

튜토리얼: 운영 중에 바꾸는 스키마 — 스키마 진화

이 문서가 답하는 질문

- 잘못 선언한 프로퍼티 타입을, 데이터를 잃지 않고 쓰기를 멈추지 않은 채 어떻게 고치는가?
- 옛 이름을 쓰는 애플리케이션이 살아 있는 동안 프로퍼티 이름을 어떻게 바꾸는가?
- 선택 필드를 필수로, 평평한 타입을 계층으로 바꾸는 데 행을 몇 개나 다시 쓰는가?
- 변경이 실패하면 무엇이 남고, 어떻게 되돌리며, 지난 스키마는 어디서 보는가?

처음이라면 데모 영상부터 보세요. Studio에서 운영 중인 그래프에서 잘못된 타입을 고치는 과정을 실제로 실행한 화면이고, 설명은 자막으로 들어 있습니다. 이 문서는 PDF 와 Word 문서 로도 받을 수 있습니다.

인사(HR) 그래프 하나에 요구 네 가지가 차례로 들어옵니다. ① 문자열로 선언해 버린 `age` 를 정수로 (`alter_property_type`), ② `name` 을 `fullName` 으로 — 옛 앱은 아직 `name` 을 씁니다(`rename_property`), ③ `age` 를 필수로(`set_required`), ④ 사람 위에 추상 상위 타입 `Party` 를(`add_parent`). 이 가운데 기존 행을 모두 다시 쓰는 것은 ①뿐입니다.

데이터가 있는 그래프에서 스키마 변경은 한 문장이 아니라 네 단계입니다. 변경 하나는 카탈로그의 한 행 (`og_catalog.schema_change`)이고, 아래 함수로 한 단계씩 나아갑니다.

단계	함수	하는 일
plan	<code>og_schema_plan(graph, change)</code>	변경을 기록하고 영향을 분석한다. 아무것도 바꾸지 않는다
expand	<code>og_schema_expand(id)</code>	옛 구조 옆에 새 구조를 추가한다. 이때부터 새 쓰기는 새 규칙을 지켜야 한다
migrate	<code>og_schema_migrate(id, batch)</code>	기존 행을 <code>batch</code> 행씩 옮긴다. 여러 번 부르고, 끊겨도 이어진다
validate	<code>og_schema_validate(id)</code>	위반 건수와 위반 행 id를 보고한다. 0건일 때만 다음으로 넘어간다
contract	<code>og_schema_contract(id)</code>	옛 구조를 지우고 새 구조가 이름을 이어받는다

각 kind의 단계별 동작·잠금·오류 코드 전체는 사용자 매뉴얼, 설계 배경은 ADR-028 에 있습니다.

준비

이 문서의 출력은 손으로 적은 것이 아닙니다. `python3 examples/tutorials/build_tutorial.py schema-evolution` 이 `tutorial.src.md` 의 블록을 빈 데이터베이스 (`tut_schema_evolution`)에서 차례로 실행해 출력을 끼워 넣습니다. 예제가 깨지면 빌드가 실패합니다.

- `cypher` 블록은 그래프 `hr` 에 보내는 사이퍼입니다. Studio 편집기에 그대로 넣어도 됩니다.
- `sql` 블록은 `psql` 로 실행합니다. 스키마 변경의 표면은 관리자용 SQL 함수이고, Studio에서는 `:sql <문장>` 으로 실행합니다.

- 변경 번호(`change_id`)와 노드 id는 빈 데이터베이스에서 시작했을 때의 값입니다.

1단계. 출발점 — 문자열로 선언된 `age`

SQL

```
SELECT og_create_graph('hr');
SELECT og_create_type('hr', 'Person', 'entity');
SELECT og_add_property('hr', 'Person', 'name', 'string');
SELECT og_add_property('hr', 'Person', 'age', 'string');      -- 실수: 나이를 문자열로
SELECT og_create_type('hr', 'Employee', 'entity', ARRAY['Person']);
SELECT og_create_index('hr', 'Person', 'age');
```

사람 넷과 직원 하나를 넣습니다. cy의 나이는 숫자가 아니고, dee는 나이가 없습니다.

CYPHER

```
CREATE (:Person {name: 'ann', age: '31'}), (:Person {name: 'bob', age: '42'}),
      (:Person {name: 'cy', age: 'abc'}), (:Person {name: 'dee'}),
      (:Employee {name: 'eve', age: '9'})
```

"마흔이 넘는 사람"을 찾습니다. 문자열 비교는 사전순이라 아홉 살 eve가 나옵니다.

CYPHER

```
MATCH (p:Person) WHERE p.age > '40' RETURN p.name AS name, p.age AS age, labels(p) AS labels ORDER BY
name
```

결과

name	age	labels
bob	42	["Person"]
cy	abc	["Person"]
eve	9	["Person", "Employee"]

(3 rows)

검산: 사전순으로 `'42'`, `'abc'`, `'9'` 는 모두 `'40'` 보다 뒤입니다. 맞는 답은 bob 한 명입니다.

2단계. plan — 바꾸기 전에 영향부터 본다

plan이 돌려주는 것은 변경 번호뿐이고, 분석 결과는 변경 기록의 `report` 에 있습니다.

SQL

```
SELECT og_schema_plan('hr',
'{"kind":"alter_property_type","type":"Person","property":"age","to":"int"}') AS change_id;
SELECT phase, report->>'from' AS "from", report->>'to_sql' AS "to", report->>'shadow' AS shadow,
report->'uncastable' AS uncastable, report->'uncastable_samples' AS samples
FROM og_schema_changes('hr') WHERE change_id = 1;
```

결과

```
change_id
-----
1
(1 row)

phase | from | to | shadow | uncastable | samples
-----+-----+---+-----+-----+-----
planned | text | int4 | p_age__s1 | 1 | [68719476739]
(1 row)
```

정수로 바꿀 수 없는 값이 1건이고, 그 노드 id가 샘플로 나왔습니다. 누구인지 확인합니다.

CYPHER

```
MATCH (p) WHERE id(p) = 68719476739 RETURN id(p) AS id, p.name AS name, p.age AS age
```

결과

```
id | name | age
---+---+---
68719476739 | cy | abc
(1 row)
```

단계는 `planned` 이고 아직 아무것도 바뀌지 않았습니다. 여기서 `og_schema_abort(1)` 로 그만둬도 됩니다. 주의: plan은 영향을 세느라 테이블을 스캔합니다. 큰 테이블에서는 plan도 시간대를 골라 부르세요.

3단계. expand — 새 컬럼을 옆에 두고 이중으로 쓴다

SQL

```
SELECT og_schema_expand(1)->>'lock' AS lock;
```

결과

```
lock
-----
brief ACCESS EXCLUSIVE at expand (ADD COLUMN) and contract (DROP/RENAME)
(1 row)
```

`Person` 과 하위 타입 `Employee` 의 테이블에 그림자 컬럼 `p_age__s1 int4` 가 생기고, 이때부터 쓰기는 옛 컬럼과 그림자 컬럼을 모두 채웁니다. 그래서 **새 쓰기는 지금부터 정수로 바뀔 수 있어야** 합니다.

```
CREATE (:Person {name: 'fay', age: 'old'})
```

CYPHER

```
ERROR:  invalid input syntax for type integer: "old"
```

결과

이중 쓰기는 엔진이 아니라 테이블에 붙은 트리거가 합니다. 사이퍼를 거치지 않는 SQL `UPDATE` 도 똑같이 막힙니다.

```
UPDATE og_data.n_1 SET p_age = 'forty' WHERE p_name = 'bob';
```

SQL

```
ERROR:  invalid input syntax for type integer: "forty"
```

결과

바뀔 수 있는 값은 들어갑니다. 그리고 읽기는 여전히 옛 컬럼을 봅니다. 1단계의 질의는 gus가 더해졌을 뿐 아직 사전순으로 답합니다. 옛 애플리케이션에는 아무 변화가 없습니다.

```
CREATE (:Person {name: 'gus', age: '60'})
```

CYPHER

```
MATCH (p:Person) WHERE p.age > '40' RETURN p.name AS name, p.age AS age ORDER BY name
```

CYPHER

```
name | age
-----+-----
bob   | 42
cy    | abc
eve   | 9
gus   | 60
(4 rows)
```

결과

4단계. migrate — 배치로 옮기고, 끊겨도 이어서

운영에서는 `batch` 를 수천~수만으로 두고 호출마다 커밋합니다. 여기서는 배치가 보이도록 2행씩 옮깁니다.

```
SELECT og_schema_migrate(1, 2) AS first_call;
SELECT report->'cursor' AS cursor_after_first_call, phase FROM og_schema_changes('hr') WHERE
change_id = 1;
SELECT og_schema_migrate(1, 2) AS second_call;
SELECT og_schema_migrate(1, 2) AS third_call;
```

first_call

```
-----
{"done": false, "moved": 3, "migrated": 3, "change_id": 1}
(1 row)
```

cursor_after_first_call | phase

```
-----+-----
{"og_data.n_1": 68719476738, "og_data.n_2": 137438953473} | migrating
(1 row)
```

second_call

```
-----
{"done": false, "moved": 2, "migrated": 5, "change_id": 1}
(1 row)
```

third_call

```
-----
{"done": true, "moved": 0, "migrated": 5, "change_id": 1}
(1 row)
```

- 검산: 배치는 **테이블마다** 적용됩니다. 첫 호출은 `Person` 에서 2행(ann, bob), `Employee` 에서 1행(eve)이라 3행입니다.
- `cursor` 는 테이블별로 마지막에 옮긴 노드 id입니다. 여기서 세션이 끊겨도 다음 호출은 이 지점부터 이어집니다.
- 두 번째 호출이 남은 2행(cy, gus)을 옮겼습니다. 나이가 없는 dee는 옮길 것이 없어 세지 않습니다. 검산: $3 + 2 = 5$.
- `done` 은 "이번 호출에서 옮긴 행이 없다"는 뜻이라, 마지막에 한 번 더 불러야 `true` 가 됩니다.
- cy도 `migrated` 에 세어졌지만 `'abc'` 는 정수가 될 수 없어 그림자 값이 비어 있습니다. 그것을 잡는 것이 `validate`입니다.

5단계. validate 와 contract — 막는 행을 id로 알려 준다

```
SELECT og_schema_validate(1) AS validate;
```

validate

```
-----
{"phase": "migrating", "samples": [68719476739], "change_id": 1, "violations": 1}
(1 row)
```

위반 1건, 2단계에서 본 cy의 id입니다. 단계는 `migrating` 에 머물고, 이 상태에서 `contract`는 거부됩니다.

```
SELECT og_schema_contract(1);
```

SQL

```
ERROR:  OG_SCHEMA_PHASE: change 1 is 'migrating'; contract needs it to be validated
```

결과

데이터는 **평소 쓰던 방법으로** 고칩니다. 이중 쓰기가 그림자 컬럼도 채우므로 다시 migrate할 필요가 없습니다.

```
MATCH (p:Person {name: 'cy'}) SET p.age = '38'
```

CYPHER

```
SELECT og_schema_validate(1) AS validate;  
SELECT og_schema_contract(1)->>'shadow' AS shadow_that_became_p_age;
```

SQL

validate

```
-----  
{ "phase": "validated", "samples": [], "change_id": 1, "violations": 0 }  
(1 row)
```

```
shadow_that_became_p_age  
-----  
p_age__s1  
(1 row)
```

결과

같은 질문을 숫자로 다시 묻습니다. 이제 비교는 수치 비교입니다.

```
MATCH (p:Person) WHERE p.age > 40 RETURN p.name AS name, p.age AS age, labels(p) AS labels ORDER BY  
name
```

CYPHER

```
name | age | labels  
-----+-----+-----  
bob  | 42  | ["Person"]  
gus  | 60  | ["Person"]  
(2 rows)
```

결과

검산: 31, 42, 38, 60, 9 가운데 40보다 큰 것은 42(bob)와 60(gus)뿐입니다. 아홉 살 eve는 빠졌습니다. 옛 컬럼은 지워지고 그림자 컬럼이 `p_age` 라는 이름을 이어받았으며, 인덱스도 두 테이블의 새 컬럼에 다시 만들어졌습니다.

```
SELECT indexname, indexdef FROM pg_indexes WHERE schemaname = 'og_data' AND indexname LIKE 'ix%age'  
ORDER BY 1;
```

SQL

```
indexname | indexdef  
-----+-----  
ix_1_p_age | CREATE INDEX ix_1_p_age ON og_data.n_1 USING btree (p_age)  
ix_2_p_age | CREATE INDEX ix_2_p_age ON og_data.n_2 USING btree (p_age)  
(2 rows)
```

결과

6단계. name → fullName — 옛 앱이 살아 있는 동안

SQL

```
SELECT og_schema_plan('hr',  
'{"kind":"rename_property","type":"Person","property":"name","new_name":"fullName"}') AS change_id;  
SELECT og_schema_expand(2)->>'lock' AS lock;
```

결과

```
change_id  
-----  
2  
(1 row)  
  
lock  
-----  
catalog only; no table is rewritten  
(1 row)
```

expand는 같은 물리 컬럼에 이름을 하나 더 붙입니다. 새 앱은 `fullName` 으로 쓰고, 옛 앱은 `name` 으로 씁니다. 두 이름은 같은 값을 가리키고, 하위 타입에도 적용됩니다.

CYPHER

```
CREATE (:Person {fullName: 'hal', age: 20}), (:Employee {name: 'ivy', age: 27})
```

CYPHER

```
MATCH (p:Person) WHERE p.name IN ['hal', 'ivy']  
RETURN p.name AS name, p.fullName AS fullName, labels(p) AS labels ORDER BY name
```

결과

```
name | fullName | labels  
-----+-----+-----  
hal | hal | ["Person"]  
ivy | ivy | ["Person", "Employee"]  
(2 rows)
```

옛 앱을 모두 새 버전으로 바꾼 뒤 contract합니다. 옮길 행이 없으므로 migrate는 부르지 않습니다.

SQL

```
SELECT og_schema_validate(2)->>'phase' AS phase;  
SELECT og_schema_contract(2)->>'lock' AS lock;
```

결과

```
phase  
-----  
validated  
(1 row)  
  
lock  
-----  
catalog only; no table is rewritten  
(1 row)
```

```
MATCH (p:Person {fullName: 'ann'}) RETURN p.fullName AS fullName, p.name AS name
```

결과

```
fullName | name
-----+-----
ann      | null
(1 row)
```

contract 뒤의 `name` 은 선언되지 않은 프로퍼티라 `null` 입니다. 물리 컬럼은 여전히 `p_name` 이며 11단계의 스냅샷에서 확인합니다.

7단계. `age` 를 필수로

SQL

```
SELECT og_schema_plan('hr', '{"kind":"set_required","type":"Person","property":"age","default":0}'))
AS change_id;
SELECT report->'missing' AS missing, report->'missing_samples' AS samples, report->'lock' AS lock
FROM og_schema_changes('hr') WHERE change_id = 3;
SELECT og_schema_expand(3) IS NOT NULL AS expanded;
```

결과

```
change_id
-----
3
(1 row)

missing | samples | lock
-----+-----+-----
1       | [68719476740] | NOT VALID check at expand; VALIDATE takes SHARE UPDATE EXCLUSIVE (writes
continue)
(1 row)

expanded
-----
t
(1 row)
```

나이가 없는 사람이 1명(dee)입니다. expand는 `CHECK (p_age IS NOT NULL) NOT VALID` 를 붙였습니다. 기존 행은 검사하지 않지만 **새 쓰기에는 바로 적용**되므로, 정리하는 동안 위반이 더 늘지 않습니다.

```
CREATE (:Person {fullName: 'joe'})
```

결과

```
ERROR: new row for relation "n_1" violates check constraint "og_req_3"
DETAIL: Failing row contains (68719476743, null, joe, null).
```

migrate가 빈 값을 기본값 0으로 채우고, validate가 확인하고, contract가 `NOT NULL` 을 겁니다. 기본값을 주지 않으면 migrate는 아무것도 채우지 않으므로, validate가 알려 주는 행을 직접 고쳐야 합니다.

SQL

```
SELECT og_schema_migrate(3) AS migrate;
SELECT og_schema_validate(3)->>'phase' AS phase, og_schema_contract(3) IS NOT NULL AS contracted;
```

결과

```
migrate
-----
{"done": false, "moved": 1, "migrated": 1, "change_id": 3}
(1 row)

phase | contracted
-----+-----
validated | t
(1 row)
```

CYPHER

```
MATCH (p:Person {fullName: 'dee'}) RETURN p.fullName AS fullName, p.age AS age
```

결과

```
fullName | age
-----+-----
dee      | 0
(1 row)
```

8단계. 한 번에 — og_schema_apply

og_schema_apply 는 plan부터 contract까지를 **한 트랜잭션**에서 실행합니다. 검증에 실패하면 아무것도 남지 않습니다. email 을 새로 선언하고 ann에게만 값을 준 뒤, 기본값 없이 필수로 만들어 봅니다.

SQL

```
SELECT og_add_property('hr', 'Person', 'email', 'string');
SELECT * FROM og_cypher('hr', $$MATCH (p:Person {fullName: 'ann'}) SET p.email = 'ann@example.com'$$);
```

SQL

```
SELECT og_schema_apply('hr', '{"kind":"set_required","type":"Person","property":"email"}');
```

결과

```
ERROR: OG_SCHEMA_VIOLATION: 7 row(s) violate the change; nothing was applied. samples:
[68719476738,68719476739,68719476740,68719476741,68719476742,137438953473,137438953474]
DETAIL: use og_schema_plan / expand / migrate / validate to fix the data before contracting
```

검산: 사람은 8명(ann, bob, cy, dee, gus, hal, eve, ivy)이고 email이 있는 것은 ann뿐이라 7행입니다. 하위 타입의 행동 셉니다. 변경 기록까지 롤백되어 11단계의 이력에 4번이 없고, email 은 스냅샷에서 보듯 여전히 선택(required = false)입니다.

카탈로그만 바꾸는 변경은 한 번에 해도 됩니다. Employee 를 Staff 로 바꿉니다. 노드 id는 그대로입니다.

```
SELECT og_schema_apply('hr', '{"kind":"rename_type","type":"Employee","new_name":"Staff"}')->>'lock'  
AS lock;
```

lock

결과

```
catalog only; identifiers do not change  
(1 row)
```

CYPHER

```
MATCH (s:Staff) RETURN id(s) AS id, s.fullName AS fullName, labels(s) AS labels ORDER BY fullName
```

결과

id	fullName	labels
137438953473	eve	["Person", "Staff"]
137438953474	ivy	["Person", "Staff"]

(2 rows)

타입 이름 변경에는 프로퍼티와 달리 **두 이름이 함께 통하는 기간이 없습니다**. 이 순간부터 옛 라벨로는 아무것도 나오지 않습니다. 오류가 아니라 빈 결과이므로, 옛 라벨을 쓰는 앱이 남아 있지 않은지 먼저 확인하세요.

CYPHER

```
MATCH (e:Employee) RETURN e.fullName AS fullName
```

결과

```
fullName  
-----  
(0 rows)
```

행을 다시 쓰는 변경(`alter_property_type` 등)을 큰 테이블에 apply하면 expand가 잡은 잠금이 커밋까지 유지됩니다. 그런 변경은 3~5단계처럼 나눠 부르세요.

9단계. 추상 상위 타입 Party — 행은 한 줄도 다시 쓰지 않는다

조직도 사람과 함께 "당사자"로 다루려고, 추상 타입 `Party` 를 만들어 `Person` 위에 둡니다.

SQL

```
SELECT og_create_type('hr', 'Party', 'entity', ARRAY[]::text[], true); -- true = 추상  
SELECT og_add_property('hr', 'Party', 'since', 'date');
```

행이 다시 쓰이면 행 버전(`xmin`)이 바뀝니다. 변경 전의 `xmin` 을 적어 두고 변경 뒤와 비교합니다.

```

CREATE TEMP TABLE before AS SELECT id, xmin::text AS x FROM og_data.n_1 UNION ALL SELECT id,
xmin::text FROM og_data.n_2;
SELECT og_schema_plan('hr', '{"kind":"add_parent","type":"Person","parent":"Party"}') AS change_id;
SELECT report->'gains_properties' AS gains, report->'rows_touched' AS rows_touched, report->'lock'
AS lock
  FROM og_schema_changes('hr') WHERE change_id = 6;
SELECT og_schema_expand(6) IS NOT NULL AS expanded, og_schema_validate(6)->'phase' AS phase;
SELECT og_schema_contract(6) IS NOT NULL AS contracted;
SELECT count(*) AS "rows", count(*) FILTER (WHERE n.x IS DISTINCT FROM b.x) AS rewritten
  FROM before b LEFT JOIN (SELECT id, xmin::text AS x FROM og_data.n_1 UNION ALL SELECT id,
xmin::text FROM og_data.n_2) n USING (id);

```

```

change_id
-----
        6
(1 row)

gains  | rows_touched | lock
-----+-----+-----
["since"] | 0           | catalog only; inherited columns are added without a rewrite
(1 row)

expanded | phase
-----+-----
t        | validated
(1 row)

contracted
-----
t
(1 row)

rows | rewritten
-----+-----
  8  |         0
(1 row)

```

8행 가운데 다시 쓰인 행은 0입니다. 그런데도 모든 사람이 `Party` 로 조회되고, 물려받은 `since` 를 바로 쓸 수 있습니다. 날짜는 문자열로 넣습니다. `SET p.since = date('2020-03-01')` 는 오류 없이 `null` 을 넣으므로 쓰지 마세요.

```
MATCH (p:Person {fullName: 'ann'}) SET p.since = '2020-03-01'
```

```
MATCH (x:Party) RETURN x.fullName AS fullName, x.since AS since, labels(x) AS labels ORDER BY
fullName
```

결과

fullName	since	labels
ann	2020-03-01	["Person", "Party"]
bob	null	["Person", "Party"]
cy	null	["Person", "Party"]
dee	null	["Person", "Party"]
eve	null	["Person", "Staff", "Party"]
gus	null	["Person", "Party"]
hal	null	["Person", "Party"]
ivy	null	["Person", "Staff", "Party"]

(8 rows)

타입 계층의 구간 라벨만 다시 매겼고, 물려받은 컬럼은 재작성 없는 `ADD COLUMN` 입니다. 그래서 이 변경의 비용은 데이터 양과 무관합니다(측정). `Party` 는 추상이라 직접 만들 수는 없습니다.

CYPHER

```
CREATE (:Party {fullName: 'nobody'})
```

결과

```
ERROR: 'Party' is abstract and cannot be instantiated
```

10단계. 되돌리기 — `og_schema_abort`

contract 전이라면 언제든지 버릴 수 있습니다. `age` 를 다시 문자열로 바꾸려다 그만둡니다.

```
SELECT og_schema_plan('hr',
'{"kind":"alter_property_type","type":"Person","property":"age","to":"string"}') AS change_id;
SELECT og_schema_expand(7)->>'shadow' AS shadow_column;
SELECT og_schema_abort(7) AS abort;
SELECT count(*) AS shadow_columns_left FROM pg_attribute
WHERE attrelid = 'og_data.n_1'::regclass AND attname LIKE 'p_age\_s%' AND NOT attisdropped;
```

결과

```
change_id
-----
          7
(1 row)

shadow_column
-----
p_age__s7
(1 row)

          abort
-----
{"phase": "aborted", "change_id": 7}
(1 row)

shadow_columns_left
-----
          0
(1 row)
```

그림자 컬럼과 이중 쓰기 트리거가 사라졌습니다. contract까지 끝난 변경은 abort할 수 없고, 반대 방향의 새 변경을 계획해야 합니다.

11단계. 변경 이력과 버전별 스냅샷

변경 기록에는 누가 했는지와 시작·종료 스키마 버전이 함께 남습니다. Studio에서는 `:changes` 가 이 목록을 보여 줍니다.

```
SELECT change_id, kind, phase, coalesce(spec->>'property', spec->>'type') AS target, created_by,
       from_version AS from_v, to_version AS to_v
FROM og_catalog.schema_change
WHERE graph_id = (SELECT graph_id FROM og_catalog.graph WHERE name = 'hr') ORDER BY change_id;
```

결과

change_id	kind	phase	target	created_by	from_v	to_v
1	alter_property_type	contracted	age	dev	6	8
2	rename_property	contracted	name	dev	8	10
3	set_required	contracted	age	dev	10	12
5	rename_type	contracted	Employee	dev	14	16
6	add_parent	contracted	Person	dev	18	21
7	alter_property_type	aborted	age	dev	21	23

(6 rows)

4번이 없는 것은 8단계에서 롤백된 apply 때문입니다(번호를 매기는 시퀀스는 롤백되지 않습니다). 스키마 버전이 오를 때마다 그래프의 카탈로그 전체가 스냅샷으로 저장됩니다. 1번 변경 직전 버전(`from_v` = 6)의 `Person` 과 지금의 `Person` 을 나란히 봅니다. 버전을 주지 않으면 현재 스키마입니다(Studio에서는 `:snapshot 6` 과 `:snapshot`).

SQL

```
SELECT p.key AS property, p.value->>'type' AS type, p.value->>'column' AS "column", p.value->>'required' AS required
  FROM jsonb_array_elements(og_schema_snapshot('hr', 6)->'types') t, jsonb_each(t->'properties') p
 WHERE t->>'name' = 'Person' ORDER BY 1;
SELECT t->>'parents' AS parents, p.key AS property, p.value->>'type' AS type, p.value->>'column' AS "column", p.value->>'required' AS required
  FROM jsonb_array_elements(og_schema_snapshot('hr')->'types') t, jsonb_each(t->'properties') p
 WHERE t->>'name' = 'Person' ORDER BY 2;
```

결과

property	type	column	required
age	text	p_age	false
name	text	p_name	false

(2 rows)

parents	property	type	column	required
["Party"]	age	int4	p_age	true
["Party"]	email	text	p_email	false
["Party"]	fullName	text	p_name	false
["Party"]	since	date	p_since	false

(4 rows)

`fullName` 의 컬럼이 여전히 `p_name` 인 것이 "이름 변경은 카탈로그만 바꾼다"의 증거입니다. 스냅샷은 시각으로도 조회합니다. 2번 변경(이름 변경)을 계획한 시각의 스키마에는 `fullName` 이 아직 없습니다.

SQL

```
SELECT s->>'description' AS snapshot_written_by, t->'properties' ? 'name' AS has_name, t->'properties' ? 'fullName' AS has_fullname
  FROM og_schema_snapshot('hr', at => (SELECT created_at FROM og_catalog.schema_change WHERE change_id = 2)) s,
       jsonb_array_elements(s->'types') t
 WHERE t->>'name' = 'Person';
```

결과

snapshot_written_by	has_name	has_fullname
schema change 1 contract (alter_property_type)	t	f

(1 row)

12단계. 아직 안 되는 것

타입을 나누거나 합치는 변경은 없습니다. 노드 id가 타입 id를 품고 있어, 노드를 다른 타입으로 옮기려면 id를 새로 매기고 그 노드를 가리키는 인접 목록을 모두 다시 써야 합니다. 지원하는 kind는 오류 메시지에 나오는 열 가지가 전부입니다.

SQL

```
SELECT og_schema_plan('hr', '{"kind":"split_type","type":"Person","into":["Adult","Minor"]}');
```

결과

```
ERROR: unknown schema change kind 'split_type'. one of: add_property, drop_property,
rename_property, alter_property_type, set_required, add_unique, rename_type, alter_role, add_parent,
remove_parent
```

유니크는 계층 전체가 아니라 구체 타입(테이블)마다 강제됩니다. `email` 을 유니크로 만든 뒤 같은 주소를 써 봅니다.

SQL

```
SELECT og_schema_apply('hr', '{"kind":"add_unique","type":"Person","property":"email"}')-->'note' AS
note;
```

결과

```
note
-----
one unique index per concrete table, as og_add_property(is_key) builds
(1 row)
```

CYPHER

```
CREATE (:Person {fullName: 'ann2', age: 31, email: 'ann@example.com'})
```

결과

```
ERROR: duplicate key value violates unique constraint "uq_1_p_email"
DETAIL: Key (p_email)=(ann@example.com) already exists.
```

`Person` 안에서는 막히지만, 하위 타입 `Staff` 에 같은 주소를 쓰는 것은 막지 못합니다. `validate`는 계층 전체에서 중복을 찾지만, `contract` 뒤의 강제는 테이블마다 하나씩 만든 유니크 인덱스이기 때문입니다.

CYPHER

```
CREATE (:Staff {fullName: 'kim', age: 45, email: 'ann@example.com'})
```

CYPHER

```
MATCH (p:Person {email: 'ann@example.com'}) RETURN p.fullName AS fullName, labels(p) AS labels ORDER
BY fullName
```

결과

```
fullName | labels
-----+-----
ann      | ["Person", "Party"]
kim      | ["Person", "Staff", "Party"]
(2 rows)
```

TypeQL `redefine` / `undefine` 은 이 파이프라인에 연결되어 있지 않습니다. `undefine` 은 구현되지 않았고, `redefine` 은 `define` 과 똑같이 처리되어 변경 기록을 남기지 않고 기존 데이터도 옮기지 않습니다. 스키마 변경은 위 SQL 함수로 하세요.

```
SELECT og_typeql('hr', 'undefine attribute email;');
```

SQL

```
ERROR: typeql error: 'undefine' is not implemented yet (spec 010 phase 5)
```

결과

그 밖에, 변경은 프로퍼티를 **선언한 타입에서만** 할 수 있고(`Staff` 에서 `age` 를 바꾸려 하면 `'age' is inherited from 'Person'; change it there`), 관계의 방향을 뒤집는 `kind`는 없습니다.

한 장으로 다시 보기

하려는 일	문장	행 재작성
영향만 보기	<code>og_schema_plan('hr', '{"kind":...}')</code> → <code>og_schema_changes('hr')</code> 의 <code>report</code>	없음 (스캔은 함)
타입 고치기	<code>alter_property_type</code> — plan → expand → migrate(반복) → validate → contract	값이 있는 모든 행, 배치로
이름 바꾸기	<code>rename_property</code> — expand와 contract 사이에 두 이름이 함께 통한다	없음
필수로 만들기	<code>set_required</code> (+ <code>"default"</code>) — expand부터 새 쓰기에 강제	비어 있던 행만
상위 타입 넣기	<code>add_parent</code> / <code>remove_parent</code>	없음
타입 이름 바꾸기	<code>rename_type</code> — 두 이름이 공존하는 기간 없음	없음
한 번에	<code>og_schema_apply(graph, change)</code> — 한 트랜잭션, 실패하면 아무것도 남지 않음	kind에 따라
그만두기	<code>og_schema_abort(id)</code> — contract 전까지	—
이력	<code>og_schema_changes(graph)</code> , <code>og_catalog.schema_change</code> · Studio :changes	—
지난 스키마	<code>og_schema_snapshot(graph, version)</code> / <code>(graph, at => 시각)</code> · Studio :snapshot	—

- 막히는 행은 validate가 id로 알려 주고, **평소의 쓰기로 고친 뒤** validate를 다시 부릅니다.
- 오류: 단계 순서 위반 `OG_SCHEMA_PHASE` (55000), apply 검증 실패 `OG_SCHEMA_VIOLATION` (23514).
- 안 되는 것: 타입 분할/병합, TypeQL `redefine` · `undefine` 연동, 계층 전체 유니크.
- 더 보기: 사용자 매뉴얼 · 크기별 실측 비용 · 스키마 버전을 시점 질의와 함께 쓰기는 바이템포럴 튜토리얼