандартное для подобных шаблонов.

Ограничения:

- Поддерживаются только вышеуказанные метасимволы. Другие символы, характерные для полноценного синтаксиса регулярных выражений (например, `*` или `+`), не поддерживаются.
- Символ **$** не используется, а его правило отсутствует как в файлах реализации, так и в лексере.

Синтаксис использования LIKE:
Формат условия с оператором LIKE выглядит следующим образом:

{ Field, 'LIKE', Pattern }

где:

- **Field** — строковое поле, которое подлежит сопоставлению.
- **Pattern** — строковый шаблон, включающий поддерживаемые метасимволы.

Примеры:

Якорь начала строки:

{<<".fp_path">>, 'LIKE', <<"^/root/FP/PROJECT/">>}

1. Выбирает все объекты, у которых значение поля `".fp_path"` начинается с `"/root/FP/PROJECT/"`.

Использование подстановочного символа для одного символа:

{<<".fp_path">>, 'LIKE', <<"/root/_/">>}

2. Будут выбраны строки, где после `"/root/"` идет ровно один символ (например, `"/root/a/"` или `"/root/b/"`), но не `"/root/ab/"`.

Использование подстановочного символа для произвольной последовательности:

{<<".fp_path">>, 'LIKE', <<"/root/%/">>}

3. Выбирает строки, начинающиеся с `"/root/"` и затем содержащие произвольную последовательность символов (например, `"/root/a/"`, `"/root/ab/"`, `"/root/abc/"` и т.д.).

Применение в коде:
Рассмотрим фрагмент из `fp_replica_ts_send.erl`, где LIKE используется для поиска архивов:

```
ArchiveIDs = fp_db:get(?PROJECT_DBs, [<<".oid">>], {'AND', [
  {<<".pattern">>, '=', ?OID(<<"/root/.patterns/ARCHIVE">>)},
  {<<".fp_path">>, 'LIKE', <<"^", BasePath/binary, "/">>}
]}),
```

- **<<".fp_path">>** — поле, содержащее путь.
- **'LIKE'** — оператор сопоставления.
- **<<"^", BasePath/binary, "/">>** — шаблон, где `^` обеспечивает сопоставление только тех строк, которые начинаются с `BasePath/`.

Также оператор LIKE используется в подписке (subscription):

```
fp_db:subscribe(archives, ?PROJECT_DBs, [<<".oid">>],
  {'AND', [
    {<<".pattern">>, '=', ?OID(<<"/root/.patterns/ARCHIVE">>)},
    {<<".fp_path">>, 'LIKE', <<"^", BasePath/binary, "/">>}
  ]}, #{
    stateless => true
  }),
```

Поведение оператора идентично: он выбирает объекты, у которых значение поля `".fp_path"` начинается с заданного базового пути.

Заключение:
Оператор LIKE в Ecomet Query Language обеспечивает ограниченное сопоставление шаблонов:

- **^** гарантирует, что сопоставление начинается с указанного шаблона.
- **_** и **%** могут использоваться для подстановки одного или произвольного количества символов соответственно, хотя в приведённых примерах они не продемонстрированы.
- Другие элементы регулярных выражений не поддерживаются.

Примеры из проекта BPM журналы

Текущая смена

```
GET status, .name, .folder

FROM * WHERE AND(

    .pattern=$oid('/root/.patterns/shift'),      .path :like
'/root/JOURNALS/OPERATIONAL/technical/',

    OR(         status='active'
```

```
    ))

PAGE 1:50

format $to_json
```

```
GET status, .name

FROM * WHERE AND(

    .pattern=$oid('/root/.patterns/shift'),      .path :like
'/root/JOURNALS/OPERATIONAL/technical/YUG_RDC/',

    OR(         status='active'

    ))

PAGE 1:50

format $to_json
```

Несколько ролей

```
GET .name, .oid, .pattern, .ts

FROM *

WHERE AND(

    .folder=$oid('/root/.users'),

    OR(

        usergroups=$oid('/root/.usergroups/oper_chief_dispatcher'),

        usergroups=$oid('/root/.usergroups/oper_dispatcher'),
```

```
 usergroups=$oid('/root/.usergroups/oper_chief'),
        usergroups=$oid('/root/.usergroups/oper_engineer')
    )
)
ORDER  BY .ts DESC
PAGE 1:200
format $to_json
```

ограничения

```
GET .name, .oid, .pattern, .ts
FROM *
WHERE AND(
    .folder=$oid('/root/.users'),
    OR(
        usergroups=$oid('/root/.usergroups/restrict_commenter')
    )
)
ORDER  BY .ts DESC
PAGE 1:200
format $to_json
```

Сообщения

```
GET .name, .oid, .pattern, .ts
FROM *
WHERE AND(
    .folder=$oid('/root/.users'),
```

```
    OR(
        usergroups=$oid('/root/.usergroups/oper_message_author')
    )
)
ORDER  BY .ts DESC
PAGE 1:200
format $to_json
```

Диапазон

```
GET
.name,
.folder
FROM *
WHERE AND(
    .pattern=$oid('/root/.patterns/shift'),
     AND(
.name :>= '159',
.name :=< '180')
)
ORDER BY .name DESC
PAGE 1:100
format $to_json
```

Точное соответствие

```
GET .oid, .path, .folder, .pattern, .name

FROM *

WHERE AND(

    .pattern=$oid('/root/.patterns/shift'),

    .name='3122'

)

format $to_json


LIKE

GET .folder, message, .oid, created_at

FROM *

WHERE AND(

    .pattern=$oid('/root/.patterns/record'),

    message :like 'Пригородная-Восточн'

)

ORDER  BY created_at DESC

format $to_json




GET .oid, .path, .folder, .pattern, .name

FROM *

WHERE AND(
```

```
    .pattern=$oid('/root/.patterns/shift'),
    .name :like '3122'
)
format $to_json
```

```
LIKE
GET .name
FROM *
WHERE AND(
.folder=$oid('/root/.users'),
usergroups=$oid('/root/.usergroups/oper_dispatcher')
)
format $to_json
```

```
GET .name, .oid, .pattern
FROM *
WHERE AND(
    .folder=$oid('/root/.users'),
    OR(
        usergroups=$oid('/root/.usergroups/oper_chief_dispatcher'),
        usergroups=$oid('/root/.usergroups/oper_dispatcher'),
        usergroups=$oid('/root/.usergroups/oper_engineer')
    )
)
format $to_json
```

я

1. Выборка данных

```
GET <список_полей>
FROM *
WHERE AND(
    <условия_фильтрации>
)
```

```
ORDER BY <поле> <ASC|DESC>  # Опционально
PAGE <номер_страницы>:<размер_страницы>  # Опционально
format $to_json
```

2. Папки, статусы и условия

Фильтрация по конкретной папке (.path :like)

Поиск по status='active'

Использование OR() для нескольких условий

```
GET <список_полей>
FROM *
WHERE AND(
    .pattern=$oid('<путь_к_шаблону>'),
    .path :like '<папка>',
    OR(
        status='active'
    )
)
PAGE 1:50
format $to_json
```

3. Группы пользователей (usergroups=$oid)

```
GET .name, .oid, .pattern, .ts
FROM *
WHERE AND(
    .folder=$oid('/root/.users'),
    OR(
        usergroups=$oid('/root/.usergroups/oper_chief_dispatcher'),
        usergroups=$oid('/root/.usergroups/oper_dispatcher'),
        usergroups=$oid('/root/.usergroups/oper_engineer')
    )
)
ORDER BY .ts DESC
PAGE 1:200
format $to_json
```

4. Диапазон значений (.name :>=, .name :=<)

```
GET .name, .folder
FROM *
WHERE AND(
    .pattern=$oid('/root/.patterns/shift'),
    AND(
        .name :>= '176',
        .name :=< '180'
    )
```

```
)
ORDER BY .name DESC
PAGE 1:100
format $to_json
```

5. Точное значение (.name='...')

```
GET .oid, .path, .folder, .pattern, .name
FROM *
WHERE AND(
    .pattern=$oid('/root/.patterns/shift'),
    .name='25777537024'
)
format $to_json
```

6. Частичное соответствие (.name :like)

```
GET .name, .oid, .path, .folder, .pattern
FROM *
WHERE AND(
    .pattern=$oid('/root/.patterns/shift'),
    .name :like '25777'
)
```

format $to_json

Итог

GET <поля> — указываешь нужные поля.

FROM * — означает, что ищем во всех данных.

WHERE AND(...) — основное условие поиска.

ORDER BY <поле> — сортировка, если нужна.

PAGE 1:100 — пагинация (номер страницы:размер).

format $to_json — формат вывода JSON.

я

```
GET .name, .oid, .pattern
FROM *
WHERE AND(
    .folder=$oid('/root/.users'),
    usergroups :in [
        $oid('/root/.usergroups/oper_chief_dispatcher'),
        $oid('/root/.usergroups/oper_dispatcher'),
        $oid('/root/.usergroups/oper_engineer')
    ]
)
```

format $to_json