

튜토리얼: 시간을 아는 그래프 — 유효 시간과 시스템 시간 (bitemporal)

이 문서가 답하는 질문

- "2023년 6월에 ann의 연봉은 얼마였나"와 "그때 우리는 그것을 얼마로 알고 있었나"를 어떻게 따로 묻는가?
- 세상이 바뀐 것(변경)과 우리가 잘못 알았던 것(정정)은 어떻게 구분해서 기록하고, 읽을 때 어떻게 달라지는가?
- 시점이 바뀔 때 사이퍼와 컴파일된 SQL은 무엇이 바뀌는가? (답: 아무것도 바뀌지 않는다)
- 아직 시작되지 않은 변경, 그리고 시점을 답할 수 없는 질의는 어떻게 처리되는가?

처음이라면 데모 영상부터 보세요. Studio에서 같은 질의를 시점만 바꿔 다섯 번 묻는 과정을 실제로 실행한 화면이고, 설명은 자막으로 들어 있습니다. 이 문서는 PDF 와 Word 문서 로도 받을 수 있습니다.

작은 회사 acme의 인사 기록입니다. 직원은 셋, 기록은 여섯 번입니다.

기록	세상에서 언제부터 (valid time)	내용	종류
W1	2020-01-01	ann(엔지니어, 연봉 100), bob(디자이너, 80) 입사	생성
W2	2022-07-01	cho(분석가, 90) 입사	생성
W3	2023-01-01	ann 연봉 인상, 200	변경 — 세상이 바뀌었다
W4	(그대로)	인상액을 잘못 적었다. 200이 아니라 250	정정 — 우리가 틀렸다
W5	2024-06-01	ann 퇴사	삭제
W6	2030-01-01	bob 연봉 인상 예정, 120	미래 날짜 변경

여섯 기록 모두 이 문서를 빌드하는 순간에 입력됩니다. 그래서 축이 둘 필요합니다.

축	묻는 것	저장되는 곳	읽을 때 지정	쓸 때 지정
유효 시간 (valid time)	세상에서 언제 사 실이었나	행마다 <code>__valid</code> <code>tstzrange</code>	<code>og.valid_at</code>	<code>og.valid_from</code> , <code>og.valid_to</code>
시스템 시간 (system time)	우리가 언제부터 그렇게 믿었나	현재 행 <code>__sys_from</code> , 이력 행 <code>[__sys_from, __sys_to)</code>	<code>og.system_at</code>	지정할 수 없다. 항상 실제 시계

시스템 시간은 실제 시계라서 실행할 때마다 다릅니다. 이 문서는 각 기록의 트랜잭션 시각을 `marks` 테이블에 `W1`~`W6` 로 적어 두고, 출력에서는 시각 대신 그 라벨을 보여 줍니다. 레퍼런스는 사용자 매뉴얼, 설계 배경은 ADR-027, 구현 상태(일부 미구현)는 spec 013 에 있습니다.

준비

이 문서의 출력은 손으로 적은 것이 아닙니다. 아래 명령이 빈 데이터베이스 `tut_temporal` 을 만들고 원본 `tutorial.src.md` 의 예제를 순서대로 실행해 결과를 끼워 넣습니다. 예제가 하나라도 기대와 다르면 빌드가 실패합니다.

```
python3 examples/tutorials/build_tutorial.py temporal
```

SHELL

직접 따라 하려면 `psql` 세션에 SQL 블록을 붙여 넣으세요. 사이퍼 블록은 `SELECT og_cypher('corp', $$ ... $$)` 로 감싸면 됩니다. **시점은 질의가 아니라 세션 설정입니다** (`SET og.valid_at = '...'`, `SET og.system_at = '...'`, 끝나면 `RESET`). SQL 블록 첫 줄의 `\set q '...'` 는 사이퍼 문장을 `psql` 변수에 담는 것이고, 한 블록 안에서 **같은 q** 를 시점만 바꿔 실행합니다.

0단계. 스키마를 만들고 시간 버전을 켜다

시간 버전은 그래프 전체가 아니라 **타입마다** 켵니다. 켜 타입만 이력을 남깁니다.

```
SELECT og_create_graph('corp');
SELECT og_create_type('corp', 'Person', 'entity');
SELECT og_add_property('corp', 'Person', 'name', 'string'), og_add_property('corp', 'Person',
'salary', 'int');
SELECT og_create_type('corp', 'Company', 'entity'), og_add_property('corp', 'Company', 'name',
'string');
SELECT og_create_type('corp', 'WORKS_AT', 'relation'), og_add_property('corp', 'WORKS_AT', 'title',
'string');
CREATE TABLE marks (label text PRIMARY KEY, at timestamptz); -- 각 기록의 시스템 시각
SELECT og_enable_temporal('corp', 'Person');
SELECT og_enable_temporal('corp', 'Company');
SELECT og_enable_temporal('corp', 'WORKS_AT');
```

SQL

```
SELECT og_temporal_types('corp');
```

SQL

```
og_temporal_types
```

결과

```
-----
{Company,Person,WORKS_AT}
(1 row)
```

관계 타입 `WORKS_AT` 도 켵습니다. 시점 질의에 등장하는 타입은 모두 켵야 합니다. 켜지 않은 타입이 어떻게 답하는지는 8 단계에서 봅니다. 켜 것을 끄는 함수는 없습니다.

1단계. 다섯 번 기록한다 (W1~W5)

쓰기는 평소의 사이퍼입니다. 달라지는 것은 트랜잭션 안에서 `SET LOCAL` 로 **언제부터**(`og.valid_from`), **누가**(`og.actor`), **어디서**(`og.source`) 를 말한다는 점뿐입니다. `SET LOCAL` 은 커밋과 함께 사라지므로 다음 기록으로 새 지 않습니다. `og.valid_from` 을 주지 않으면 `now()` 입니다.

SQL

```
BEGIN;                                -- W1: 2020년 입사
SET LOCAL og.valid_from = '2020-01-01';
SET LOCAL og.actor = 'kim@hr'; SET LOCAL og.source = 'hr-portal';
SELECT og_cypher('corp', $$
    CREATE (c:Company {name: 'acme'})
    CREATE (:Person {name: 'ann', salary: 100})-[:WORKS_AT {title: 'engineer'}]->(c)
    CREATE (:Person {name: 'bob', salary: 80})-[:WORKS_AT {title: 'designer'}]->(c)$$);
INSERT INTO marks VALUES ('W1', now());
COMMIT;

BEGIN;                                -- W2: 2022년 7월 입사
SET LOCAL og.valid_from = '2022-07-01'; SET LOCAL og.actor = 'kim@hr';
SELECT og_cypher('corp', $$
    MATCH (c:Company {name: 'acme'}) CREATE (:Person {name: 'cho', salary: 90})-[:WORKS_AT {title:
'analyst'}]->(c)$$);
INSERT INTO marks VALUES ('W2', now());
COMMIT;
```

W3은 **변경**입니다. 2023년 1월 1일부터 세상이 달라졌습니다. W4는 **정정**입니다. 세상은 그대로이고 우리가 적은 숫자가 틀렸습니다. `'-infinity'` 가 그 뜻이고, 유효 구간(2023-01-01부터)은 건드리지 않고 값만 바꿉니다. W5는 삭제인데, 관계는 2024-06-01까지 존재한 것으로 남습니다.

SQL

```
BEGIN;                                -- W3: 2023년부 인상 (변경)
SET LOCAL og.valid_from = '2023-01-01'; SET LOCAL og.actor = 'kim@hr';
SELECT og_cypher('corp', $$MATCH (p:Person {name: 'ann'}) SET p.salary = 200$$);
INSERT INTO marks VALUES ('W3', now());
COMMIT;

BEGIN;                                -- W4: 200이 아니라 250이었다 (정정)
SET LOCAL og.valid_from = '-infinity'; SET LOCAL og.actor = 'lee@payroll';
SELECT og_cypher('corp', $$MATCH (p:Person {name: 'ann'}) SET p.salary = 250$$);
INSERT INTO marks VALUES ('W4', now());
COMMIT;

BEGIN;                                -- W5: 2024년 6월 퇴사
SET LOCAL og.valid_from = '2024-06-01'; SET LOCAL og.actor = 'kim@hr';
SELECT og_cypher('corp', $$MATCH (:Person {name: 'ann'})-[:WORKS_AT]->() DELETE w$$);
INSERT INTO marks VALUES ('W5', now());
COMMIT;
```

2단계. 지금을 묻는다

이 문서 내내 쓸 질문입니다. "누가 어떤 직함으로 일하고 얼마를 받는가." 시점을 정하지 않으면 현재입니다.

```
MATCH (p:Person)-[w:WORKS_AT]->(c:Company)
RETURN p.name AS name, w.title AS title, p.salary AS salary ORDER BY name
```

결과

```
name | title | salary
-----+-----+-----
bob  | designer | 80
cho  | analyst  | 90
(2 rows)
```

두 행입니다. ann은 2024년에 퇴사해서 없습니다.

3단계. 유효 시간 — "그때 실제로 어땠나"

같은 사이퍼를 `og.valid_at` 만 바꿔 네 번 묻습니다. 결과 표의 머리글이 시점입니다.

SQL

```
\set q 'MATCH (p:Person)-[w:WORKS_AT]->(c:Company) RETURN p.name AS name, w.title AS title, p.salary
AS salary ORDER BY name'
SET og.valid_at = '2019-06-01';
SELECT og_cypher('corp', :q) AS "valid_at = 2019-06-01";
SET og.valid_at = '2021-06-01';
SELECT og_cypher('corp', :q) AS "valid_at = 2021-06-01";
SET og.valid_at = '2023-06-01';
SELECT og_cypher('corp', :q) AS "valid_at = 2023-06-01";
SET og.valid_at = '2024-07-01';
SELECT og_cypher('corp', :q) AS "valid_at = 2024-07-01";
```

결과

```
valid_at = 2019-06-01
-----
(0 rows)

valid_at = 2021-06-01
-----
{"name": "ann", "title": "engineer", "salary": 100}
{"name": "bob", "title": "designer", "salary": 80}
(2 rows)

valid_at = 2023-06-01
-----
{"name": "ann", "title": "engineer", "salary": 250}
{"name": "bob", "title": "designer", "salary": 80}
{"name": "cho", "title": "analyst", "salary": 90}
(3 rows)

valid_at = 2024-07-01
-----
{"name": "bob", "title": "designer", "salary": 80}
{"name": "cho", "title": "analyst", "salary": 90}
(2 rows)
```

- 2019-06: 아무도 입사하기 전입니다. 0행.
- 2021-06: ann 100, bob 80. cho는 2022년 7월에 입사하므로 없습니다.
- 2023-06: ann은 **250** 입니다. 200은 정정되었으므로, 지금 아는 바로는 사실이었던 적이 없습니다.
- 2024-07: ann의 **WORKS_AT** 은 2024-06-01에 끝났습니다. 삭제한 관계가 그 전 시점에서는 보입니다.

검산: 행 수는 0 → 2 → 3 → 2, 급여 합계는 0 → 180 → 420 → 170 입니다.

4단계. 시스템 시간 — "그때 우리는 무엇을 믿었나"

유효 시간을 2023-06-01에 **고정**하고 언제의 믿음으로 볼지를 바꿉니다. 시각은 **marks** 에서 **\gset** 으로 꺼냅니다.

SQL

```
\set q 'MATCH (p:Person)-[w:WORKS_AT]->(c:Company) RETURN p.name AS name, w.title AS title, p.salary
AS salary ORDER BY name'
SET og.valid_at = '2023-06-01';
SELECT at AS w1 FROM marks WHERE label = 'W1' \gset
SELECT at AS w3 FROM marks WHERE label = 'W3' \gset
SELECT at AS w4 FROM marks WHERE label = 'W4' \gset
SET og.system_at = :w1';
SELECT og_cypher('corp', :q) AS "2023-06-01, W1 직후에 알던 대로";
SET og.system_at = :w3';
SELECT og_cypher('corp', :q) AS "2023-06-01, W3 직후에 알던 대로";
SET og.system_at = :w4';
SELECT og_cypher('corp', :q) AS "2023-06-01, W4 직후에 알던 대로";
```

결과

2023-06-01, W1 직후에 알던 대로

```
-----
{"name": "ann", "title": "engineer", "salary": 100}
{"name": "bob", "title": "designer", "salary": 80}
(2 rows)
```

2023-06-01, W3 직후에 알던 대로

```
-----
{"name": "ann", "title": "engineer", "salary": 200}
{"name": "bob", "title": "designer", "salary": 80}
{"name": "cho", "title": "analyst", "salary": 90}
(3 rows)
```

2023-06-01, W4 직후에 알던 대로

```
-----
{"name": "ann", "title": "engineer", "salary": 250}
{"name": "bob", "title": "designer", "salary": 80}
{"name": "cho", "title": "analyst", "salary": 90}
(3 rows)
```

"2023년 6월의 급여 대장"이라는 같은 질문의 답이 **언제 물었느냐**에 따라 다릅니다.

- W1 직후: cho의 입사도 ann의 인상도 아직 기록되지 않았습니다. ann 100, bob 80.
- W3 직후: 인상은 알았지만 액수를 틀리게 알았습니다. ann **200**. 당시 보고서에 적혔을 값입니다.
- W4 직후: 정정 뒤입니다. ann **250**.

이것이 변경과 정정의 차이입니다. ann의 연봉만 놓고 2022년 6월과 2023년 6월을, W3 직후와 지금의 믿음으로 물어 봅니다.

SQL

```
\set q 'MATCH (p:Person {name: "ann"}) RETURN p.salary AS salary'
SELECT at AS w3 FROM marks WHERE label = 'W3' \gset
SET og.valid_at = '2022-06-01';
SELECT og_cypher('corp', :q) AS "2022-06, 지금 아는 대로";
SET og.valid_at = '2023-06-01';
SELECT og_cypher('corp', :q) AS "2023-06, 지금 아는 대로";
SET og.system_at = :w3;
SELECT og_cypher('corp', :q) AS "2023-06, W3 직후에 알던 대로";
```

결과

2022-06, 지금 아는 대로

```
-----
{"salary": 100}
(1 row)
```

2023-06, 지금 아는 대로

```
-----
{"salary": 250}
(1 row)
```

2023-06, W3 직후에 알던 대로

```
-----
{"salary": 200}
(1 row)
```

유효 시간으로 물으면

옛 값이 남는 곳

변경 (W3, `valid_from = '2023-01-01'`)

2023년 전은 100, 이후는 새 값

유효 시간 축. `og.valid_at` 을 2022년으로 두면 보인다

정정 (W4, `valid_from = '-infinity'`)

어느 날짜를 물어도 200은 나오지 않는다

시스템 시간 축에만. `og.system_at` 을 정정 전으로 두어야 보인다

유효 시간 하나만 있었다면 "작년 감사 때 보고한 2023년 연봉(200)"은 정정과 함께 사라졌을 것입니다.

5단계. 같은 사이퍼, 같은 SQL

시점은 생성된 타입 뷰가 읽는 세션 설정(`og_valid_at()`, `og_system_at()`)이라서, 사이퍼가 컴파일된 SQL에는 날짜가 들어가지 않습니다. 시점 없이, 그리고 두 시점을 모두 설정하고 컴파일한 SQL의 지문(md5)을 비교합니다.

SQL

```
\set q 'MATCH (p:Person)-[w:WORKS_AT]->(c:Company) RETURN p.name AS name, w.title AS title, p.salary
AS salary ORDER BY name'
SELECT md5(og_cypher_sql('corp', :q)) AS now_sql \gset
SET og.valid_at = '2021-06-01';
SET og.system_at = '2026-01-01';
SELECT md5(og_cypher_sql('corp', :q)) = :now_sql AS same_sql,
       og_cypher_sql('corp', :q) LIKE '%og_valid_at()%' AS reads_setting,
       og_cypher_sql('corp', :q) LIKE '%2021%' AS contains_date;
```

결과

```
same_sql | reads_setting | contains_date
-----+-----+-----
t        | t        | f
(1 row)
```

same_sql = t, contains_date = f. 시점마다 다른 문장을 만들지 않으므로 플랜 캐시가 시점 수만큼 늘지 않습니다. Studio는 결과 프레임마다 이 지문을 `compiled SQL ...` 로 보여 줍니다.

6단계. 아직 시작되지 않은 변경 (W6)

bob의 인상은 2030년 1월 1일부터입니다. 오늘 기록하지만 오늘의 답을 바뀌어서는 안 됩니다.

SQL

```
BEGIN;
-- W6: 2030년부 인상 예정
SET LOCAL og.valid_from = '2030-01-01'; SET LOCAL og.actor = 'kim@hr';
SELECT og_cypher('corp', $$MATCH (p:Person {name: 'bob'}) SET p.salary = 120$$);
INSERT INTO marks VALUES ('W6', now());
COMMIT;
```

```
\set q 'MATCH (p:Person {name: "bob"}) RETURN p.salary AS salary'
SELECT og_cypher('corp', :q) AS "지금";
SET og.valid_at = '2029-12-31';
SELECT og_cypher('corp', :q) AS "valid_at = 2029-12-31";
SET og.valid_at = '2030-01-01';
SELECT og_cypher('corp', :q) AS "valid_at = 2030-01-01";
```

지금

```
-----
{"salary": 80}
(1 row)
```

valid_at = 2029-12-31

```
-----
{"salary": 80}
(1 row)
```

valid_at = 2030-01-01

```
-----
{"salary": 120}
(1 row)
```

지금과 2029년 말은 80, 2030년 1월 1일부터 120입니다. 유효 구간은 [시작, 끝) 이라 시작 시각은 포함됩니다.

주의 — 이 보장은 라벨이 있는 패턴에만 해당합니다. 라벨 없는 노드 패턴은 시간 필터가 없는 경로로 저장 테이블의 현재 행 ("2030년부터 120")을 읽어서, 시점을 정하지 않았는데도 아직 시작되지 않은 값을 냅니다 (실측, 엔진 결함으로 보고). 미래 날짜 변경을 쓰는 타입은 항상 라벨을 붙여 질의하세요. 같은 노드를 두 패턴으로 지금 묻습니다.

```
MATCH (p:Person {name: 'bob'}), (n {name: 'bob'}) RETURN p.salary AS labelled, n.salary AS unlabelled
```

```
labelled | unlabelled
-----+-----
80       | 120
(1 row)
```

7단계. 이력 테이블에는 무엇이 남았나

저장 테이블(og_data.n_<id>)은 지금 믿는 행을, 그 옆의 이력 테이블(og_data.h_<id>)은 대체된 믿음과 잘려 나간 과거 구간을 갖습니다. Person 의 두 테이블을 봅니다.

```

SELECT storage_table AS cur, 'og_data.h_' || type_id AS hist FROM og_catalog.type WHERE name =
'Person' \gset
SELECT p_name AS name, p_salary AS salary, lower(__valid)::date AS valid_from, upper(__valid)::text
AS valid_to,
      m.label AS sys_from, __actor AS actor
FROM :cur c JOIN marks m ON m.at = c.__sys_from ORDER BY name;
SELECT p_name AS name, p_salary AS salary, lower(__valid)::date AS valid_from,
upper(__valid)::date::text AS valid_to,
      m.label AS sys_from, coalesce(n.label, '∞') AS sys_to, __op AS op
FROM :hist h JOIN marks m ON m.at = h.__sys_from LEFT JOIN marks n ON n.at = h.__sys_to
ORDER BY h.__sys_from, h.__sys_to;

```

결과

name	salary	valid_from	valid_to	sys_from	actor
ann	250	2023-01-01	infinity	W4	lee@payroll
bob	120	2030-01-01	infinity	W6	kim@hr
cho	90	2022-07-01	infinity	W2	kim@hr

(3 rows)

name	salary	valid_from	valid_to	sys_from	sys_to	op
ann	100	2020-01-01	infinity	W1	W3	u
bob	80	2020-01-01	infinity	W1	W6	u
ann	200	2023-01-01	infinity	W3	W4	u
ann	100	2020-01-01	2023-01-01	W3	∞	v
bob	80	2020-01-01	2030-01-01	W6	∞	v

(5 rows)

첫 표가 현재 행 3개, 둘째 표가 이력 5행입니다. 이력을 한 줄씩 읽으면:

1. **u** ann 100, [2020, ∞), W1~W3 동안의 믿음. "2020년부터 계속 100"이라던 것을 W3이 대체했습니다.
2. **u** bob 80, [2020, ∞), W1~W6 동안의 믿음. W6이 대체했습니다.
3. **u** ann 200, [2023, ∞), W3~W4 동안의 믿음. 틀린 인상액입니다. W4가 정정했습니다.
4. **v** ann 100, [2020, 2023), W3부터 **지금까지** 믿음 (sys_to = ∞). 변경이 잘라 낸 앞부분입니다.
5. **v** bob 80, [2020, 2030), W6부터 지금까지 믿음. bob의 **오늘 연봉은 이 행에** 있습니다.

정정(W4)은 **v** 조각을 만들지 않았습니다. '-infinity' 는 구간을 자르지 않기 때문입니다. sys_to = ∞ 인 **v** 행은 과거 기록이 아니라 현재의 일부이고, 시점을 정하지 않은 질의도 이 행을 읽습니다 (6단계의 bob 80이 그 예입니다).

8단계. 답할 수 없으면 거부한다

시점이 설정된 상태에서 현재만 볼 수 있는 경로를 타는 질의는, 오늘 데이터로 그럴듯한 답을 내는 대신

OG_TEMPORAL_UNSUPPORTED (SQLSTATE 0A000) 로 거부됩니다. 네 가지입니다.

```
SET og.valid_at = '2021-06-01';
SELECT og_cypher('corp', 'MATCH (p:Person)-[:WORKS_AT*1..2]->(c:Company) RETURN c.name');
SELECT og_cypher('corp', 'MATCH (n) RETURN count(n) AS n');
SELECT og_cypher('corp', 'MATCH (p:Person)-[r]->(c:Company) RETURN type(r)');
SELECT og_typeql('corp', 'match $p isa Person;');
```

```
ERROR: OG_TEMPORAL_UNSUPPORTED: variable-length traversal over a temporal relationship type cannot
answer a point-in-time query (og.valid_at / og.system_at is set)
HINT: Use typed patterns and single-hop relationships over temporal types, or RESET og.valid_at and
og.system_at.
ERROR: OG_TEMPORAL_UNSUPPORTED: an unlabelled node pattern cannot answer a point-in-time query
(og.valid_at / og.system_at is set)
HINT: Use typed patterns and single-hop relationships over temporal types, or RESET og.valid_at and
og.system_at.
ERROR: OG_TEMPORAL_UNSUPPORTED: a relationship variable without a single type cannot answer a
point-in-time query (og.valid_at / og.system_at is set)
HINT: Use typed patterns and single-hop relationships over temporal types, or RESET og.valid_at and
og.system_at.
ERROR: OG_TEMPORAL_UNSUPPORTED: TypeQL cannot answer a point-in-time query (og.valid_at /
og.system_at is set)
HINT: Use typed patterns and single-hop relationships over temporal types, or RESET og.valid_at and
og.system_at.
```

가변 길이 경로, 라벨 없는 노드 스캔, 타입이 하나로 정해지지 않은 관계 변수, TypeQL입니다. 3단계처럼 라벨을 붙이고 한
홉씩 쓰면 같은 시점에서 답이 나오고, **RESET** 하면 거부된 질의도 그대로 실행됩니다.

거부되지 않지만 알아 두어야 할 것이 셋 있습니다. 모두 실제로 실행해 확인한 동작입니다.

① 시간 버전을 켜지 않은 타입은 어느 시점을 물어도 현재를 답합니다. **Skill** 은 켜지 않았고 오늘 만들었습니다.

```
SELECT og_cypher('corp', $$CREATE (:Skill {name: 'sql'})$$);
SET og.valid_at = '2019-06-01';
SELECT og_cypher('corp', 'MATCH (s:Skill) RETURN s.name AS skill') AS "valid_at = 2019-06-01";
```

```
og_cypher
-----
(0 rows)

valid_at = 2019-06-01
-----
{"skill": "sql"}
(1 row)
```

② 시점을 설정한 채로 쓰면 과거가 아니라 현재 행이 바뀝니다. **MATCH** 는 2021년의 ann(100)을 찾지만 **SET** 은 같은 id
의 현재 행을 고칩니다. 과거 유효 구간을 직접 고치는 기능은 없습니다.

SQL

```
BEGIN;
SET LOCAL og.valid_at = '2021-06-01';
SELECT og_cypher('corp', 'MATCH (p:Person {name: "ann"}) RETURN p.salary AS seen') AS "MATCH 가 본 값";
SELECT og_cypher('corp', 'MATCH (p:Person {name: "ann"}) SET p.salary = 111');
SELECT p_salary AS salary, lower(__valid)::date = current_date AS valid_from_is_today
  FROM og_data.n_1 WHERE p_name = 'ann';
ROLLBACK;
```

결과

MATCH 가 본 값

{"seen": 100}

(1 row)

og_cypher

(0 rows)

salary | valid_from_is_today

111 | t

(1 row)

③ 이력 보존(retention) 정책은 구현되어 있지 않습니다. 이력은 계속 쌓입니다. 관련 함수는 커기·목록·트리거뿐입니다.

SQL

```
SELECT prname FROM pg_proc WHERE prname LIKE 'og%temporal%' ORDER BY 1;
```

결과

prname

og__temporal

og_enable_temporal

og_temporal_types

(3 rows)

9단계. 누가, 어디서 — 스탬프와 감사 로그

`og_enable_temporal` 은 스탬핑도 함께 켭니다. `og_meta(id)` 는 현재 행의 시스템 컬럼을 돌려줍니다.

SQL

```
SELECT m->>'actor' AS actor, m->>'source' AS source, m->>'created_by' AS created_by, m->>'valid' AS
valid
FROM (SELECT og_meta((og_cypher('corp', 'MATCH (p:Person {name: "ann"}) RETURN id(p) AS i')-
->>'i')::int8) AS m) x;
```

결과

actor	source	created_by	valid
lee@payroll		dev	["2023-01-01 00:00:00+00",infinity)

(1 row)

현재 행을 마지막으로 바꾼 것은 W4의 `lee@payroll` 이고, W4는 `og.source` 를 주지 않았으므로 `source` 는 비어 있습니다. `created_by` 는 애플리케이션이 주장한 값이 아니라 실제 DB 역할입니다. 감사 로그에도 같은 문맥이 남습니다.

SQL

```
SELECT m.label AS record, a.actor, a.source, left(regexp_replace(a.query, '\s+', ' ', 'g'), 58) AS
query
FROM og_data.og_audit a JOIN marks m ON m.at = a.at ORDER BY a.audit_id;
```

결과

record	actor	source	query
W1	kim@hr	hr-portal	[corp] CREATE (c:Company {name: 'acme'}) CREATE (:Person {
W2	kim@hr		[corp] MATCH (c:Company {name: 'acme'}) CREATE (:Person {n
W3	kim@hr		[corp] MATCH (p:Person {name: 'ann'}) SET p.salary = 200
W4	lee@payroll		[corp] MATCH (p:Person {name: 'ann'}) SET p.salary = 250
W5	kim@hr		[corp] MATCH (:Person {name: 'ann'})-[w:WORKS_AT]->() DELE
W6	kim@hr		[corp] MATCH (p:Person {name: 'bob'}) SET p.salary = 120

(6 rows)

Neo4j 드라이버에서는 `SET LOCAL` 대신 트랜잭션 메타데이터(`validAt`, `systemAt`, `validFrom`, `validTo`, `actor`, `source`)로 같은 설정을 넘깁니다 (매핑: `bolt/src/session.rs` 의 `TX_SETTINGS`, 검증: `tests/bolt-context/run.py`). 이 문서의 빌드는 Bolt 를 실행하지 않습니다. 이 매핑이 들어가기 전에 빌드된 게이트웨이는 키를 **조용히 무시하고 현재를 답하므로**, 처음 연결할 때 답을 아는 과거 시점 하나(예: 입사 전 → 0행)로 먼저 확인하세요.

10단계. 세 번째 축 — 그 행은 어떤 스키마 아래서 쓰였나

`Person` 에 `email` 을 추가한 뒤, W1 시각의 스키마와 지금의 스키마를 비교합니다.

```
SELECT og_add_property('corp', 'Person', 'email', 'string') \gset
SELECT at AS w1 FROM marks WHERE label = 'W1' \gset
SELECT og_schema_snapshot('corp', at => :w1) #> '{"types}" @> '{"name": "Person", "properties":
{"email": {}}}' AS email_at_w1,
      og_schema_snapshot('corp') #> '{"types}" @> '{"name": "Person", "properties":
{"email": {}}}' AS email_now;
```

```
email_at_w1 | email_now
-----+-----
f          | t
(1 row)
```

이력 행의 `__sys_from` 을 `at` 에 넣으면 그 믿음이 기록될 당시의 스키마입니다. 스키마 변경은 스키마 진화 튜토리얼에서 다룹니다.

Studio에서

편집기 옆의 시계 버튼이 시간 여행 막대를 엽니다. `valid at` 에 날짜를 넣거나 `as known at` 에서 감사 로그의 쓰기를 고르고 같은 질의를 다시 실행하면, 프레임에  `valid ... · known ...` 배지가 붙고 `compiled SQL` 지문은 그대로입니다.

- 막대는 읽는 시점만 정합니다. `og.valid_from` 을 주는 쓰기는 `:sql` 이나 드라이버로 합니다.
- `as known at` 의 시각은 밀리초로 잘려 전달됩니다. 엔진의 시스템 시각은 마이크로초라서, 고른 쓰기 자체는 포함되지 않고 그 직전 상태가 나옵니다 (실측, Studio 결함으로 보고).

한 장으로 다시 보기

하고 싶은 것	방법
타입에 시간 버전 커기	<code>SELECT og_enable_temporal('corp', 'Person')</code> — 엔티티·관계 타입, 타입마다. 끄는 함수 없음
과거·미래 시점부터 바뀜 (변경)	<code>SET LOCAL og.valid_from = '2023-01-01'</code> 뒤 평소의 <code>SET</code> / <code>DELETE</code>
원래부터 그 값이었다 (정정)	<code>SET LOCAL og.valid_from = '-infinity'</code> — 유효 구간은 그대로, 옛 값은 시스템 시간에만 남는다
그때 실제로 어땠나	<code>SET og.valid_at = '2023-06-01'</code> 뒤 같은 사이퍼
그때 우리는 무엇을 믿었나	<code>SET og.system_at = '<그때의 시각>'</code> 뒤 같은 사이퍼. 둘을 함께 쓸 수 있고, <code>RESET</code> 하면 현재
시점이 SQL을 바꾸지 않는지 확인	<code>md5(og_cypher_sql(graph, cypher))</code> 가 시점과 무관하게 같다
누가·어디서	쓰기 때 <code>og.actor</code> , <code>og.source</code> → <code>og_meta(id)</code> , <code>og_data.og_audit</code>
이력 직접 보기	<code>og_data.h_<type_id></code> — <code>__op</code> : <code>u</code> 대체된 믿음, <code>d</code> 삭제된 믿음, <code>v</code> 지금도 믿는 과거 구간

되지 않는 것	실제 동작
가변 길이 경로·라벨 없는 스캔·타입 없는 관계 변수·TypeQL의 시점 질의	<code>OG_TEMPORAL_UNSUPPORTED</code> 로 거부
켜지 않은 타입의 시점 질의	거부하지 않고 현재를 답한다
과거 유효 구간의 소급 편집	없음. 쓰기는 항상 현재 행을 바꾼다 (시점을 설정한 채 써도)
미래 날짜 변경 뒤 라벨 없는 패턴	시점을 정하지 않아도 미래 값을 답한다 — 라벨을 붙일 것
이력 보존 정책	미구현. 이력은 계속 쌓인다