



문장 범위 행 락 — 설계

행 락을 커밋이 아니라 문장 끝에 놓는다 · CUBRID 락 매니저 개선 Phase 1

2026-09 · hgryoo · 엔진팀 세미나

이 발표의 질문과 범위

이 발표를 들은 뒤, 문장 범위 행 락이 왜 이렇게 설계됐는지 — 어떤 대안을 재서 버렸고 어떤 근거로 성립하는지 — 를 말할 수 있다.

Part	질문	슬라이드
1 배경	락은 왜 필요하고 락 매니저는 무엇을 하는가 · 다른 엔진은 어떻게 하는가 · CUBRID 는 지금 어떻게 그 대가는 무엇인가	4-9
2 설계	무엇을 바꾸는가 · 왜 성립하는가 · 어떤 대안을 버렸는가 · 무엇이 달라지고 무엇이 그대로인가	11-21

- 변경의 본체는 PR #7881 (CBRD-27034) — DELETE·UPDATE 의 행 락 보유를 **커밋까지** → **문장 끝까지** 로 줄이고, 행 락이 지키던 대기는 그 행을 쓴 트랜잭션의 락이 받는다. 미커밋 인스턴스 엔트리 **150** → **1**
- 조각마다 배경의 어느 사실에 기대는지 두 장의 표로 보인다 (12·13번) — 주장이 아니라 유도다
- **적용 범위** — 서버 질의 실행기가 실행하는, MVCC 클래스의, 온라인 인덱스 빌드 중이 아닌, READ COMMITTED 의, savepoint·DDL 을 지나지 않은 트랜잭션의 가장 바깥 문장. 숫자는 여기서만 성립한다

Part 1

배경 — 락의 필요에서 CUBRID 의 대가까지

엄격한 2PL 이 요구하는 것은 성질이지 표현이 아니다

- 락이 막는 것은 하나다 — 미커밋 쓰기 위에 다른 쓰기가 올라가지 않는다 (lost update · dirty write)
- 엄격한 2PL 은 배타 락을 커밋까지 쥐라고 한다 — 연쇄 롤백을 끊기 위해서다
- MVCC 는 읽기를 락에서 풀지만 쓰기 충돌은 남긴다 — first-updater-wins 는 상대 트랜잭션의 결말을 기다린다
- 행 X 락 하나가 셋을 겹친다 — 충돌 검출 · 결말까지의 대기 · 롤백 보호
- "행마다 엔트리를 커밋까지" 는 그 성질의 한 표현이다 — 표현이 달라도 성질이 유지되면 규약은 지켜진다
- 락에는 두 축이 있다 — 획득 시점과 보유 기간. 이 변경은 보유 기간만 옮긴다

락 매니저는 자원의 이름을 모른다

- 락 테이블 = 자원 이름 → 요청 리스트의 해시, 그리고 트랜잭션이 쥔 것의 리스트 (교과서 구조 그대로)
- **자원은 이름이다** — 행도, 테이블도, 트랜잭션 식별자도 키가 되고 대기 큐·데드락 검출·해제가 따라온다
- 데드락 검출기는 **락 매니저 안의 대기만** 본다 — 래치 대기, 자체 스핀은 그래프에 없다
- 기아 방지(먼저 선 X 를 S 흐름이 앞지르지 못한다)도 락 큐 안의 대기에만 따라온다
- X 의 커밋까지 보유는 **정책**이다 — 락 매니저 안에 커밋 전 해제 경로가 이미 있다
- lock escalation 은 락 테이블의 메모리 밸브다 — 대가는 **건드린 적 없는 행까지** 막는 것

두 계열 — 소유는 레코드에, 대기는 트랜잭션에

락 테이블 계열 — 그림은 CUBRID

InnoDB · SQL Server(전통) · Db2 · CUBRID

힙 레코드

row1 · 헤더 — T1 이 쓰는 중

row2 · 헤더 — T1 이 쓰는 중

row3 · 헤더 — T1 이 쓰는 중

같은 사실 둘

락 테이블 — 행마다 엔트리 하나

row1 : T1 X

row2 : T1 X

row3 : T1 X

T2

T2 는 행의 락 엔트리를 기다린다

엔트리 수 = 고친 행 수 → 커밋까지 누적 → lock escalation 이 메모리 뺄브다

보유 기간은 정책 — InnoDB 는 READ COMMITTED 에서 비매칭 행을 문장 중에 놓는다

레코드 표시 계열

PostgreSQL(xmax) · Oracle(ITL)

레코드 — 소유 표시가 여기 있다

row1 · 헤더 소유자 = T1

row2 · 헤더 소유자 = T1

row3 · 헤더 소유자 = T1

락 매니저 — 트랜잭션당 자원 하나

T1 의 트랜잭션 락 (X)
XID 락 · TX enqueue

T2

T2 는 헤더를 읽고 T1 의 결말을 기다린다

엔트리 수 = 트랜잭션 수 → 상한이 없어 lock escalation 이 없다

이 변경 뒤의 CUBRID 는 여기로 온다 — 트랜잭션 락은 기존 락 테이블 안의 자원으로

- 레코드 표시 계열 — 소유는 레코드에, 대기 대상은 트랜잭션. lock escalation 할 대상이 없다
- 락 테이블 계열 — 행마다 엔트리, 대기 대상은 그 엔트리. 엔트리당 메모리가 큰 쪽은 lock escalation 을 둔다
- 락 테이블 계열 안에서도 두 이동 — InnoDB 는 보유 기간을 정책으로 다루고, SQL Server 는 대기 대상을 트랜잭션으로 옮겼다

같은 네 질문의 답이 엔진을 두 계열로 가른다

엔진	소유 표시	막힌 writer 의 대기 대상	보유 기간	lock escalation
PostgreSQL	튜플 헤더 xmax	트랜잭션 (XID 락)	트랜잭션 끝	없음
Oracle	블록 헤더 ITL	트랜잭션 (TX enqueue)	트랜잭션 끝	없음 — "never escalates"
SQL Server optimized locking	락 매니저	트랜잭션 (TID 락)	트랜잭션 끝	—
MySQL InnoDB	락 테이블 (페이지 비트맵)	행의 락 엔트리	RC 는 비매칭 행을 문장 중 해제	없음
SQL Server 전통 · Db2	락 매니저 · 락 리스트	행의 락 엔트리	트랜잭션 끝	5,000 · maxlocks
CUBRID (현재)	락 테이블	행의 락 엔트리	트랜잭션 끝	lock_escalation

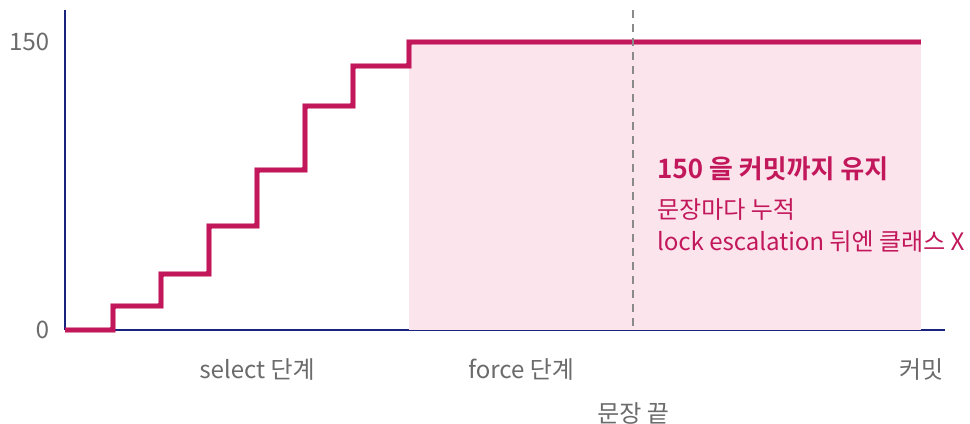
- 설계로 건너가는 원칙 넷 — 소유는 레코드에·대기는 트랜잭션에 / 트랜잭션 락은 기존 락 매니저의 자원으로 / 보유 기간은 정책 / lock escalation 은 락 테이블 계열의 증상

CUBRID — 락 테이블 계열이지만 소유 표시는 이미 두 곳에 있다

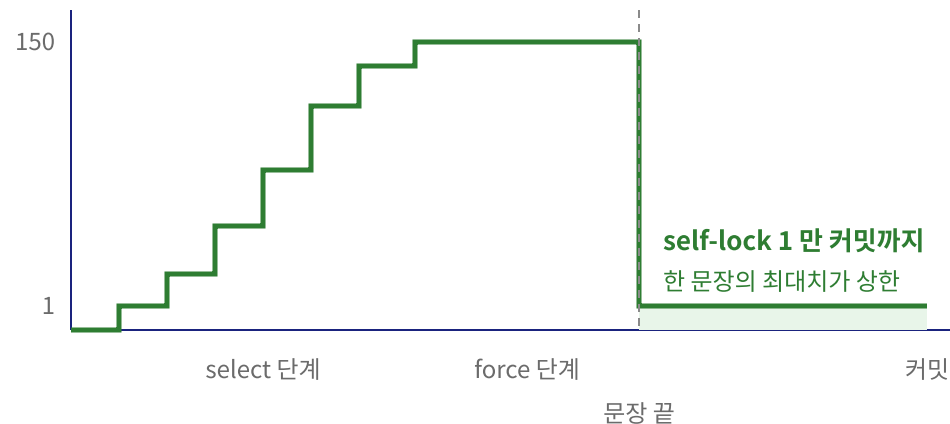
- `LK_ENTRY` 의 `count` 는 요청 수다 — 같은 행을 두 번 잡으면 엔트리 하나에 `count` 2, 획득과 해제는 1:1
- DELETE 는 두 단계다 — **select 단계**가 대상 OID 를 모으며 X 를 잡고, **force 단계**가 실제로 지우며 한 번 더 요청한다
- 그래서 지운 행 하나에 요청이 둘 붙는다 — 그리고 스캔은 자기가 DML 대상인지 `FOR UPDATE` 인지 모른다
- 문장은 겹친다 — MERGE 는 두 절을 각각 실행기 호출로 시스템 연산 안에서 돌린다
- lock escalation 은 인스턴스 엔트리를 회수하되 새 요청을 남기지 않고, 이후로는 엔트리를 만들지 않는다
- MVCC 스탬프(INSID·DELID)가 소유를 **레코드에 이미 적는다** — 행 락은 그 중복 사본이다
- INSERT 는 이미 삽입자의 MVCCID 를 자원 이름으로 잡는 락(self-lock)을 쓴다 (CBRD-26942) — 이 설계는 그것을 삭제·갱신으로 넓힌다

대가는 보유량이 아니라 보유 기간이다

오늘의 CUBRID — 커밋까지



이 변경 — 문장 끝까지



- 300행 중 150행을 건드린 미커밋 트랜잭션이 쥔 인스턴스 엔트리 **150** — 커밋까지, 문장마다 누적
- select 단계가 끝난 시점에 삭제자는 후보 전부를 쥐고 지운 행은 0 — 그 앞의 대기는 **아직 일어나지 않은 충돌**이고, 그 간선이 사이클을 닫으면 충돌한 적 없이 abort 된다
- lock escalation 뒤에는 건드린 적 없는 행까지 — 읽기 약 21,300 req/s 가 200,000행 DELETE(끝내 ROLLBACK) 하나에 **0 req/s 로 약 38초** — lock escalation 의 뭍이라 이 변경 밖이다 (16번)

Part 2

설계 — 대기 대상을 행에서 트랜잭션으로

한 문장 설계 — 세 보장을 스탬프와 self-lock 으로 옮긴다

행 락이 지키던 자리를 **행 헤더의 스탬프와 그 스탬프 주인의 MVCCID self-lock** 이 대신 지키고, 행 락 자체는 문장이 끝날 때 반납한다.

잃는 보장	무엇이 대신하나
다른 writer 가 그 행을 못 만진다	스탬프를 만난 자가 그 주인의 self-lock 을 기다린다
인덱스 검사자가 그 행 앞에서 멈춘다	unique·외래 키 검사가 같은 self-lock 을 기다린다
되돌릴 수 있다	self-lock 이 커밋까지 남아 롤백 중에도 대기가 유지된다

- 두 번째 writer 가 실제로 기다리는 것은 행이 아니라 **그 행을 쓴 트랜잭션의 결말**이다 — 결말을 직접 기다릴 자리를 두고 우회로를 걸어낸다
- 새 발명이 아니다 — PostgreSQL·Oracle 이 운영하는 형태이고 CUBRID 의 INSERT 경로가 이미 같다. **획득 시점은 그대로, 보유 기간만 바뀐다**

설계는 배경에서 유도된다 — 반납·self-lock·정착

조각	필요 — 어느 사실이 요구하나	자리 — 어느 사실이 허용하나	배제 — 어느 사실이 다른 형태를 막나
문장 끝 반납	대가는 전부 보유 기간 에서 온다 — 바꿀 것은 놓는 시점이다 (9번)	X 의 커밋까지 보유는 정책 이다 — 성질만 다른 수단으로 지키면 된다 (4·5번)	획득 시점을 옮기는 갈래는 판정과 기록 사이에 창을 연다 — 보유 기간만 옮긴다
self-lock	놓은 뒤 기다릴 자리 — 그 실체는 상대 트랜잭션의 결말 이다 (4번)	자원은 이름이다 · 스탬프와 INSERT 의 self-lock 이 이미 있다 (5·8번)	검출기는 락 매니저 안의 대기만 본다 — 래치·자체 스핀은 그래프 밖이다 (5번)
정착 지점	행 X 락이 겹하던 셋이 만나던 자리마다 대기가 놓여야 한다 (4번)	스탬프가 소유를 이미 적는다 (8번) — 그 자리들이 읽는 값이고, 활성이면 주인을 기다렸다 다시 판정	놓고 기다리면 한 번의 대기 보장을 잃는다 — 쥔 채 기다린다 (5번)

- 사실이 정하지 않은 것은 셋 — 놓는 시점(게시 직후인가 문장 끝인가 → 17번의 측정이 정했다), self-lock 의 단위와 모드, 정착 대기를 쥔 채 하는가(그 대가가 재접촉이다)

설계는 배경에서 유도된다 — 셈·문장 경계·스캔·생략

조각	필요	자리	배제
셈	이 문장이 잡은 요청만 돌려줘야 한다 — <code>count</code> 는 요청 수다 (8번)	행 하나에 요청 둘 — 표시는 깃발이 아니라 수여야 하고, 엔트리에 얹힌다 (8번)	깃발은 돌을 못 가르고, 별도 장부는 O(행) 메모리를 다른 자리에 다시 만든다
문장 경계	문장은 겹친다 — 안쪽이 끝날 때 반납하면 바깥 것까지 돌려준다 (8번)	실행기는 문장의 진입·종료를 안다 — 깊이 하나로 가장 바깥만	트랜잭션에 대상 클래스 OID 를 적어 두는 안은 겹친 문장이 남의 것을 지운다
스캔의 자기 성질	select 단계 요청도 셈에 들어야 하는데 스캔은 자기가 DML 대상인지 모른다 (8번)	판별자는 이미 있다 — UPDATE·DELETE 만이 구동 select 에 세우는 카운터	OID 안과 같다 — 프루닝된 파티션에서 영원히 불일치
소유자 재접촉의 행 락 생략	전 채 기다리는 선택의 대가 — 소유자가 자기 대기자에 막힌다	스탬프와 self-lock 이 이미 답이다 — 소유자 유일 (19번)	대기 중 해제·S 프로브는 보장 상실·변환 교착 (18번)

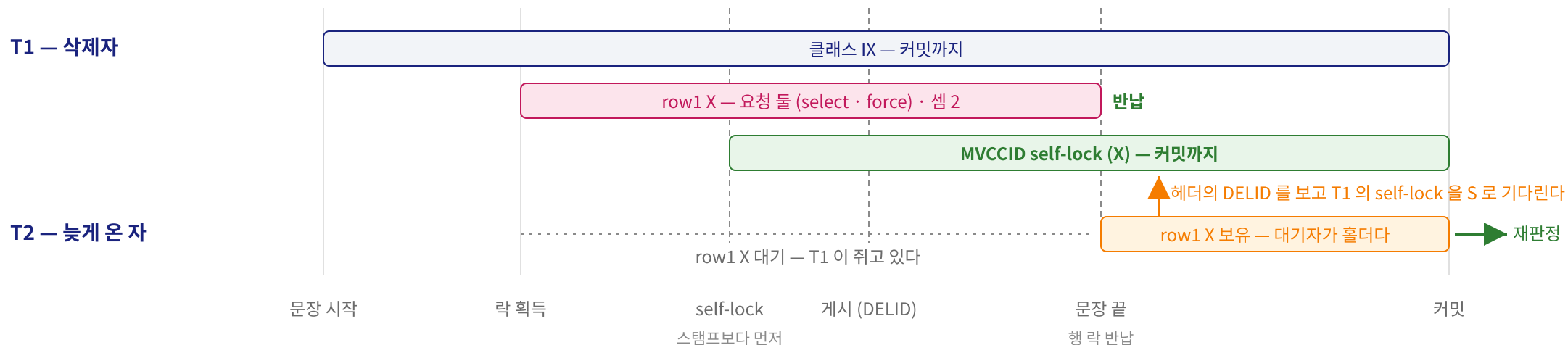
- 네 조각은 전부 **회계**다 — 어느 락이 이 문장의 것인가. select 무락화를 미룬 대가로 이 설계가 그 질문을 스스로 답한다

새로 생기는 여섯 조각

조각	무엇을 지는가	어느 모듈
MVCCID self-lock	행 락이 없어진 자리의 대기 대상 — 자원 타입 <code>LOCK_RESOURCE_TRANSACTION</code>	락 매니저 · 트랜잭션 테이블
정착 지점	대기가 실제로 놓이는 네 자리 — 힙 최신 버전, unique 검사, 외래 키 검사, 삭제 스탬프 소유자 검색	로케이터 · B-tree
셈 (<code>transient_count</code>)	문장 끝에 돌려줄 요청의 수. 엔트리에 얹히고 <code>셈 < count</code> 가드가 앞선 문장의 요청을 지킨다	락 매니저
문장 경계	셈은 트랜잭션 단위인데 반납은 문장 단위 — 깊이를 세어 가장 바깥 문장만 반납한다	락 매니저 · 질의 실행기
스캔의 자기 성질	select 단계의 요청까지 세려면 스캔이 자기가 DML 대상인지 들고 다닌다	스캔 매니저
소유자 재접촉의 행 락 생략	소유자가 자기 스탬프가 아직 살아 있는 행(self-lock 이 서 있는 동안)을 다시 건드릴 때는 행 락을 잡지 않는다 — 이것이 없으면 소유자의 다음 문장이 자기 대기자에 막힌다. 조건부 락 실패 뒤에만 판정	로케이터

- 코드는 이 발표 밖이다 — 표는 모듈 열까지만 보이고, 구현은 설계 독본의 모듈·구현 장에 있다

한 행이 겪는 순서 — self-lock 은 스탬프보다 먼저, 행 락보다 늦게



- 삭제자는 DELID 를 새기기 **전에** 자기 MVCCID 를 X 로 주고, 문장 끝에 행 락을 반납하고, 커밋에서 self-lock 을 놓는다
- 늦게 온 자는 행 X 를 **먼저 잡고** 헤더의 스탬프를 만나 주인의 self-lock 을 S 로 기다렸다 깨어나 다시 판정한다 — 대기자가 홀더가 된다
- 실패한 문장은 다르다 — 락은 커밋까지 쥐고 **셈만 지운다**

반납을 포기하는 다섯 자리, 그리고 lock escalation

자리	왜 반납하지 않나
온라인 인덱스 빌드 중인 클래스	항목 상태가 INSID 슬롯을 점유해 MVCCID 를 담지 못한다 — 두 writer 를 가르는 것은 행 락뿐이라 커밋까지 유지한다
savepoint 를 지난 트랜잭션의 문 장	<code>ROLLBACK TO SAVEPOINT</code> 가 트랜잭션을 끝내지 않고 게시된 삭제를 되돌린다 — 트랜잭션이 끝나지 않았으므로 락을 놓을 수 없다
MERGE	두 절이 시스템 연산 안에서 도는 한 문장 — 어느 절도 문장 끝을 선언하지 못한다
REPEATABLE READ 이상	문장이 닿지 않은 행의 락도 유지될 것을 기대한다 — 문장 끝 반납은 READ COMMITTED 이하에서만
<code>SELECT ... FOR UPDATE</code>	그 락은 문장의 것이 아니라 트랜잭션의 것이다 — 처음부터 썸에 넣지 않는다

- **lock escalation 은 이 설계 밖이다** — 회수된 엔트리의 썸은 함께 사라지고, 클래스 락을 원래 모드로 내리는 총(강등)은 다음 PR (CBRD-27238) 이다

반납 시점은 문장 끝이다 — 게시 직후가 아닌 이유

- 게시 직후에 놓는 안 — "뒤에 온 자는 정착하므로 행 락은 할 일이 없다" 는 문장 이후의 writer 에는 맞고, 문장 도중의 writer 에는 틀리다
- 인덱스에서 옛 키에 삭제 스탬프를 찍고 새 키를 넣은 뒤 락을 놓으면, 그 락을 얻은 T2 가 힙 헤더로 판별하는데 힙의 최신 버전은 확정된 옛 값이라 통과한다 — 이미 지워진 인덱스 엔트리를 지우려다 죽는다

워크로드 (JDBC 부하)	즉시 해제	문장 끝 반납
원본, autocommit	3 런 12건 실패	3 런 0건
문장마다 명시 커밋 — 문장 종료와 커밋 사이에 클라이언트 왕복	3 런 1건 재현	8 런 0건
한 트랜잭션에 7 문장	0건	0건 — 경합이 사라져 판별력 없음

- 셋째 줄은 판별력이 없다 — 경합이 사라져 두 안이 같은 값을 낸다. 대조군 없이 봤다면 처방이 통했다고 잘못 읽었을 자리다

기각한 여섯 대안 — 근거는 측정이거나 논증이다

대안	왜 버렸나	되살아나는 조건
게시 직후 즉시 해제	문장 도중 의 writer 에 틀리다 (부하 12건 실패 — 17번)	스탬프 undo 가 대기자를 깨우거나, 대기자가 행 X 없이 정착할 때
페이지 래치로 게시 전 원자성	다중 페이지에서 래치가 끊겨 창을 메우는 코드가 커지고(1,771줄), 래치 대기는 검출기에 보이지 않는다 (실측 22.64초 → 156.179초)	없다
행 락을 커밋까지 쥔 채 self-lock 만 더하기	정확하지만 발자국(쥔 락의 총수)이 통째로 되돌아온다	없다
기존 instant lock 머신 재사용	트랜잭션 전역 플래그라 중첩에서 남의 락을 풀고, 독립 커밋 뒤 푸는 것이라 논증이 다르다	반대 방향(문장 경계가 그 머신을 흡수)만
정착 대기 중 행 락을 놓기	한 번의 대기 보장을 잃고 주인이 커밋할 때 대기자 전원이 깨어 경합한다	행 락 생략을 넓혀도 안 닫히는 경로가 드러날 때
S 로 프로브하고 X 로 변환	두 writer 가 깨끗한 행을 동시에 칠 때 변환 교착을 새로 들인다	위와 같다

네 불변식 — 각각 논증과 검출기를 가진다

불변식	왜 성립하나	깨지면	무엇이 잡나
순서 — self-lock 은 스템프가 보이기 전에 잡히고 되돌려질 수 없게 된 뒤에 풀린다	삭제자가 스템프를 새기기 전에 자기 MVCCID 를 X 로 쥐고, 커밋까지 놓지 않는다 (15번)	스스템프는 보이는데 불잡을 self-lock 이 없는 창 — 대기 없는 미커밋 덧쓰기	대기 함수의 no-op 승인 단정 (debug 빌드)
귀속 — 문장 끝에 반납되는 요청은 전부 이 문장이 잡은 것이다	표시할 때 $sem < count$ 가드가 앞선 문장의 요청을 제외하고, 반납은 가장 바깥 문장만 한다 (14번)	앞선 문장의 요청이 대신 풀린다	반납 루프의 $n \leq count$ 단정 (가드와 별개) · shell 스위트
유지 — 트랜잭션을 끝내지 않고 스템프를 되돌리는 기전은 행 락을 유지한다	savepoint·MERGE 는 문장 끝을 선언하지 못하고 (16번), 실패한 문장은 셈만 지운다 (15번)	대기자가 홀더이므로 부분 롤백 뒤 주인과 서로를 기다린다	데드락 검출기 · savepoint 시나리오
소유자 유일 — self-lock 이 서 있는 동안 자기 스템프 행의 논리적 유일 변경자는 소유자다	다른 writer 는 스템프를 만나 self-lock 에 먼저 정착한다 (15번) — 그래서 소유자는 자기 행에 행 락을 잡지 않는다 (14번)	정착 지점을 지나지 않은 writer 가 소유자의 미커밋 변경 위에 덧쓴다	갱신 경로의 락 보유 단정 — "내가 삽입하지 않은 버전이면 락을 쥐고 있어야 한다" (debug 빌드) · 재접촉 시나리오

- 정확성 주장은 **논증**이 지고 실측은 반박하지 못했음을 말한다 — 이득 주장은 실측이 진다

효과 — 바뀌는 것, 바뀌지 않는 것, 나빠지는 것

현재의 문제	이 설계가 바꾸는 것	숫자 (#7881 빌드 · 154 시나리오)
미커밋 인스턴스 엔트리 150 이 커밋까지	문장 끝에 반납 — self-lock 하나만 남는다	150 → 1 , DELETE·UPDATE·파티션·계층·대상 둘 전부
문장이 끝난 뒤까지 이어지던 대기	문장이 끝난 뒤 도착한 세션은 막히지 않는다	대조군만 막히는 스텝 1
결과값	바뀌지 않는다 — 바꾸면 안 되는 것이다	결과값 차이 0

- **바뀌지 않는 것** — 같은 행에 대한 대기 시간(대상이 행에서 트랜잭션으로 바뀔 뿐) · 획득 비용 · lock escalation 이 세운 클래스 락 · 트리거·뷰를 통한 문장 · RR 이상과 **FOR UPDATE**
- **나빠지는 것** — 재접촉 교착 둘이 남는다: **ON DUPLICATE KEY UPDATE** 의 충돌 행 fetch, **SELECT ... FOR UPDATE** 로 자기 갱신 행을 다시 잡는 경로
- 그리고 진단 둘 — 타임아웃 메시지에서 행·테이블이 사라지고, **lock_escalation** 의 뜻이 "한 문장이 몇 행까지" 로 바뀐다
- 메모리와 종료 비용은 줄어드는 방향이지만 근거로 삼지 않는다 — lock escalation 이 상한을 이미 묶고 있다

검증 — 세 빌드, 두 축, 결과값 0

빌드	무엇
대조군	무수정 develop
#7881 빌드	이 변경 한 커밋 (CBRD-27034)
두 커밋 빌드	그 위에 lock escalation 강등 (CBRD-27238) — 막힘 18 중 17 은 이 층의 뭉

- 두 축 — **results** 는 0 이어야 하고 **blocking** 은 0 이 아니어야 한다. 154 시나리오, 트레이스 끄
- 발자국(**cubrid lockdb** 직접 계수)이 이 변경의 증거다 — 막힘 축이 아니다. 발자국과 시나리오가 함께 보지 않는 축 (엔트리 하나 안의 회계)은 shell 스위트가 잡았다
- 트리거·뷰를 통한 DELETE·UPDATE 는 클라이언트 경로다 — 이 설계가 하나도 적용되지 않고 행 락 150 을 커밋까지 쥘다
- 이웃 — lock escalation 강등(다음 PR, CBRD-27238) · select 무락화(미룬 갈래, 획득 시점의 축) · 외래 키(완료 — 부모 행 락 0, 교착 12 → 1)



Thank you

Q & A

- 설계 독본 — 문장 범위 행 락 18장, 락의 필요에서 구현·검증까지 (사내 설계 문서)
- 코드 — PR #7881 (CBRD-27034) · `src/transaction/lock_manager.c` · `src/transaction/locator_sr.c` · `src/query/query_executor.c`
- 이웃 — lock escalation 강등 CBRD-27238 · 외래 키 최적화 · select 무락화 설계