

튜토리얼: 그래프 순회 중에도 적용되는 권한 — 내재된 보안

이 문서가 답하는 질문

- 그래프 질의를 보내는 "사용자"는 누구이고, 어디서 인증되는가?
- 어떤 역할이 볼 수 없는 노드가 **경로의 중간**에 있으면 순회 결과는 어떻게 되는가?
- 타입·연산·속성 단위의 쓰기 권한은 사이퍼를 거치지 않는 **원시 SQL** 에도 적용되는가?
- 읽기 권한만 가진 에이전트에게 "정해진 일 하나"만 시키고, 누가 왜 했는지를 남기려면 무엇을 선언하는가?

처음이라면 데모 영상부터 보세요. Studio에서 같은 질의가 역할에 따라 다른 답을 돌려주는 과정을 실제로 실행한 화면이고, 설명은 자막으로 들어 있습니다. 이 문서는 PDF 와 Word 문서 로도 받을 수 있습니다.

Ontological DB 는 PostgreSQL 확장입니다. 사이퍼는 SQL 로 컴파일되어 평범한 테이블을 읽고 씁니다. 그래서 권한을 검사하는 곳이 애플리케이션이나 게이트웨이가 아니라 **데이터베이스 자신**입니다. 이 문서는 그 말이 어디까지 성립하는지를, 허용되는 경우와 거부되는 경우를 짚지어 실행해 보입니다. 아직 닫히지 않은 곳(3단계의 한계, 8단계)도 함께 보입니다.

시나리오는 작은 병원입니다. 환자 셋, 의사 둘, 약 셋.

환자	병동	병실	진단	담당의	처방	알레르기
한지민	내과	301	폐렴	김내과	아목시실린	—
오세훈	내과	302	당뇨	김내과	메트포르민	아목시실린
윤하늘	정신건강	501	불면증	서정신	졸피뎀	—

역할은 넷이고, 모두 PostgreSQL 역할입니다.

역할	누구	이 문서에서 주는 권한
<code>tutsec_doctor</code>	의사	읽기 + 쓰기 전체. 정신건강 병동 환자도 본다
<code>tutsec_nurse</code>	내과 간호사	읽기. 쓰기는 환자의 <code>room</code> 속성 하나만. 정신건강 병동 환자는 보지 못한다
<code>tutsec_agent</code>	처방 보조 AI 에이전트	읽기. 쓰기는 행위(Action) <code>prescribe</code> 실행뿐
<code>tutsec_pharmacy</code>	행위의 소유 역할	사람이 로그인하지 않는다. <code>PRESCRIBED</code> 관계 생성 권한만 가진다

준비

역할은 클러스터 전체에 하나씩 있으므로, 다시 실행할 수 있도록 먼저 지우고 만듭니다. SQL 블록은 슈퍼유저로 접속한 `psql` 세션이고 `SET ROLE` 로 신원을 바꿉니다. 슈퍼유저 자신은 PostgreSQL 규칙대로 행 수준 보안을 건너뜁니다.

SQL

```
DROP ROLE IF EXISTS tutsec_doctor, tutsec_nurse, tutsec_agent, tutsec_pharmacy;
CREATE ROLE tutsec_doctor LOGIN PASSWORD 'doctor-pw'; CREATE ROLE tutsec_nurse LOGIN PASSWORD 'nurse-pw';
CREATE ROLE tutsec_agent LOGIN PASSWORD 'agent-pw'; CREATE ROLE tutsec_pharmacy NOLOGIN;
SELECT og_create_graph('hospital');
```

CYPHER

```
CREATE (kim:Doctor {name: '김내과', dept: '내과'}), (seo:Doctor {name: '서정신', dept: '정신건강의학과'}),
(amx:Drug {name: '아목시실린'}), (zol:Drug {name: '졸피뎴'}), (met:Drug {name: '메트포르민'}),
(p1:Patient {name: '한지민', ward: '내과', room: '301', diagnosis: '폐렴'}),
(p2:Patient {name: '오세훈', ward: '내과', room: '302', diagnosis: '당뇨'}),
(p3:Patient {name: '윤하늘', ward: '정신건강', room: '501', diagnosis: '불면증'}),
(p1)-[:TREATED_BY]->(kim), (p2)-[:TREATED_BY]->(kim), (p3)-[:TREATED_BY]->(seo),
(p1)-[:PRESCRIBED]->(amx), (p2)-[:PRESCRIBED]->(met), (p3)-[:PRESCRIBED]->(zol),
(p2)-[:ALLERGIC_TO]->(amx)
```

타입마다 저장 테이블이 하나씩 생겼습니다. 뒤에서 원시 SQL 로 직접 건드려 볼 테이블입니다.

SQL

```
SELECT name, kind, storage_table FROM og_type_view WHERE graph = 'hospital' ORDER BY type_id;
```

결과

name	kind	storage_table
Doctor	entity	og_data.n_1
Drug	entity	og_data.n_2
Patient	entity	og_data.n_3
TREATED_BY	relation	og_data.e_4
PRESCRIBED	relation	og_data.e_5
ALLERGIC_TO	relation	og_data.e_6

(6 rows)

1단계. 신원은 PostgreSQL 역할이고, 기본은 거부다

별도의 사용자 저장소가 없습니다. 역할을 만들기만 하고 아무것도 주지 않으면, 그 역할은 사이퍼 함수도 테이블도 쓸 수 없습니다. 확장의 함수는 설치 시점에 **PUBLIC** 실행 권한이 회수되어 있습니다(ADR-025).

SQL

```
SET ROLE tutsec_nurse;
SELECT * FROM og_cypher('hospital', 'MATCH (d:Drug) RETURN d.name');
SELECT * FROM og_data.n_3;
```

결과

```
ERROR: permission denied for function og_cypher
ERROR: permission denied for schema og_data
```

og_grant(역할, 수준) 이 수준별 권한을 한 번에 줍니다. 수준은 **read** < **write** < **admin** 입니다. 의사는 **write**, 나머지는 **read** 로 시작합니다.

SQL

```
SELECT og_grant('tutsec_doctor', 'write');
SELECT og_grant('tutsec_nurse', 'read'); SELECT og_grant('tutsec_agent', 'read'); SELECT
og_grant('tutsec_pharmacy', 'read');
```

읽기 역할은 이제 질의할 수 있습니다(2단계부터 봅니다). 그러나 스키마를 바꾸는 함수와 권한을 주는 함수는 부를 수 없습니다. 목록에 없는 함수는 `admin` 전용이므로, 새로 추가되는 함수도 누군가 풀어 주기 전에는 닫혀 있습니다. 마지막 줄만은 함수 권한이 아니라 쓰기 권한에서 걸립니다(4단계).

SQL

```
SET ROLE tutsec_nurse;
SELECT og_create_type('hospital', 'Surgery', 'entity');
SELECT og_enable_qls('hospital', 'Patient', 'true');
SELECT og_grant('tutsec_nurse', 'admin');
SELECT og_cypher('hospital', 'CREATE (:Drug {name: "모르핀"})');
```

결과

```
ERROR: permission denied for function og_create_type
ERROR: permission denied for function og_enable_qls
ERROR: permission denied for function og_grant
ERROR: OG_WRITE_DENIED: role 'tutsec_nurse' may not create 'Drug'
DETAIL: grant it with og_grant_write(role, graph, type, ops, props)
```

2단계. 행 수준 보안 — 보이지 않는 환자

정신건강 병동 환자는 의사만 보게 합니다. `og_enable_qls` 는 타입(과 하위 타입)의 저장 테이블에 PostgreSQL 의 행 수준 보안 정책을 겁니다. 정책은 그 테이블의 컬럼(`p_<속성>`)에 대한 SQL 조건식입니다. 건 뒤에 같은 질의를 의사와 간호사로 실행합니다.

SQL

```
SELECT og_enable_qls('hospital', 'Patient', $$p_ward <> '정신건강' OR current_user =
'tutsec_doctor'$$);
```

```

SET ROLE tutsec_doctor;
SELECT r->>'name' AS doctor_sees, r->>'ward' AS ward
  FROM og_cypher('hospital', 'MATCH (p:Patient) RETURN p.name AS name, p.ward AS ward ORDER BY
p.room') r;
SET ROLE tutsec_nurse;
SELECT r->>'name' AS nurse_sees, r->>'ward' AS ward
  FROM og_cypher('hospital', 'MATCH (p:Patient) RETURN p.name AS name, p.ward AS ward ORDER BY
p.room') r;

```

결과

```

doctor_sees | ward
-----+-----
한지민      | 내과
오세훈      | 내과
윤하늘      | 정신건강
(3 rows)

```

```

nurse_sees | ward
-----+-----
한지민      | 내과
오세훈      | 내과
(2 rows)

```

간호사에게는 오류가 나지 않습니다. 윤하늘은 **없는 행**입니다. 사이퍼를 거치지 않고 저장 테이블 `og_data.n_3` 을 직접 읽어도 두 행입니다(3단계 끝에서 썹니다).

3단계. 순회 중간에서도 — 경로가 통째로 사라진다

이 문서의 중심입니다. "의사별로 어떤 약이 처방되었는가"는 의사에서 출발해 약에서 끝나는 질의이고, 환자는 **경로의 중간**에 만 있습니다. 반환하는 것도 의사와 약뿐입니다.

```

SET ROLE tutsec_doctor;
SELECT r->>'doctor' AS doctor, r->>'drug' AS drug_seen_by_doctor FROM og_cypher('hospital',
'MATCH (d:Doctor)<-[:TREATED_BY]-(p:Patient)-[:PRESCRIBED]->(m:Drug)
RETURN d.name AS doctor, m.name AS drug ORDER BY drug') r;
SET ROLE tutsec_nurse;
SELECT r->>'doctor' AS doctor, r->>'drug' AS drug_seen_by_nurse FROM og_cypher('hospital',
'MATCH (d:Doctor)<-[:TREATED_BY]-(p:Patient)-[:PRESCRIBED]->(m:Drug)
RETURN d.name AS doctor, m.name AS drug ORDER BY drug') r;

```

결과

```

doctor | drug_seen_by_doctor
-----+-----
김내과 | 메트포르민
김내과 | 아목시실린
서정신 | 졸피뎀
(3 rows)

doctor | drug_seen_by_nurse
-----+-----
김내과 | 메트포르민
김내과 | 아목시실린
(2 rows)

```

검산: 서정신 → 윤하늘 → 졸피뎀 경로는 윤하늘을 지납니다. 간호사는 서정신 의사도 졸피뎀도 볼 수 있지만, 둘을 잇는 환자 행이 조인에서 빠지므로 경로 전체가 없습니다. 이 동작을 위해 엔진이 따로 한 일은 없습니다. 컴파일된 SQL 이 `Patient` 의 테이블을 조인하고, 그 테이블에 정책이 걸려 있을 뿐입니다.

한계 — 라벨 없는 중간 노드와 레지스트리

정책은 `Patient` 의 **저장 테이블**에 걸려 있습니다. 패턴이 그 테이블을 조인하지 않으면 정책도 평가되지 않습니다. 중간 노드의 라벨을 빼면 순회는 간선 레지스트리(`og_adj`)만으로 이어지고, 간호사에게도 경로가 남습니다.

```
SET ROLE tutsec_nurse;
SELECT r->>'doctor' AS doctor, r->>'patient' AS patient, r->>'labels' AS labels, r->>'drug' AS drug
FROM og_cypher('hospital',
  'MATCH (d:Doctor)-[:TREATED_BY]-(p)-[:PRESCRIBED]->(m:Drug)
  RETURN d.name AS doctor, p.name AS patient, labels(p) AS labels, m.name AS drug ORDER BY drug') r;
SELECT (SELECT count(*) FROM og_data.og_node) AS nodes_in_registry,
  (SELECT count(*) FROM og_data.n_1) + (SELECT count(*) FROM og_data.n_2) + (SELECT count(*)
FROM og_data.n_3) AS nodes_visible;
```

결과

```
doctor | patient | labels | drug
-----+-----+-----+-----
김내과 | 오세훈 | ["Patient"] | 메트포르민
김내과 | 한지민 | ["Patient"] | 아목시실린
서정신 |      | ["Patient"] | 졸피뎀
(3 rows)

nodes_in_registry | nodes_visible
-----+-----
                8 |              7
(1 row)
```

윤하늘의 이름은 여전히 비어 있습니다. 그러나 "서정신 의사에게 졸피뎀을 처방받은 Patient 가 한 명 있다"는 **연결 사실**은 드러납니다. 노드-간선 레지스트리의 읽기에는 행 수준 보안이 없기 때문입니다. 계산: 노드는 의사 2 + 약 3 + 환자 3 = 8개이고, 간호사가 저장 테이블에서 읽을 수 있는 것은 7 개입니다. 정리하면, 현재 구현에서 행 수준 보안이 숨기는 것은 **속성과, 그 타입을 이름으로 지나는 경로**입니다. 어떤 id 가 존재하고 무엇과 이어져 있는지까지 숨겨 주지는 않습니다(README의 "registry reads are still unrestricted"). 연결 사실 자체가 민감한 데이터라면 이 기능만으로는 부족합니다.

4단계. 쓰기 권한 — 타입·연산·속성 단위로

`read` 수준은 아무것도 쓸 수 없습니다. `og_grant_write(역할, 그래프, 타입, 연산, 속성)` 으로 필요한 만큼만 엽니다. 간호사는 환자의 병실만 바꿀 수 있습니다. 연산은 `create`, `set`, `delete` 입니다.

SQL

```
SELECT og_grant_write('tutsec_nurse', 'hospital', 'Patient', ARRAY['set'], ARRAY['room']);
```

SQL

```
SET ROLE tutsec_nurse;
SELECT r->>'name' AS name, r->>'room' AS room FROM og_cypher('hospital',
  'MATCH (p:Patient {name: "한지민"}) SET p.room = "305" RETURN p.name AS name, p.room AS room') r;
```

결과

```
name | room
-----+-----
한지민 | 305
(1 row)
```

진단을 고치는 것, 환자를 지우는 것, 처방 관계를 만드는 것은 거부됩니다. 오류는 무엇이 모자란지 말합니다.

SQL

```
SET ROLE tutsec_nurse;
SELECT og_cypher('hospital', 'MATCH (p:Patient {name: "한지민"}) SET p.diagnosis = "완치"');
SELECT og_cypher('hospital', 'MATCH (p:Patient {name: "한지민"}) DETACH DELETE p');
SELECT og_cypher('hospital', 'MATCH (p:Patient {name: "한지민"}), (m:Drug {name: "졸피뎴"}) CREATE (p)-[:PRESCRIBED]->(m)');
```

결과

```
ERROR: OG_WRITE_DENIED: role 'tutsec_nurse' may not set properties on 'Patient' (property 'diagnosis')
DETAIL: grant it with og_grant_write(role, graph, type, ops, props)
ERROR: OG_WRITE_DENIED: role 'tutsec_nurse' may not delete 'Patient'
DETAIL: grant it with og_grant_write(role, graph, type, ops, props)
ERROR: OG_WRITE_DENIED: role 'tutsec_nurse' may not create 'PRESCRIBED'
DETAIL: grant it with og_grant_write(role, graph, type, ops, props)
```

쓰기에도 2단계의 정책이 적용됩니다. 간호사가 윤하늘의 병실을 바꾸려 하면 오류 없이 0 행입니다 — 대상이 보이지 않기 때문입니다.

SQL

```
SET ROLE tutsec_nurse;
SELECT r->>'name' AS name FROM og_cypher('hospital',
  'MATCH (p:Patient {name: "윤하늘"}) SET p.room = "999" RETURN p.name AS name') r;
```

결과

```
name
-----
(0 rows)
```

원시 SQL 로 우회할 수 있는가

위 오류(OG_WRITE_DENIED)는 엔진이 먼저 친절하게 알려 준 것이고, 실제 방어선은 그 아래에 있습니다.

og_grant_write 는 저장 테이블의 컬럼 단위 PostgreSQL GRANT 로 옮겨집니다 (ADR-026). 그래서 엔진을 거치지 않는 SQL 도 같은 답을 받습니다.

SQL

```
SELECT grantee, privilege_type, column_name FROM information_schema.column_privileges
WHERE table_schema = 'og_data' AND table_name = 'n_3' AND grantee = 'tutsec_nurse' AND
privilege_type <> 'SELECT';
```

결과

```
grantee      | privilege_type | column_name
-----+-----+-----
tutsec_nurse | UPDATE        | p_room
(1 row)
```

SQL

```
SET ROLE tutsec_nurse;
UPDATE og_data.n_3 SET p_diagnosis = '완치' WHERE p_name = '한지민';
DELETE FROM og_data.n_3;
INSERT INTO og_data.n_2 (id, p_name) VALUES (1, '모르핀');
```

결과

```
ERROR: permission denied for table n_3
ERROR: permission denied for table n_3
ERROR: permission denied for table n_2
```

허용된 컬럼(`p_room`)은 원시 SQL 로도 바뀌고, 그때도 2단계의 정책 때문에 보이는 두 행만 대상이 됩니다.

5단계. 누가 썼는가 — 스탬프와 감사 기록

`og_enable_stamping` 을 켜 타입은 행마다 누가 언제 썼는지를 갖습니다. 값은 행에 써넣는 것이 아니라 **세션에서** 옵니다. `created_by` / `updated_by` 는 데이터베이스가 아는 역할이고, `actor` · `source` 는 애플리케이션이 세션 설정으로 알려 주는 값입니다.

스탬프를 켜 뒤, 의사가 새 환자를 입원시킵니다. 왜 하는지(`og.reason`)도 함께 알립니다. 행의 스탬프는 `og_meta(id)` 로 읽습니다.

SQL

```
SELECT og_enable_stamping('hospital', 'Patient');
SELECT og_enable_stamping('hospital', 'PRESCRIBED');
SET ROLE tutsec_doctor;
SET og.actor = '김내과'; SET og.source = '외래-EMR'; SET og.reason = '응급 입원';
SELECT og_cypher('hospital', 'CREATE (:Patient {name: "정민호", ward: "내과", room: "303", diagnosis: "천식"})');
```

SQL

```
SELECT m->>'created_by' AS created_by, m->>'actor' AS actor, m->>'source' AS source, (m->>'created_at') IS NOT NULL AS has_time
FROM og_meta((SELECT (r->>'i')::int8 FROM og_cypher('hospital', 'MATCH (p:Patient {name: "정민호"})
RETURN id(p) AS i') r)) m;
```

결과

```
created_by | actor | source | has_time
-----+-----+-----+-----
tutsec_doctor | 김내과 | 외래-EMR | t
(1 row)
```

스탬프는 사이퍼 속성으로 쓸 수 없고(`SET p.__created_by = ...` 는 `OG_RESERVED_PROPERTY` 로 거부됩니다), 테이블 전체에 UPDATE 권한을 가진 의사가 원시 SQL 로 덮어써도 트리거가 되돌립니다. 아래 UPDATE 는 성공하지만 돌려받은 값은 그대로입니다.

SQL

```
SET ROLE tutsec_doctor;
UPDATE og_data.n_3 SET __created_by = 'mallory', __actor = 'forged', __created_at = '2000-01-01'
WHERE p_name = '정민호'
RETURNING p_name, __created_by, __actor, __created_at > '2020-01-01' AS time_kept;
```

결과

```
p_name | __created_by | __actor | time_kept
-----+-----+-----+-----
정민호 | tutsec_doctor | 김내과 | t
(1 row)
```

질의는 감사 로그 `og_data.og_audit` 에도 남습니다. 지금까지 간호사와 의사가 한 쓰기를 봅니다.

SQL

```
SELECT db_user, actor, reason, rows_out, substr(query, 12, 44) AS query
FROM og_data.og_audit
WHERE db_user <> 'dev' AND (query LIKE '%SET %' OR query LIKE '%CREATE %')
ORDER BY audit_id;
```

결과

```
db_user | actor | reason | rows_out | query
-----+-----+-----+-----+-----
tutsec_nurse | | | 1 | MATCH (p:Patient {name: "한지민"}) SET p.room =
tutsec_nurse | | | 0 | MATCH (p:Patient {name: "윤하늘"}) SET p.room =
tutsec_doctor | 김내과 | 응급 입원 | 0 | CREATE (:Patient {name: "정민호", ward: "내과", r
(3 rows)
```

- `db_user` 는 PostgreSQL 이 아는 역할이라 호출자가 고를 수 없습니다. `actor` · `reason` 은 세션이 **주장한** 값입니다.
- 간호사의 두 번째 행은 `rows_out = 0` 입니다. 보이지 않는 환자를 고치려던 시도입니다.
- 4단계에서 **거부된** 사이퍼 세 건은 없습니다. 오류로 끝난 문장은 트랜잭션과 함께 감사 행도 되돌려집니다 (PostgreSQL 서버 로그에는 남습니다).
- 이 로그는 엔진을 지나간 질의의 기록입니다. 원시 SQL 은 권한 검사는 받지만 여기에 적히지 않습니다.

`write` 수준인 의사도 자기 흔적은 고치지 못합니다.

SQL

```
SET ROLE tutsec_doctor;
DELETE FROM og_data.og_audit;
UPDATE og_data.og_audit SET db_user = 'tutsec_nurse';
```

결과

```
ERROR: permission denied for table og_audit
ERROR: permission denied for table og_audit
```

6단계. 행위(Action) — 에이전트는 정해진 일만 한다

처방 보조 에이전트는 읽기 권한뿐이라, LLM 이 어떤 사이퍼를 만들어 내든 쓰기는 거부됩니다(아래 거부 목록의 첫 줄). 대신 **행위** 하나를 줍니다. 행위는 본문(사이퍼 한 문장), 파라미터 계약, 전제 조건을 가진 `SECURITY DEFINER` 함수이고, 소유

역할(`run_as`)의 권한으로 실행됩니다. 여기서는 "알레르기가 없는 약만 처방한다"입니다.

SQL

```
SELECT og_grant_write('tutsec_pharmacy', 'hospital', 'PRESCRIBED', ARRAY['create']);
SELECT og_define_action('hospital', 'prescribe',
  'MATCH (p:Patient {name: $patient}), (m:Drug {name: $drug})
  CREATE (p)-[:PRESCRIBED]->(m) RETURN p.name AS patient, m.name AS drug',
  'tutsec_pharmacy',
  params => '{"patient": "string", "drug": "string"}',
  precondition => 'MATCH (p:Patient {name: $patient})-[:ALLERGIC_T0]->(m:Drug {name: $drug}) RETURN
count(p) = 0 AS ok',
  description => '알레르기가 없는 약만 처방한다');
SELECT og_grant_action('tutsec_agent', 'hospital', 'prescribe');
```

허용되는 경우: 한지민에게 메트포르민.

SQL

```
SET ROLE tutsec_agent;
SET og.actor = 'rx-assistant'; SET og.reason = '야간 처방 보조';
SELECT r->>'patient' AS patient, r->>'drug' AS drug
  FROM og_run_action('hospital', 'prescribe', '{"patient": "한지민", "drug": "메트포르민"}') r;
```

결과

```
patient |      drug
-----+-----
한지민  | 메트포르민
(1 row)
```

거부되는 경우들입니다. 차례로 ① 에이전트가 직접 쓴 사이퍼, ② 오세훈은 아목시실린 알레르기가 있고(전제 조건), ③ `dose` 는 계약에 없는 파라미터이고, ④ `drug` 의 타입이 틀렸고, ⑤ 행위 뒤의 실행기는 직접 부를 수 없으며, ⑥ 간호사는 이 행위의 실행 권한을 받지 않았습니다.

SQL

```
SET ROLE tutsec_agent;
SELECT og_cypher('hospital', 'MATCH (p:Patient {name: "한지민"}), (m:Drug {name: "졸피뎴"}) CREATE (p)-
[:PRESCRIBED]->(m)');
SELECT og_run_action('hospital', 'prescribe', '{"patient": "오세훈", "drug": "아목시실린"}');
SELECT og_run_action('hospital', 'prescribe', '{"patient": "오세훈", "drug": "졸피뎴", "dose":
"900mg"}');
SELECT og_run_action('hospital', 'prescribe', '{"patient": "오세훈", "drug": 7}');
SELECT og__action_exec(1, '{"patient": "오세훈", "drug": "아목시실린"}');
SET ROLE tutsec_nurse;
SELECT og_run_action('hospital', 'prescribe', '{"patient": "한지민", "drug": "졸피뎴"}');
```

결과

```
ERROR: OG_WRITE_DENIED: role 'tutsec_agent' may not create 'PRESCRIBED'
DETAIL: grant it with og_grant_write(role, graph, type, ops, props)
ERROR: OG_ACTION_REJECTED: the precondition of action 'prescribe' did not hold for these parameters
ERROR: action 'prescribe' does not take a parameter named 'dose'
ERROR: action 'prescribe': parameter 'drug' must be a string
ERROR: permission denied for function og__action_exec
ERROR: permission denied for schema og_action
```

행위도 2단계의 정책 아래에 있습니다. 소유 역할 `tutsec_pharmacy` 는 정신건강 병동 환자를 볼 수 없으므로, 같은 호출을 윤하늘에게 하면 오류 없이 0 행이 돌아오고 아무것도 만들어지지 않습니다. 행위가 넓혀 주는 권한은 소유 역할이 가진 만큼입니다.

감사 로그에는 행위의 이름, 실제로 쓴 역할, 에이전트가 밝힌 이름과 이유가 함께 남습니다. 성공한 처방 한 건을 봅니다.

SQL

```
SELECT action, db_user, actor, reason, rows_out
FROM og_data.og_audit WHERE action = 'prescribe' AND query LIKE '%CREATE%' AND rows_out = 1;
```

결과

action	db_user	actor	reason	rows_out
prescribe	tutsec_pharmacy	rx-assistant	야간 처방 보조	1

(1 row)

제약: 행위의 표면은 SQL 함수(`og_define_action`, `og_grant_action`, `og_run_action`) 뿐이고 사이퍼 문장은 없습니다. `lang => 'typeql'` 본문도 정의할 수 있지만 **TypeQL 본문에는 파라미터가 전달되지 않아** (`engine/src/security/action.rs`), 고정된 한 문장만 실행할 수 있습니다.

7단계. Neo4j 드라이버로 접속해도 같은 역할이다

Bolt 게이트웨이에는 사용자 저장소가 없습니다. 드라이버가 보낸 사용자 이름과 비밀번호로 **PostgreSQL 에 그 역할로 접속**하고, 접속에 실패하는 것이 곧 인증 실패입니다. `tx_metadata` 의 `actor` · `source` · `reason` 은 `og.actor` 등으로 옮겨집니다. 아래 두 블록은 문서 빌드가 실행하지 않습니다. 이 문서가 만든 데이터베이스(`tut_security`)에 게이트웨이를 붙이고, 비밀번호 인증이 커진 경로로 따로 실행한 스크립트와 그 출력입니다.

PYTHON

```
from neo4j import GraphDatabase, Query
URL = "bolt://127.0.0.1:27699"
Q = "MATCH (d:Doctor)-[:TREATED_BY]-(p:Patient)-[:PRESCRIBED]->(m:Drug) RETURN DISTINCT d.name AS doctor, m.name AS drug ORDER BY drug"
for user, pw in [("tutsec_doctor", "doctor-pw"), ("tutsec_nurse", "nurse-pw"), ("tutsec_nurse", "wrong")]:
    try:
        with GraphDatabase.driver(URL, auth=(user, pw)) as drv, drv.session(database="hospital") as s:
            print(user, [(r["doctor"], r["drug"]) for r in s.run(Q)])
    except Exception as e:
        print(user, "->", e.code)
with GraphDatabase.driver(URL, auth=("tutsec_nurse", "nurse-pw")) as drv,
drv.session(database="hospital") as s:
    s.run(Query("MATCH (p:Patient {name: '오세훈'}) SET p.room = '310'",
        metadata={"actor": "station-3", "reason": "병실 이동"})).consume()
```

결과

```
tutsec_doctor [( '김내과', '메트포르민'), ('김내과', '아목시실린'), ('서정신', '졸피뎀')]
tutsec_nurse [( '김내과', '메트포르민'), ('김내과', '아목시실린')]
tutsec_nurse -> Neo.ClientError.Security.Unauthorized
```

마지막 쓰기의 감사 행은 `principal = tutsec_nurse`, `db_user = tutsec_nurse`, `actor = station-3`, `reason = 병실 이동` 으로 남았습니다 (앞 단계들은 슈퍼유저 세션에서 `SET ROLE` 한 것이라 접속 역할인 `principal` 이 `dev` 였습니다). 같은 세션에서 간호사가 `SET p.diagnosis` 를 보내면 4단계와 같은 `OG_WRITE_DENIED` 가 드러 이버 예외로 돌아옵니다.

8단계. 하지 못하는 것

항목	현재 상태	이 문서에서 본 곳
레지스트리 읽기 제한	없음. 노드.간선의 존재, 타입, 연결은 읽기 역할 모두에게 보인다	3단계 "한계"
라벨 없는 패턴에 대한 행 수준 보안	속성은 가려지지만 경로는 남는다. 민감한 타입은 패턴에 라벨을 적어야 걸러진다	3단계 "한계"
슈퍼유저	PostgreSQL 규칙대로 행 수준 보안을 건너뛴다. 애플리케이션을 슈퍼유저로 접속시키면 2~3단계는 적용되지 않는다	준비
거부된 시도.원시 SQL 의 감사	<code>og_audit</code> 에 남지 않는다. 거부된 문장은 트랜잭션과 함께 되돌려지고, 원시 SQL 은 엔진을 지나지 않는다	5단계
<code>actor</code> · <code>source</code> · <code>reason</code>	세션이 주장하는 값. 위조할 수 없는 것은 <code>db_user</code> · <code>created_by</code> 다	5단계
행위의 표면	SQL 함수만. TypeQL 본문은 파라미터를 받지 못한다	6단계
정책식	저장 테이블의 컬럼명(<code>p_ward</code>)으로 쓰는 SQL. 타입당 정책 하나(<code>og_policy</code>)이며 다시 부르면 교체된다	2단계
Bolt 의 <code>tx_metadata</code>	현재 소스(<code>bolt/src/session.rs</code>)로 빌드한 게이트웨이에서만 전달된다. 그 전에 빌드된 바이너리는 조용히 무시한다	7단계

정리

권한을 검사한 것은 모두 PostgreSQL 이었습니다. 역할이 신원이고, 정책이 테이블에 걸려 있어 순회의 조인에서 행이 빠지고, 쓰기 권한은 컬럼 GRANT 라서 사이퍼든 SQL 이든 Bolt 든 같은 답을 받습니다. 엔진이 더한 것은 그 위의 얇은 층입니다 — 수준별 부여, 읽기 쉬운 거부 메시지, 스탬프 트리거, 감사 행, 행위.

한 장으로 다시 보기

SQL

```
-- 0. 신원은 PostgreSQL 역할. 아무것도 주지 않으면 아무것도 못 한다 (read < write < admin)
CREATE ROLE nurse LOGIN PASSWORD '...'; SELECT og_grant('nurse', 'read'); SELECT
og_revoke('nurse');
-- 1. 읽기: 행 수준 보안. 정책식은 저장 테이블의 컬럼(p_<속성>)에 대한 SQL
SELECT og_enable_rls('hospital', 'Patient', $$p_ward <> '정신건강' OR current_user = 'doctor'$$);
-- (d:Doctor)<-[:TREATED_BY]-(p:Patient)-[:PRESCRIBED]->(m:Drug) p 가 안 보이면 경로가 없다
-- (d:Doctor)<-[:TREATED_BY]-(p)-[:PRESCRIBED]->(m:Drug) 라벨이 없으면 경로는 남는다 (한계)
-- 2. 쓰기: 타입 · 연산(create|set|delete) · 속성. 컬럼 GRANT 이므로 원시 SQL 에도 적용
SELECT og_grant_write('nurse', 'hospital', 'Patient', ARRAY['set'], ARRAY['room']); --
og_revoke_write(...)
-- 3. 스탬프와 감사: 누가(db_user) · 누구를 대신해(actor) · 왜(reason). Bolt 에서는 tx_metadata
SELECT og_enable_stamping('hospital', 'Patient'); SET og.actor = '...'; SET og.reason = '...';
SELECT og_meta(<id>); SELECT * FROM og_data.og_audit;
-- 4. 행위: 파라미터 계약 + 전제 조건 + 소유 역할의 권한
SELECT og_define_action('hospital', 'prescribe', '<사이퍼 본문>', 'pharmacy',
    params => '{"patient": "string", "drug": "string"}', precondition => '<... RETURN ... AS ok>');
SELECT og_grant_action('agent', 'hospital', 'prescribe');
SELECT og_run_action('hospital', 'prescribe', '{"patient": "...", "drug": "..."}');
```

더 읽을 것: 스펙 014 · ADR-025 · ADR-026 · 보안 감사 후 수정 현황