

Запросы к **графу событий**: синтаксис, рецепты и почему здесь не нужны промежуточные таблицы

Четыре страницы: как устроены данные, из чего состоит запрос, двенадцать готовых рецептов для решений и короткое сравнение с SQL.

1. Как устроены данные: всё — события

В графе нет таблиц «задачи» и «статусы». Есть одно множество **событий**, у каждого шесть полей. Запрос смотрит на эти поля и ни на что больше.

Поле	Что в нём	Пример
<code>:\$id</code>	идентификатор события (хэш содержимого)	<code>#7976f100...</code>
<code>:\$base</code>	к кому относится: метка задачи, подзадачи, эпика, команды	<code>"BET-1037"</code>
<code>:\$type</code>	что это: свойство (<code>status</code> , <code>team</code> , <code>assignee</code> , <code>worklog</code> ...) или служебный тип (<code>Individual</code>)	<code>"status"</code>
<code>:\$value</code>	значение: новый статус, имя команды, часы (строкой), метка индивида	<code>"Тестирование"</code>
<code>:\$date</code>	момент события в миллисекундах; в запросе можно сравнивать со строкой-датой	<code>1787...000</code>
<code>:\$cause</code>	причины: id событий, на которых стоит это (у смены статуса — предыдущий статус). Многочисленное	<code>["#658f17d2"]</code>
<code>:\$actor</code>	кто записал (у списаний — исполнитель)	<code>"Лебедев И."</code>

Сущность = событие создания. Задача появляется событием `Individual` с `:$base =` концепт и `:$value =` код:
`[?i :$type "Individual"] [?i :$base "Task"] [?i :$value ?task]`. Концепты: `Task` , `Subtask` , `Epic` , `TeamEvent` .

Свойство = событие на индивиде. Команда задачи: `[?p :$base ?task] [?p :$type "team"] [?p :$value ?team]` . Вся история хранится; «текущее» значение отдельно не помечено, его выражают отрицанием (рецепт 1).

2. Из чего состоит запрос

```
[ :find ?task ?status      ; что вернуть — переменные (начинаются с ?)
:where [?e :$base ?task]   ; клюз-паттерн [событие :поле значение]
      [?e :$type "status"] ; константы в кавычках
      [?e :$value ?status]
      [(>= ?d "01.06.2026")] ; предикат — в круглых скобках внутри квадратных
      (not [?n :$cause ?id]) ; отрицание — «не существует»
      (ancestor ?id ?a)     ; вызов правила
:rules [[(имя ?a ?b) клюзы...]] ; свои правила, необязательно
```

Элемент	Как работает
Паттерн [?e :\$attr ?v]	Находит все события, у которых поле совпадает. Одна и та же переменная в двух паттернах — это соединение (join): [?e :\$base ?t] [?p :\$base ?t] связывает события одной задачи.
Порядок клюзов	Исполняется слева направо: первым ставьте самый узкий паттерн (с константой), следующие начинайте с уже связанной переменной. Так запрос в 10 раз быстрее.
Предикаты	= not= != > < >= <= над числами. Дата в графе — число, но справа можно писать строку: [(< ?d "01.08.2026")]. Переменные должны быть уже связаны.
Функции	Результат ставится после формы: [(- ?d2 ?d1) ?ms], затем [(days ?ms) ?dn]. Есть + - * / days hours num . num превращает строку «8,5» в число.
Отрицание (not ...)	Ветка отбрасывается, если тело not находит хоть одно решение при текущих связях. Хотя бы одна переменная внутри должна быть связана раньше. Внутри :rules нельзя.
Правила	Именованный набор клюзов с параметрами; могут вызывать сами себя — рекурсия. Встроены (ancestor ?id ?a) и (descendant ?a ?d) : все причины события вглубь и все следствия.
Результат	Множество строк: одинаковые схлопываются. Чтобы видеть каждую задачу, добавьте в :find её код или id события. Лимиты: 5 секунд, 5000 строк. Ctrl+Enter — выполнить.

Три идиомы, из которых собирается почти всё. Текущее значение — событие, у которого нет следующего: (not [?n :\$cause ?id]). Переход $A \rightarrow B$ — событие B, у которого в причинах событие A: [?e :\$cause ?c] [?p :\$id ?c] [?p :\$value "A"]. Время в статусе — дата следующего минус дата этого: [(- ?d2 ?d1) ?ms] [(days ?ms) ?dn].

3. Рецепты для решений

Статусы — как в вашем раск; в примерах эталонный поток. Каждый рецепт есть в приложении в списке «примеры запросов», группа «Для решений», с готовой диаграммой.

1. Где сейчас стоят открытые задачи

в каком статусе копится незавершенное

```
[ :find ?status ?task
:where [?i :$type "Individual" ] [?i :$base
"Task" ] [?i :$value ?task]
[?e :$base ?task] [?e :$type "status" ] [?
e :$value ?status] [?e :$id ?id]
(not [?n :$cause ?id] [?n :$type
"status" ])
(not [?d :$base ?task] [?d :$type
"status" ] [?d :$value "Закрыта" ])]
```

Диаграмма: бары по ?status. Высокий столбик очереди — узкое место ниже по потоку.

2. Открытые задачи по исполнителям

кто тянет больше всех одновременно

```
[ :find ?who ?task
:where [?i :$type "Individual" ] [?i :$base
"Task" ] [?i :$value ?task]
(not [?d :$base ?task] [?d :$type
"status" ] [?d :$value "Закрыта" ])
[?a :$base ?task] [?a :$type "assignee" ]
[?a :$value ?who]]
```

Бары по ?who. Больше 3–4 на человека — переключения съедают время.

3. Возвраты из тестирования

чья задача возвращает с проверки

```
[ :find ?who ?e
:where [?e :$type "status" ] [?e :$value
"Разработка" ] [?e :$cause ?c] [?e :$date ?d]
[?p :$id ?c] [?p :$type "status" ] [?p
:$value "Тестирование" ]
[(>= ?d "01.06.2026" )]
[?e :$base ?task] [?a :$base ?task] [?a
:$type "assignee" ] [?a :$value ?who]]
```

Один шаг назад по причинной связи. ?e в :find, чтобы считать события, а не людей.

4. Закрывают, минуя тестирование

что ушло в прод без проверки

```
[ :find ?team ?task
:where [?e :$type "status" ] [?e :$value
"Закрыта" ] [?e :$base ?task]
[?i :$type "Individual" ] [?i :$base
"Task" ] [?i :$value ?task]
(not [?x :$base ?task] [?x :$type
"status" ] [?x :$value "Тестирование" ])
[?tm :$base ?task] [?tm :$type "team" ] [?
tm :$value ?team]]
```

Бары по ?team. Единицы — норма, много у одной команды — риск качества.

5. Ждали в статусе дольше N дней

время в статусе через функции

```
[ :find ?task ?who ?days
:where [?e :$type "status" ] [?e :$value "Готова
к тестированию" ]
[?e :$base ?task] [?e :$date ?d1] [?e
:$id ?id]
[?n :$cause ?id] [?n :$type "status" ] [?n
:$date ?d2]
[(- ?d2 ?d1) ?ms] [(days ?ms) ?days] [(>
?days 5) ]
[?a :$base ?task] [?a :$type "assignee" ]
[?a :$value ?who]]
```

Бары: x = ?task, y = сумма ?days. Застрявшие сейчас — рецепт 6.

6. Застряли в текущем статусе с даты

кто стоит на месте слишком долго

```
[ :find ?task ?status ?since ?who
:where [?i :$type "Individual" ] [?i :$base
"Task" ] [?i :$value ?task]
[?e :$base ?task] [?e :$type "status" ] [?
e :$value ?status] [?e :$id ?id]
(not [?n :$cause ?id] [?n :$type
"status" ])
(not [?d :$base ?task] [?d :$type
"status" ] [?d :$value "Закрыта" ])
[?e :$date ?since] [(< ?since
"01.10.2026" )]
[?a :$base ?task] [?a :$type "assignee" ]
[?a :$value ?who]]
```

Дату поставьте «две недели назад». Разобрать каждую на Дейли.

7. Закрыто за период по командам

сколько реально довели до конца

```
[ :find ?team ?task
  :where [?e :$type "status"] [?e :$value
    "Закрыта"] [?e :$base ?task] [?e :$date ?d]
    [(>= ?d "01.09.2026")] [(< ?d
    "01.10.2026")]
    [?tm :$base ?task] [?tm :$type "team"] [?
    tm :$value ?team]]
```

Для динамики: верните ?d вместо ?team, диаграмма «линия», корзина «неделя».

8. В работе без единого списания

где работа идёт «фантомно»

```
[ :find ?task ?who
  :where [?e :$type "status"] [?e :$value
    "Разработка"] [?e :$base ?task]
    [?i :$type "Individual"] [?i :$base
    "Task"] [?i :$value ?task]
    (not [?pp :$type "parent"] [?pp :$value ?
    task] [?pp :$base ?sub]
    [?w :$type "worklog"] [?w :$base ?
    sub])
    [?a :$base ?task] [?a :$type "assignee"]
    [?a :$value ?who]]
```

Списания лежат на подзадачах; связь parent ведёт от подзадачи к задаче.

9. Часы по исполнителям

кто сколько списал за период

```
[ :find ?who ?h ?w
  :where [?w :$type "worklog"] [?w :$factor ?who]
    [?w :$value ?hs] [(num ?hs) ?h] [?w
    :$date ?d]
    [(>= ?d "01.09.2026")]]
```

Бары: x = ?who, y = ?h, агрегат «сумма». ?w в :find, чтобы строки не схлопывались.

10. История одной задачи

где она лежала

```
[ :find ?status ?date ?who
  :where [?e :$base "BET-1037"] [?e :$type
    "status"]
    [?e :$value ?status] [?e :$date ?date] [?
    e :$factor ?who]]
```

Первый инструмент разбора «почему шла месяц». Сортировка — по колонке даты.

11. Провенанс: из чего выведено закрытие

вся цепочка причин вглубь

```
[ :find ?a
  :where [?e :$base "BET-1037"] [?e :$type
    "status"]
    [?e :$value "Закрыта"] [?e :$id ?id]
    (ancestor ?id ?a)]
```

Встроенное правило: причины, причины причин и так далее до корня.

12. Что происходило после события журнала

окно 14 дней после даты

```
[ :find ?task ?s ?d
  :where [?e :$type "status"] [?e :$base ?task] [?
    e :$value ?s] [?e :$date ?d]
    [(>= ?d "26.08.2026")] [(< ?d
    "09.09.2026")]
    [?p :$base ?task] [?p :$type "team"] [?p
    :$value "Бета"]]
```

Дату события возьмите из журнала команды. Двух окон в одном запросе нет (нет or) — делайте два запроса.

Своё правило: возврат как отношение

```
[ :find ?task
  :where (returned ?task "Тестирование" "Разработка")
  :rules [[(returned ?t ?from ?to)
    [?e :$base ?t] [?e :$type "status"] [?e :$value ?to] [?e :$cause ?c]
    [?p :$id ?c] [?p :$type "status"] [?p :$value ?from]]]]
```

Правило — это именованный кусок запроса. Его можно вызывать сколько угодно раз, с разными параметрами, из других правил и рекурсивно. Это и есть ответ на вопрос следующего раздела.

4. Почему в SQL нужны промежуточные таблицы, а здесь нет

В SQL запрос описывает **таблицу-результат**: откуда взять строки, как соединить, как сгруппировать. Когда вопрос в два шага («текущий статус», а потом «кто в нём стоит дольше N дней»), первый шаг приходится **материализовать**: временная таблица, CTE, представление или подзапрос с оконной функцией, и только потом второй шаг читает его результат. Каждый новый вопрос — новая промежуточная таблица, потому что «текущий статус» в реляционной модели не существует как вещь: его надо вычислить и где-то положить.

В Datalog запрос описывает **отношение**: набор условий на переменные. Любое условие можно назвать правилом и подставлять в другие условия, как функцию. Движок сам строит план и вычисляет всё до неподвижной точки, ничего не материализуя по дороге. «Текущий статус» становится не таблицей, а строкой `(not [?n :$cause ?id])`, которую можно вписать в любой запрос.

SQL: «задачи, вернувшиеся из тестирования в разработку»

```
WITH s AS (  
  SELECT task, status, at,  
         LAG(status) OVER (PARTITION BY task  
                           ORDER BY at) AS prev  
  FROM status_events)  
SELECT DISTINCT task FROM s  
WHERE prev = 'Тестирование'  
AND status = 'Разработка';
```

Порядок восстанавливается по времени (LAG), а время у событий из разных источников может совпасть или быть исправлено задним числом. Промежуточная таблица `s` нужна, потому что «предыдущий статус» не хранится.

Datalog: то же самое

```
[ :find ?task  
  :where [?e :$type "status"] [?e :$value  
    "Разработка"  
    [?e :$base ?task] [?e :$cause ?c]  
    [?p :$id ?c] [?p :$type "status"]  
    [?p :$value "Тестирование"]]
```

«Предыдущий статус» хранится явно — в причине события. Запрос читает связь, а не восстанавливает её по часам. Ни таблицы, ни окна, ни сортировки.

Задача	SQL	Datalog здесь
Текущее значение свойства	оконная функция <code>ROW_NUMBER() OVER (... ORDER BY at DESC)</code> и фильтр <code>= 1</code> , обычно в CTE	<code>(not [?n :\$cause ?id])</code> : событие без следующего
Переход A → B	self-join по задаче с условием на время или LAG	<code>[?e :\$cause ?c] [?p :\$id ?c]</code> : причина хранится
Цепочка причин любой длины	рекурсивный CTE <code>WITH RECURSIVE</code> , свой на каждый случай	<code>(ancestor ?id ?a)</code> : правило вызывается где угодно
Переиспользование шага	представление или временная таблица; меняется схема, нужны права	правило в <code>:rules</code> прямо в запросе
«Нет ни одного события»	<code>NOT EXISTS (SELECT ...)</code> с коррелированным подзапросом	<code>(not ...)</code> с теми же переменными
Состояние на прошлую дату	фильтр по времени во всех CTE сразу, легко забыть одно место	те же клозы плюс предикат по <code>:\$date</code>

Честно о цене. У SQL есть агрегаты, оконные функции, индексы и оптимизатор на терабайты; здесь — пять секунд и 5000 строк в браузере, агрегация вынесена в диаграмму (сумма, среднее, медиана, p85 по колонке). Datalog выигрывает там, где вопрос причинный и многошаговый, а данные — события с явными связями. Для «сколько строк в таблице за месяц» SQL ничуть не хуже.

5. Если не работает

- **«несвязанная переменная в предикате»** — предикат или функция стоят раньше паттерна, который даёт значение. Переставьте.
- **«(not ...) должен ссылаться на связанную переменную»** — сначала найдите сущности, потом исключайте.
- **Пусто, хотя должно быть** — регистр и пробелы в значениях важны: "Тестирование" ≠ "тестирование"; имена свойств как на вкладке «Схема».
- **Меньше строк, чем ожидали** — одинаковые строки схлопнулись: добавьте в `:find` код задачи или `?e`.
- **Долго** — первым клозом поставьте самую узкую константу (`:$value "Закрыта"`), а не `[?e :$type "status"]` по всему графу.
- **Не хочется писать руками** — «Вопрос к базе» и AI-аналитик составляют запрос по вопросу своими словами, «Поиск» собирает его из фразы; кнопка «открыть в редакторе» показывает текст, который можно править.

Vanilitik · AREAL Flow Analytics · vanilitik2.areals.ru (свои данные) · vanilitik5.areals.ru (вопросов) · workshop.vanilitik.areals.ru · примеры «Для решений» встроены во вкладку Datalog