슬라이드 학습 가이드 — SQL 한 문장이 InnoDB 페이지를 만날 때까지

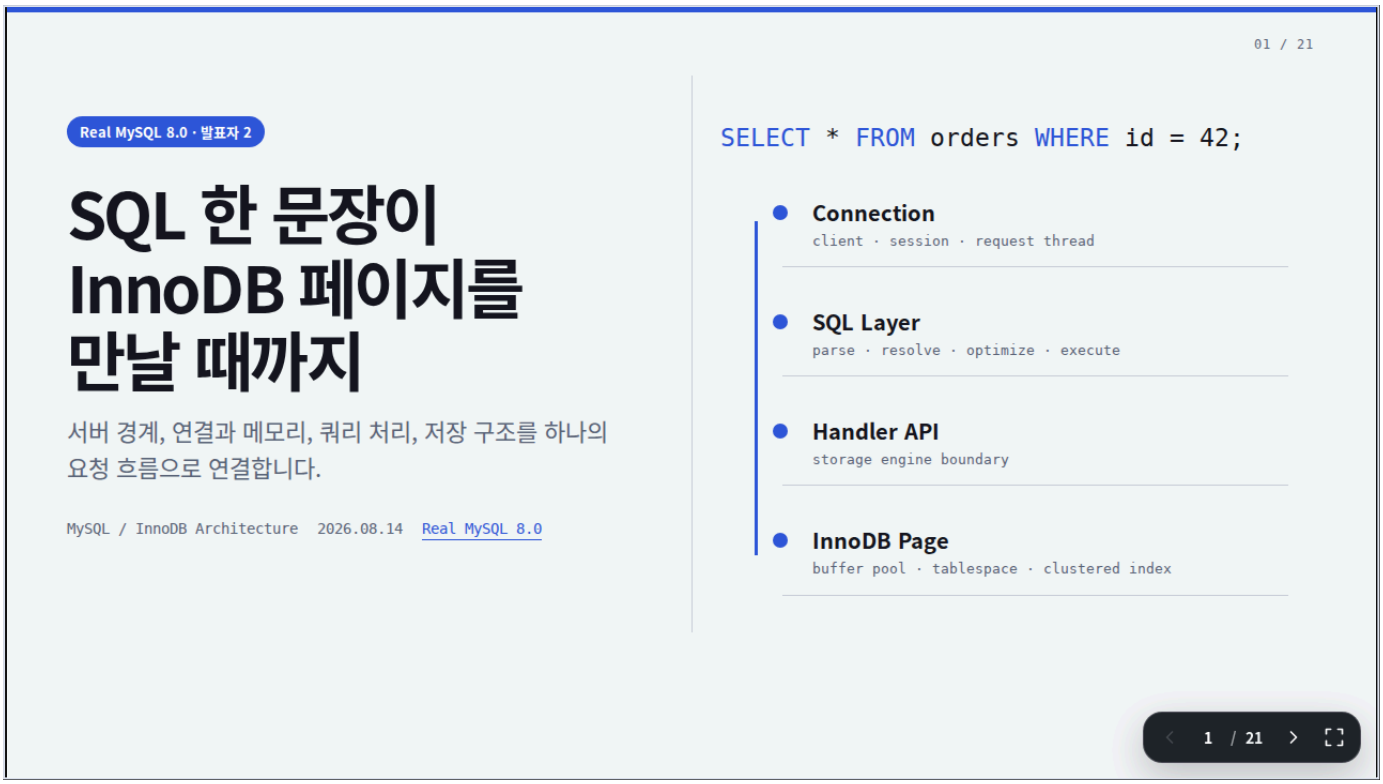
> 발표를 듣지 않아도 MySQL 서버와 InnoDB의 경계, SQL 처리 과정, 저장 구조, 저장 프로그램 객체를 순서대로 이해할 수 있도록 정리한 가이드입니다. 설명은 MySQL 8.0 공식 문서와 MySQL 8.0.46 Source Code Documentation을 기준으로 검증했습니다.

읽기 전에 알아둘 것

MySQL은 SQL을 이해하고 실행하는 **서버**와 실제 테이블 데이터를 읽고 쓰는 **스토리지 엔진**을 분리합니다. InnoDB는 MySQL 8.0의 기본 스토리지 엔진이지만 MySQL 서버 전체와 같은 뜻은 아닙니다. 이 구분이 가이드 전체의 출발점입니다.

또 하나의 기준은 **페이지(Page)**입니다. InnoDB는 행을 하나씩 디스크와 메모리 사이로 옮기지 않습니다. 여러 행과 인덱스 레코드가 들어 있는 페이지를 기본 I/O·캐시 단위로 씁니다. 기본 크기는 16KB이며 다른 크기는 인스턴스를 초기화할 때 정합니다.

1. SQL 한 문장이 InnoDB 페이지를 만날 때까지



핵심 개념

발표는 SQL 한 문장을 끝까지 추적합니다. 애플리케이션이 보낸 `SELECT`가 곧바로 데이터 파일을 읽는 것은 아닙니다. 연결과 세션을 거쳐 서버에 들어오면 SQL 구조와 의미를 확인하고, 옵티마이저가 실행 계획을 고릅니다. 실행기는 그 계획에 맞춰 스토리지 엔진에 필요한 행을 요청합니다. InnoDB는 Buffer Pool에서 페이지를 찾고, 없으면 Tablespace에서 읽습니다.

흐름과 의미

이 흐름을 알면 문제를 한 덩어리로 보지 않게 됩니다. 연결이 많아 생긴 메모리 문제인지, 옵티마이저가 잘못된 계획을 선택한 것인지, InnoDB가 많은 페이지를 읽는 것인지 나눠 볼 수 있습니다. 뒤의 모든 슬라이드는 이 경로의 한 구간을 확대합니다.

슬라이드가 생략한 세부 사항

화면은 읽기 중심의 대표 경로만 보여 줍니다. 실제 요청에는 인증, 메타데이터 잠금, 트랜잭션 가시성, 네트워크 전송과 cleanup도 끼어듭니다. 어느 단계가 병목인지 확인하려면 실행 계획과 서버·InnoDB 지표를 함께 봅니다.

확인 포인트

- Parser에서 Page까지의 순서를 말로 연결해 본다.
- 느린 요청을 Connection·Plan·Page 문제로 나눠 본다.

2. 오늘 따라갈 네 개의 경계

핵심 개념

첫 번째 경계는 **Client와 Connection** 사이입니다. 외부 프로그램이 인증된 통신 채널을 만들고 Session 문맥을 얻습니다. 두 번째는 SQL을 구조화하고 계획으로 바꾸는 **SQL 처리 경계**입니다. 세 번째는 Executor가 Handler API로 Storage Engine을 호출하는 **엔진 경계**입니다. 네 번째는 Buffer Pool과 Tablespace, Page가 만나는 **메모리·디스크 경계**입니다.

흐름과 의미

네 경계는 설명을 돕기 위한 큰 구분입니다. 실제 MySQL 소스 코드가 정확히 네 모듈로만 나뉜다는 뜻은 아닙니다. 특히 Preprocessor는 공식 구현의 고정된 단일 컴포넌트가 아닙니다. 여기서는 Resolver, Preparation, 권한·의미 검사를 묶어 부르는 교육용 표현으로 씁니다.

슬라이드가 생략한 세부 사항

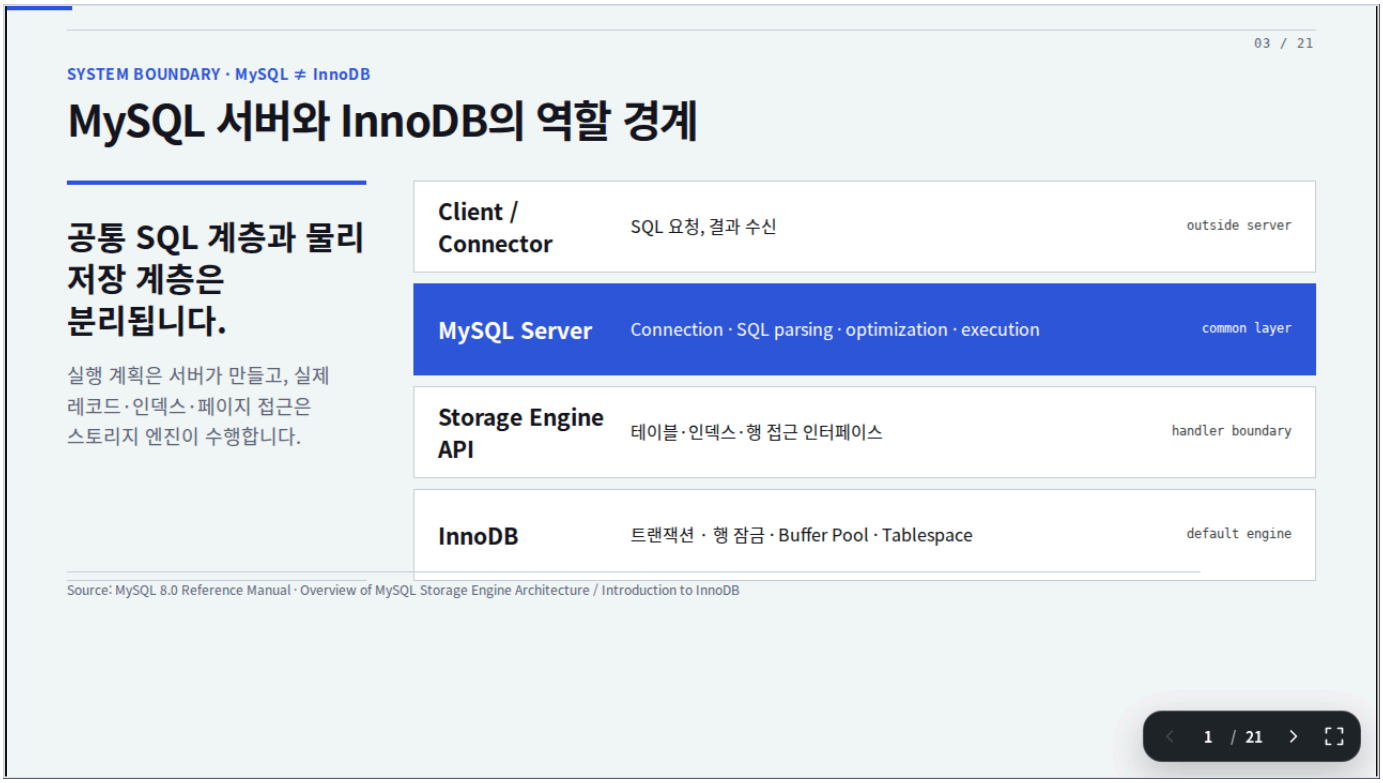
네 경계는 소스 코드의 디렉터리나 함수와 일대일로 맞춘 모듈도가 아닙니다. 설명을 위해 책임을 묶은 지도이며, 버전과 실행 문장 종류에 따라 세부 단계와 순서는 달라집니다.

확인 포인트

- 네 경계마다 책임 주체를 하나씩 적어 본다.

- Preprocessor가 교육용 묶음이라는 점을 기억한다.

3. MySQL 서버와 InnoDB의 역할 경계



핵심 개념

MySQL 서버의 공통 계층은 클라이언트 연결, 인증과 권한, SQL 문법 처리, 이름 해석, 실행 계획 최적화, 실행 흐름을 담당합니다. InnoDB는 이 공통 계층 아래에서 B-tree 인덱스, 행 레코드, Buffer Pool, Tablespace, 트랜잭션과 복구 구조를 담당합니다. MySQL 서버가 계획을 만든 뒤 실제 행 접근을 InnoDB에 위임하는 구조입니다.

흐름과 의미

이 분리는 진단에 직접 도움이 됩니다. `EXPLAIN`에서 조인 순서나 접근 방식이 이상하면 SQL·Optimizer 계층을 먼저 봅니다. 계획은 합리적인데 디스크 읽기가 지나치게 많다면 Buffer Pool hit/miss, 인덱스 선택도, 페이지 수를 확인합니다. MySQL Storage Engine Architecture는 서버가 공통 API로 여러 엔진을 감싸는 구조를 설명합니다.

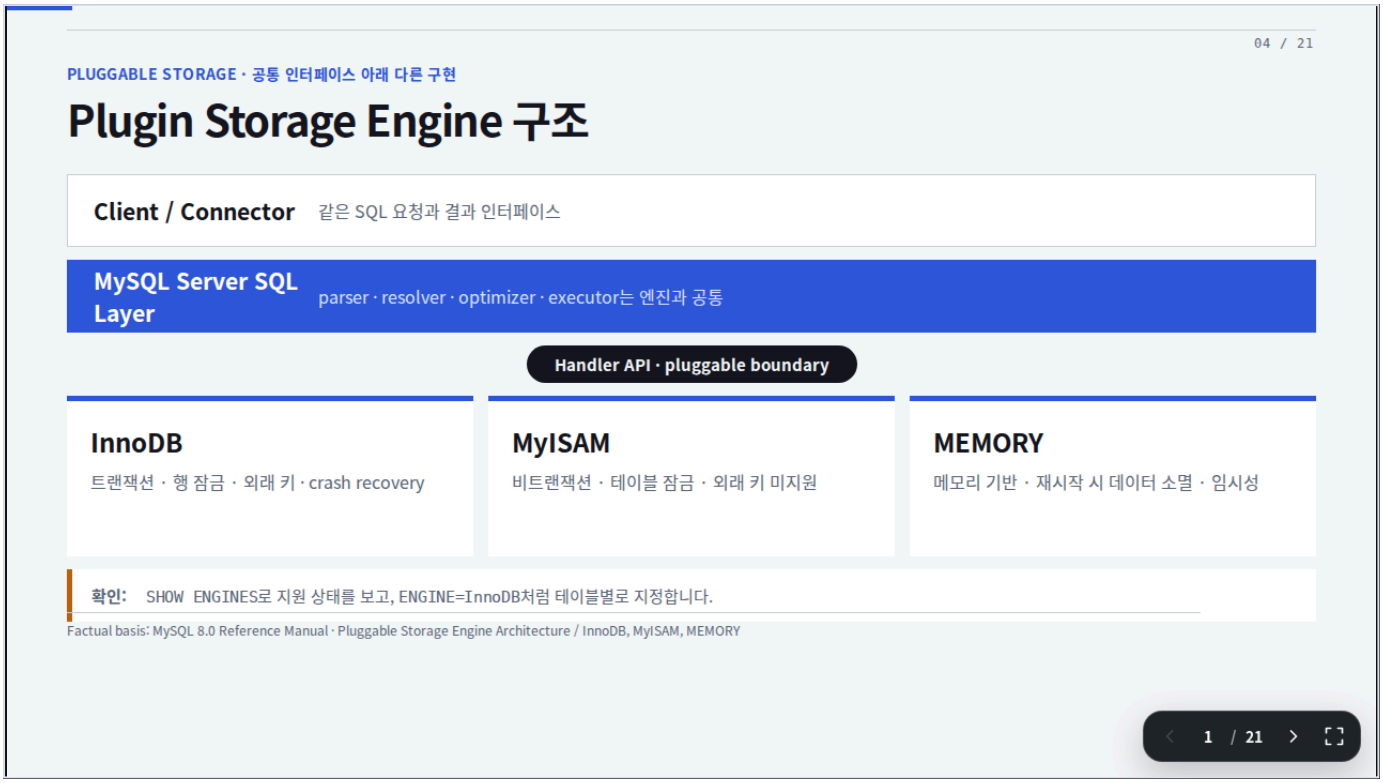
슬라이드가 생략한 세부 사항

Optimizer는 계획을 만들고 InnoDB는 행을 읽는다는 구분은 유용하지만, 두 영역은 통계와 비용 정보, Handler API, 잠금 요청을 주고받습니다. 완전히 고립된 두 프로세스로 이해하면 안 됩니다.

확인 포인트

- 실행 계획은 서버, 행 접근은 엔진이라는 경계를 확인한다.
- Handler API가 두 책임을 어떻게 잇는지 설명한다.

4. Plugin Storage Engine 구조



핵심 개념

MySQL은 테이블마다 스토리지 엔진을 정할 수 있습니다. 서버 위쪽의 Connector와 SQL 계층은 공통이지만, 아래쪽에는 InnoDB, MyISAM, MEMORY처럼 서로 다른 엔진 구현이 놓입니다. 로드 가능한 엔진 플러그인은 `INSTALL PLUGIN`과 `UNINSTALL PLUGIN`으로 관리합니다. 실제 지원 상태는 `SHOW ENGINES`로 확인합니다.

흐름과 의미

플러그형 구조가 엔진 차이까지 없애 주지는 않습니다. InnoDB는 트랜잭션, 행 수준 잠금, MVCC, 외래 키를 제공하지만 다른 엔진의 기능은 다를 수 있습니다. 존재하지 않거나 쓸 수 없는 엔진을 지정했을 때 의도치 않은 대체를 막으려면 `NO_ENGINE_SUBSTITUTION` SQL mode도 확인합니다. 자세한 내용은 Pluggable Storage Engine Architecture에 있습니다.

슬라이드가 생략한 세부 사항

InnoDB는 트랜잭션과 행 수준 잠금, crash recovery, 외래 키를 지원합니다. MyISAM은 트랜잭션을 지원하지 않고 테이블 수준 잠금을 사용합니다. MEMORY는 데이터를 메모리에 두므로 서버 재시작 뒤 데이터가 남는 영속 테이블 용도로 선택하면 안 됩니다. 엔진 비교는 기능표 하나보다 트랜잭션·잠금 단위·영속성·인덱스 제약을 함께 읽어야 합니다.

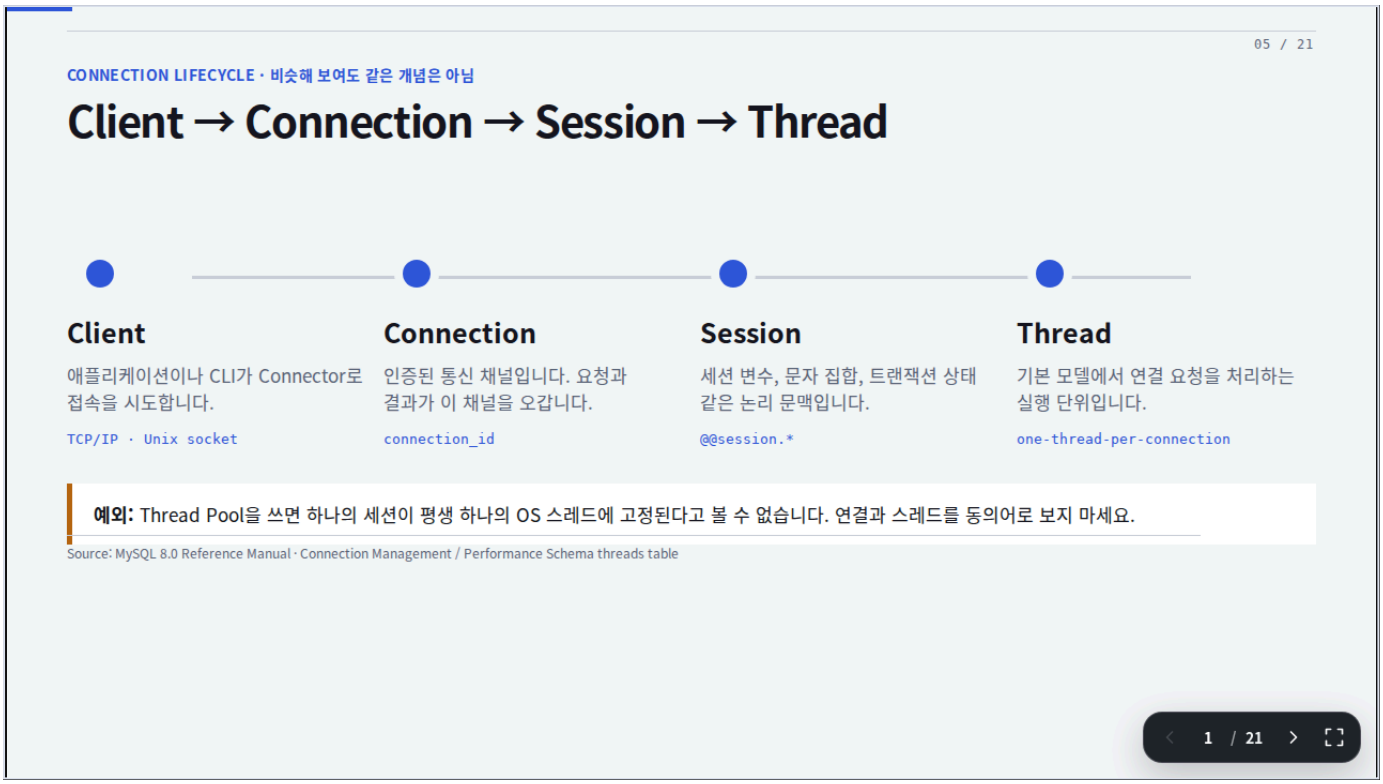
공식 문서 도식은 사실 확인에 유용하지만 Oracle 문서의 공개·상업적 재배포 권리가 자동으로 허용되는 것은 아닙니다. 이 자료는 로컬 학습용으로 원형을 유지해 사용했습니다. 외부 배포 전에는 별도 권리 검토가 필요합니다.

엔진을 바꾸면 SQL 표면이 같아 보여도 트랜잭션, 잠금 단위, 외래 키, crash recovery, 영속성 조건이 바뀝니다. 특히 MEMORY의 데이터는 서버 재시작 뒤 유지되지 않습니다.

확인 포인트

- `SHOW ENGINES`로 실제 지원 상태를 확인한다.
- InnoDB·MyISAM·MEMORY의 트랜잭션·잠금·영속성을 비교한다.

5. Client → Connection → Session → Thread



핵심 개념

Client는 MySQL에 접속하는 애플리케이션이나 CLI입니다. **Connection**은 인증된 통신 채널이고, **Session**은 연결에 묶여 유지되는 논리 상태입니다. 세션 변수, 문자 집합, 현재 데이터베이스, 트랜잭션 상태가 여기에 포함됩니다. **Thread**는 요청을 처리하는 실행 단위입니다.

흐름과 의미

기본 `one-thread-per-connection` 모델에서는 연결마다 전용 connection thread가 요청을 처리합니다. 연결이 끝난 뒤 스레드는 `thread_cache_size` 설정에 따라 캐시에 돌아가 다음 연결에서 재사용될 수 있습니다. 스레드 재사용은 이전 세션이 살아 있다는 뜻이 아닙니다. Session 상태는 새 연결에 맞게 초기화됩니다.

슬라이드가 생략한 세부 사항

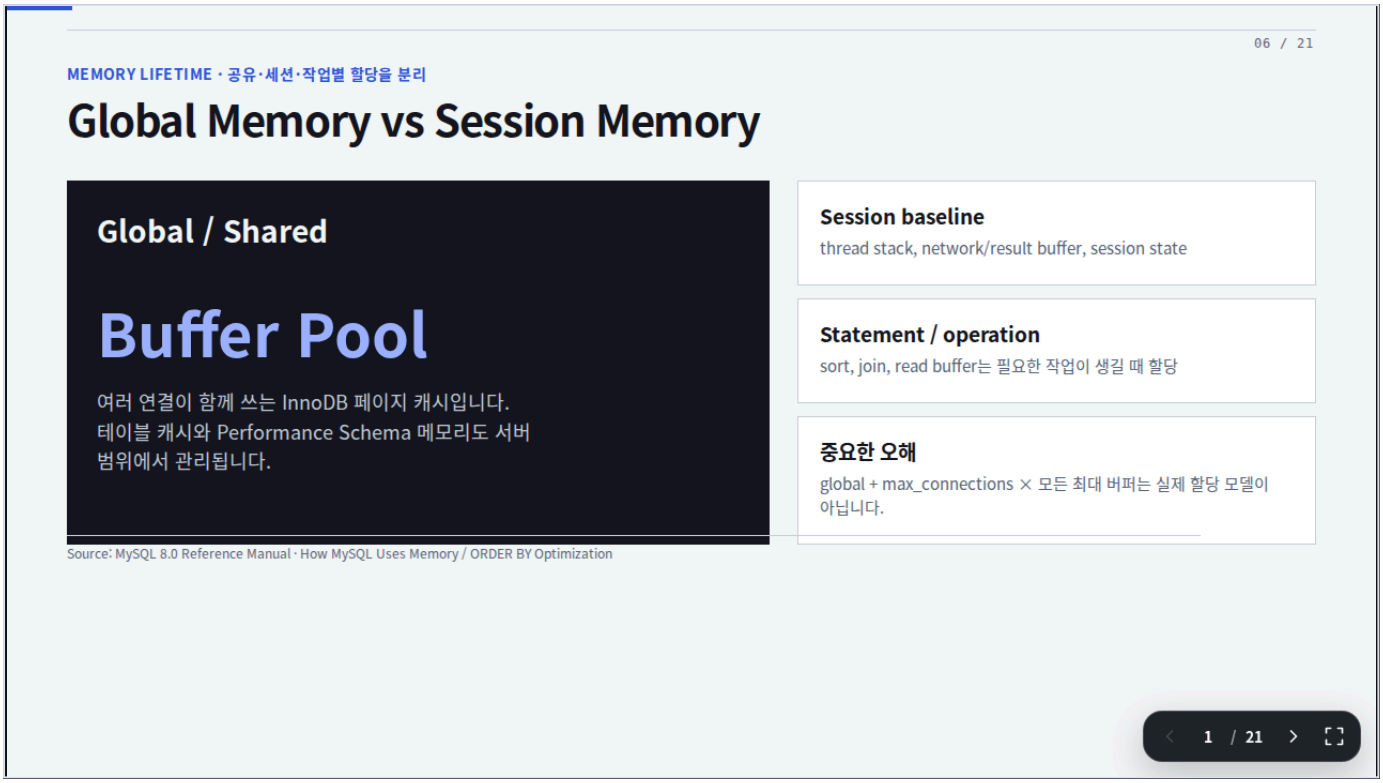
MySQL Enterprise Thread Pool을 쓰면 연결과 OS 스레드의 관계가 달라집니다. 여러 연결을 thread group과 실행 스레드가 나눠 처리하므로 “Connection 하나는 항상 OS thread 하나”라고 일반화하면 안 됩니다. 이 차이는 Connection Management와 Performance Schema의 `threads` 테이블에서 관찰할 수 있습니다.

애플리케이션의 connection pool에 있는 논리 요청 수명은 MySQL 서버 Connection 수명과 다릅니다. Thread Cache도 Session을 보존하는 기능이 아니라 종료된 connection thread의 생성 비용을 줄이는 장치입니다.

확인 포인트

- Connection과 Session의 종료 조건을 구분한다.
- Thread Pool 사용 시 일대일 관계 가정을 버린다.

6. Global Memory vs Session Memory



핵심 개념

메모리는 수명과 공유 범위로 나눠야 합니다. InnoDB Buffer Pool은 대표적인 글로벌 공유 메모리입니다. 서버의 여러 연결이 같은 데이터·인덱스 페이지 캐시를 사용합니다. 테이블 캐시와 테이블 정의 캐시도 서버 범위에서 공유됩니다.

흐름과 의미

반면 connection thread에는 thread stack, network/result buffer, 현재 SQL 문자열과 세션 상태가 필요합니다. Sort, join, read buffer는 해당 작업이 생길 때 문장이나 연산 단위로 할당됩니다. MySQL 8.0.12부터 filesort 메모리는 `sort_buffer_size` 전체를 항상 먼저 잡지 않고 필요한 만큼 증가시켜 상한까지 사용합니다.

슬라이드가 생략한 세부 사항

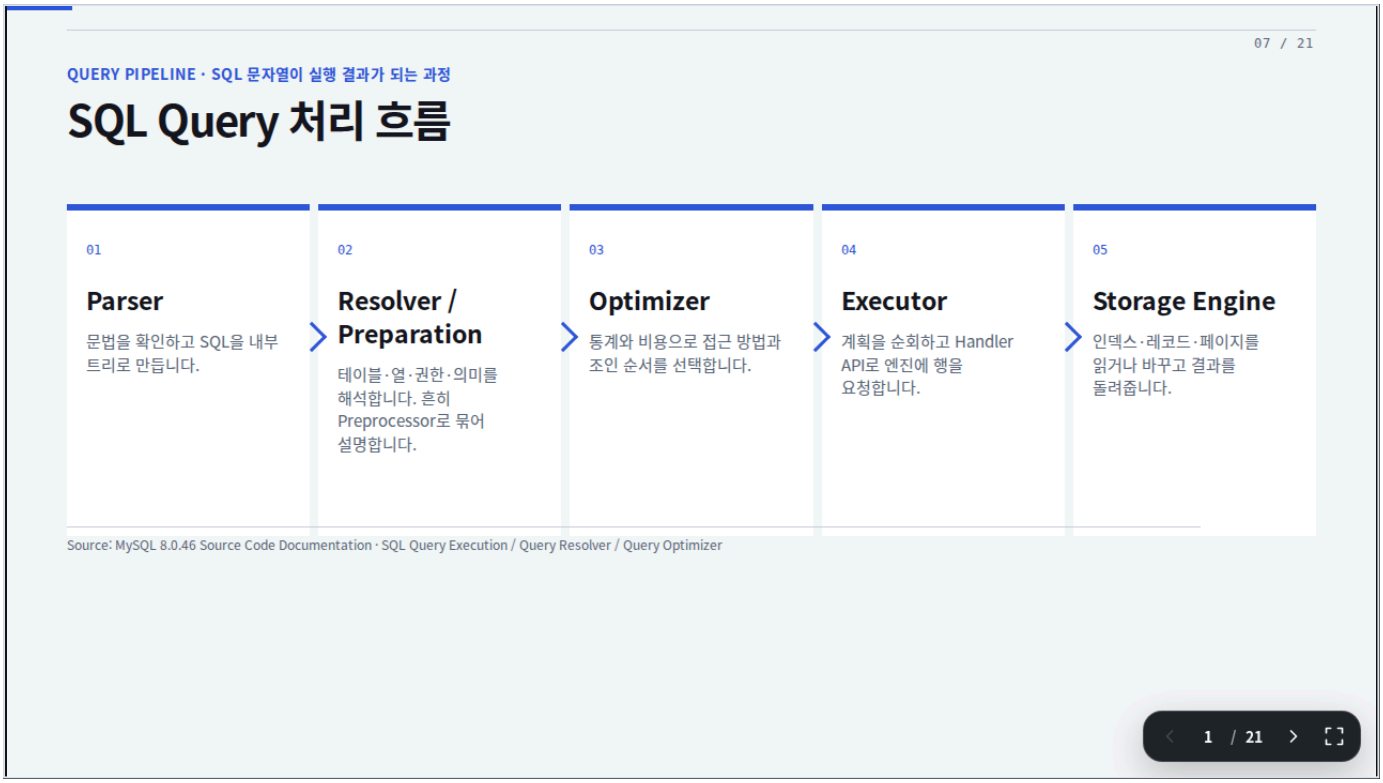
메모리를 `global memory + max_connections × 모든 session buffer 최대값`으로만 계산하면 실제 사용과 어긋날 수 있습니다. 일부 버퍼는 쓰지 않으면 할당되지 않습니다. 반대로 복잡한 문장은 여러 join buffer를 동시에 요구하기도 합니다. How MySQL Uses Memory는 공유 캐시와 thread-local 메모리 풀을 구분해 설명합니다.

세션 버퍼는 모두 연결 시점에 최대치로 선할당되지 않습니다. 필요한 연산이 생길 때 커지는 버퍼가 있고, 한 문장이 여러 join buffer를 동시에 만들기도 합니다. 최댓값의 단순 곱은 상한 추정일 뿐 실제 사용량이 아닙니다.

확인 포인트

- 공유 메모리와 작업별 버퍼를 따로 계산한다.
- 최대 연결 수에 모든 버퍼 최댓값을 곱한 수치를 실제 사용량으로 단정하지 않는다.

7. SQL Query 처리 흐름



핵심 개념

Connection thread가 SQL 요청을 받으면 Parser가 문자열을 토큰화하고 문법을 검사해 내부 트리를 만듭니다. 이어지는 Resolver/Preparation 단계는 테이블, 열, 별칭, 뷰, 서브쿼리 참조를 해석하고 권한과 의미 조건을 확인합니다. 필요한 prelocking과 table locking 뒤 Optimizer가 접근 방법과 조인 순서, 변환을 선택합니다.

흐름과 의미

Executor는 선택된 계획을 실행합니다. 실행기는 계획의 iterator를 순회하면서 필요한 행 연산을 Handler API에 요청합니다. Storage Engine은 인덱스와 레코드, 페이지를 다루고 결과를 돌려줍니다. 문장 처리가 끝나면 cleanup 단계가 이어집니다.

슬라이드가 생략한 세부 사항

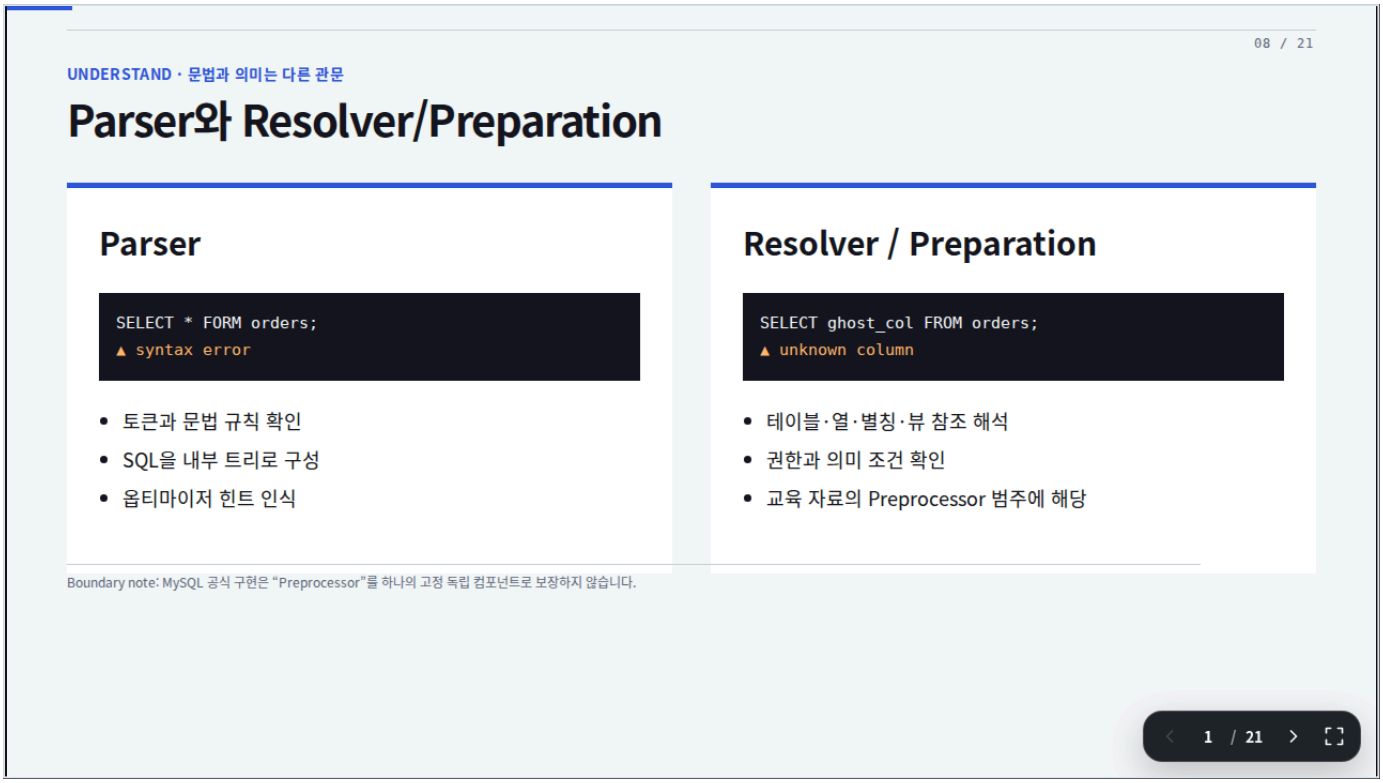
MySQL 8.0.46 Source Code Documentation은 parsing 이후 DML 단계를 `Prelocking → Preparation → Locking of tables → Optimization → Execution or explain → Cleanup`으로 설명합니다. 교육용 도식과 실제 구현 용어는 구분해서 읽어야 합니다. 근거는 SQL Query Execution에 정리돼 있습니다.

실제 DML 경로에는 prelocking, table locking, cleanup이 포함됩니다. Prepared Statement 재실행은 preparation 일부를 생략할 수 있으며, DDL이나 관리 문장은 DML과 같은 경로를 그대로 따르지 않습니다.

확인 포인트

- parse·resolve·optimize·execute 뒤 cleanup까지 이어 본다.
- Executor가 Handler API를 호출하는 지점을 표시한다.

8. Parser와 Resolver/Preparation



핵심 개념

Parser는 문장이 SQL 문법에 맞는지 확인합니다. `SELECT * FORM orders`처럼 `FROM`을 잘못 쓴 문장은 여기에서 멈춥니다. Resolver/Preparation은 문법이 맞는 문장 안의 이름과 의미를 확인합니다. 존재하지 않는 `ghost_col`을 참조하거나 권한이 없는 테이블에 접근하면 이 단계에서 오류가 납니다.

흐름과 의미

Resolver는 단순한 오탈자 검사기가 아닙니다. 별칭과 와일드카드, 뷰와 서브쿼리, 식의 참조를 실제 객체와 연결합니다. 이 준비가 끝나야 Optimizer가 후보 계획의 비용을 비교합니다. Prepared Statement를 다시 실행할 때는 일부 preparation을 생략할 수 있습니다. 모든 요청이 똑같은 비용으로 이 단계를 반복하지는 않습니다.

슬라이드가 생략한 세부 사항

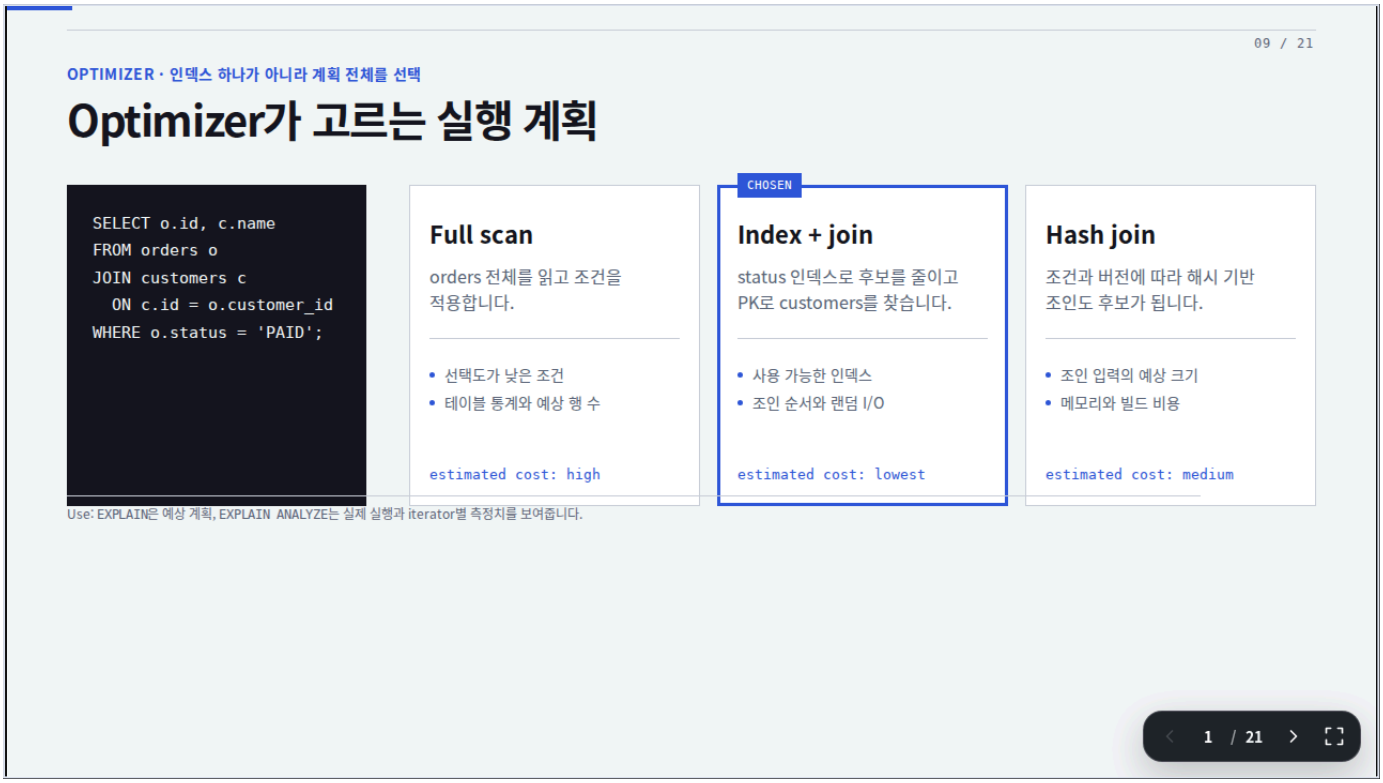
책에서 쓰는 Preprocessor는 Resolver/Preparation과 권한·의미 검사를 한데 묶은 교육용 이름입니다. MySQL 8.0 소스 문서가 고정된 단일 Preprocessor 컴포넌트를 보장하는 것은 아닙니다.

확인 포인트

- 문법 오류와 이름·권한 오류의 실패 단계를 나눈다.

- Prepared Statement 재실행에서 생략 가능한 preparation을 구분한다.

9. Optimizer가 고르는 실행 계획



핵심 개념

Optimizer는 “인덱스를 쓸지 말지”보다 넓은 결정을 합니다. 어떤 테이블을 먼저 읽을지, 어떤 인덱스와 접근 방식을 쓸지, 서브쿼리를 변환하거나 materialize할지, 조인을 어떤 연산으로 수행할지를 비교합니다. 각 후보의 통계와 비용을 평가해 실행 계획을 선택합니다.

흐름과 의미

`EXPLAIN`은 Optimizer가 예상한 계획을 보여줍니다. 여기서 예상 행 수와 접근 방식, 조인 순서를 확인합니다. `EXPLAIN ANALYZE`는 MySQL 8.0.18부터 문장을 실제로 실행하고 iterator별 예상치와 실제 시간·행 수를 함께 보여줍니다. 데이터 변경 문장이나 비용이 큰 쿼리에 쓸 때는 실제 실행에 따른 부작용을 고려합니다.

슬라이드가 생략한 세부 사항

통계가 오래됐거나 데이터 분포가 치우치면 비용 추정이 실제와 달라집니다. 인덱스 존재 여부에서 확인을 끝내지 말고 선택된 계획과 실제 실행을 비교합니다. MySQL 8.0에는 hash join도 있습니다. 모든 조인이 늘 nested loop라고 설명하면 부정확합니다.

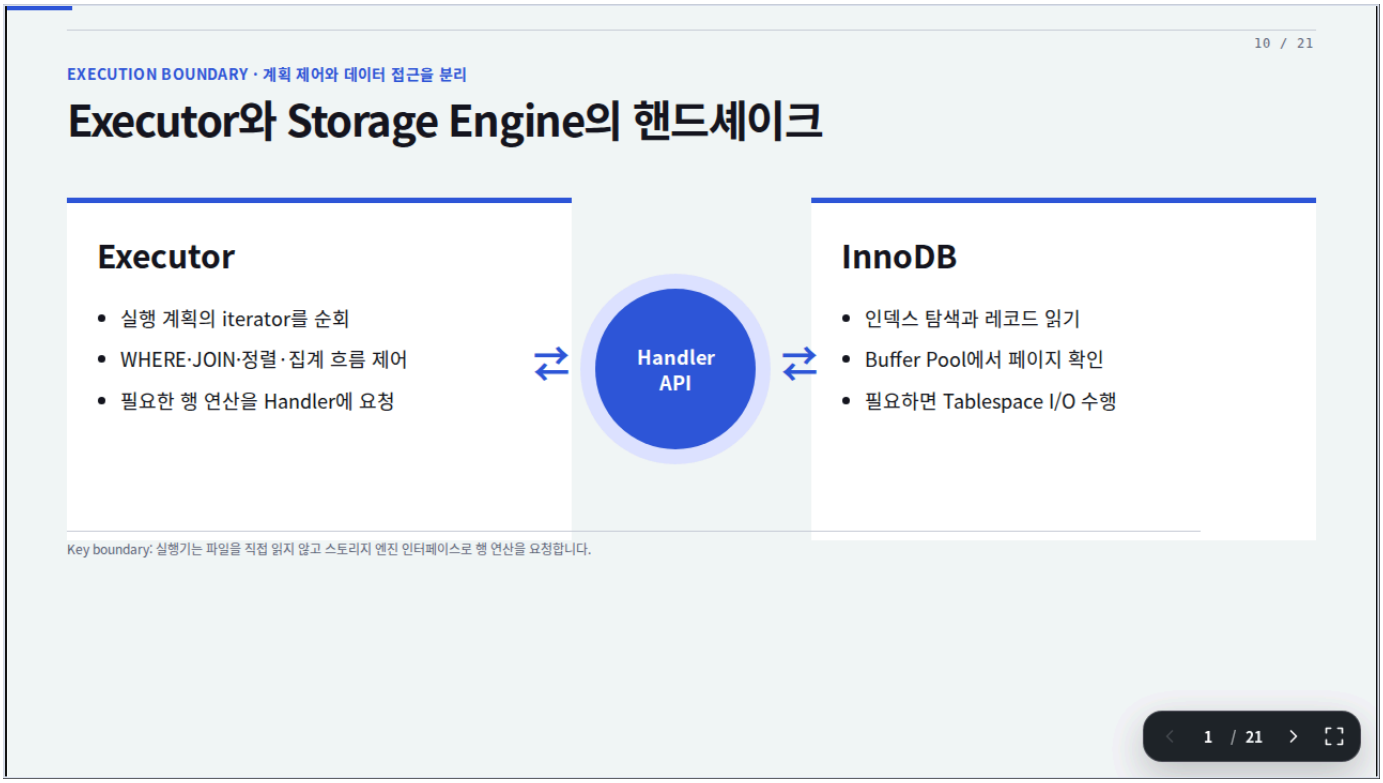
`EXPLAIN ANALYZE`는 관찰만 하는 명령이 아니라 문장을 실제로 실행합니다. 비용이 큰 쿼리와 데이터 변경 문장에는 실행 시간, 잠금, 변경 부작용이 생길 수 있으므로 검증 환경과 트랜잭션 경계를

정합니다.

확인 포인트

- `EXPLAIN`의 예상 행 수와 실제 행 수를 비교한다.
- `EXPLAIN ANALYZE`의 실제 실행 부작용을 확인한다.

10. Executor와 Storage Engine의 핸드셰이크



핵심 개념

Executor는 실행 계획을 실제 연산 순서로 수행합니다. 스캔, 필터, 조인, 정렬, 집계 iterator가 다음 행을 요청하면서 결과를 만들어 냅니다. Executor가 `.ibd` 파일을 직접 열지는 않습니다. Handler API로 Storage Engine에 행 연산을 요청합니다.

흐름과 의미

InnoDB는 요청받은 인덱스 접근이나 레코드 읽기를 수행합니다. 필요한 페이지가 Buffer Pool에 있는지 확인하고, 없으면 Tablespace에서 읽습니다. 레코드를 찾은 뒤 필요한 열을 Executor로 돌려보내면 상위 연산이 이어집니다. 반복이 끝난 결과 집합은 Connection을 거쳐 Client로 돌아갑니다.

슬라이드가 생략한 세부 사항

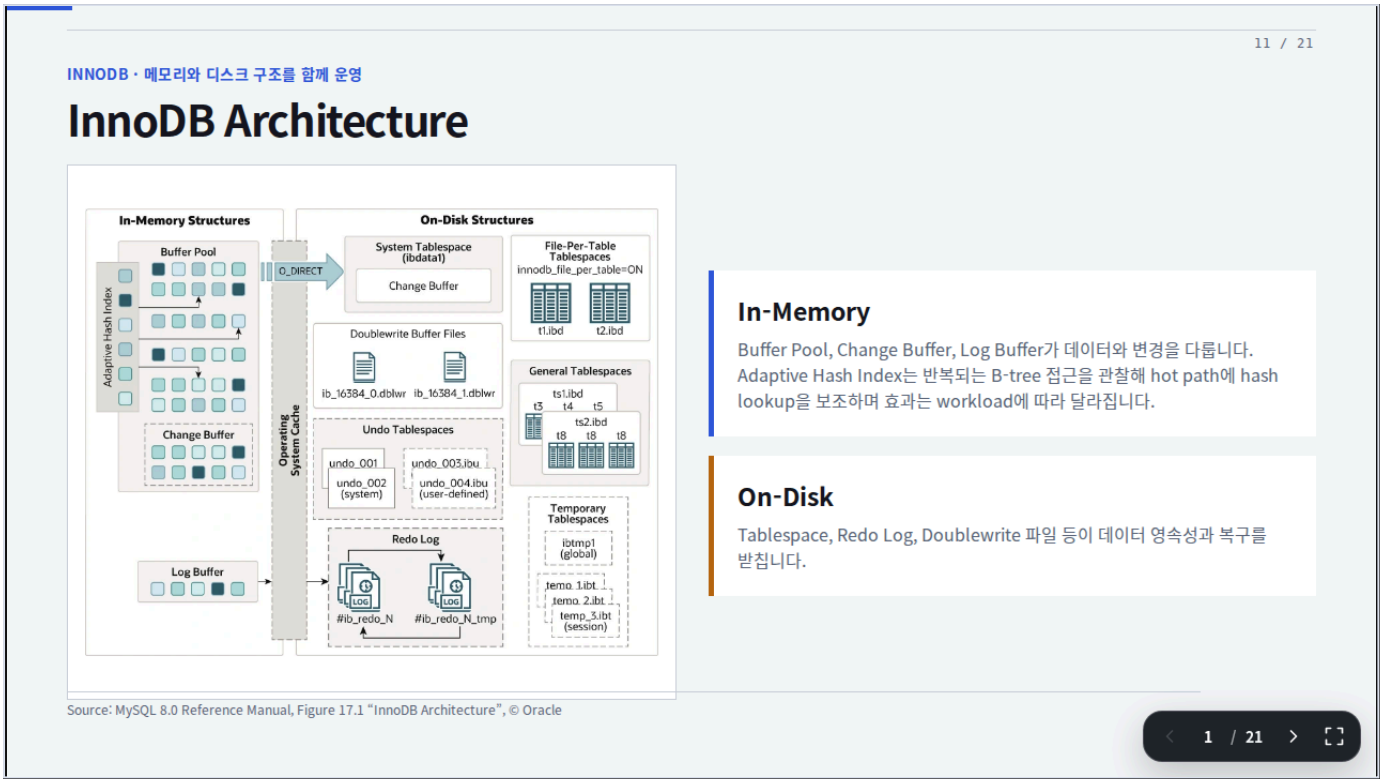
이 접점이 플러그형 구조의 핵심입니다. Executor는 공통 인터페이스를 사용하되 페이지·잠금·트랜잭션의 내부 구현은 엔진마다 다릅니다.

Handler API는 추상 경계입니다. 실제 호출은 index read, range scan, row fetch처럼 계획과 엔진 구현에 따라 달라집니다. Executor가 데이터 파일 형식을 직접 해석하지 않는다는 책임 분리가 핵심입니다.

확인 포인트

- Executor와 InnoDB가 각각 무엇을 모르는지 설명한다.
- range scan과 row fetch가 계획에 따라 달라짐을 기억한다.

11. InnoDB Architecture



핵심 개념

InnoDB의 구조는 메모리 영역과 디스크 영역으로 나눠 보면 이해하기 쉽습니다. 메모리 쪽의 핵심은 Buffer Pool입니다. Change Buffer, Adaptive Hash Index, Log Buffer도 여기에 속합니다. 디스크에는 Tablespace, Redo Log, Doublewrite Buffer 같은 구조가 놓입니다. 각 요소는 읽기 성능, 쓰기 지연, 장애 복구라는 서로 다른 문제를 맡습니다.

흐름과 의미

다만 이 구분을 "메모리는 휘발성, 디스크는 영구 저장"이라는 한 문장으로 끝내면 부족합니다. Buffer Pool의 dirty page는 아직 디스크에 반영되지 않은 변경을 담고 있고, Redo Log는 그 간극에서 복구 가능성을 보장합니다. Change Buffer는 보조 인덱스 변경을 나중에 병합할 수 있도록 돕습니다. Adaptive Hash Index는 자주 접근하는 B-tree 경로를 해시 방식으로 빠르게 찾도록 돕지만 워크로드에 따라 효과가 달라집니다.

슬라이드가 생략한 세부 사항

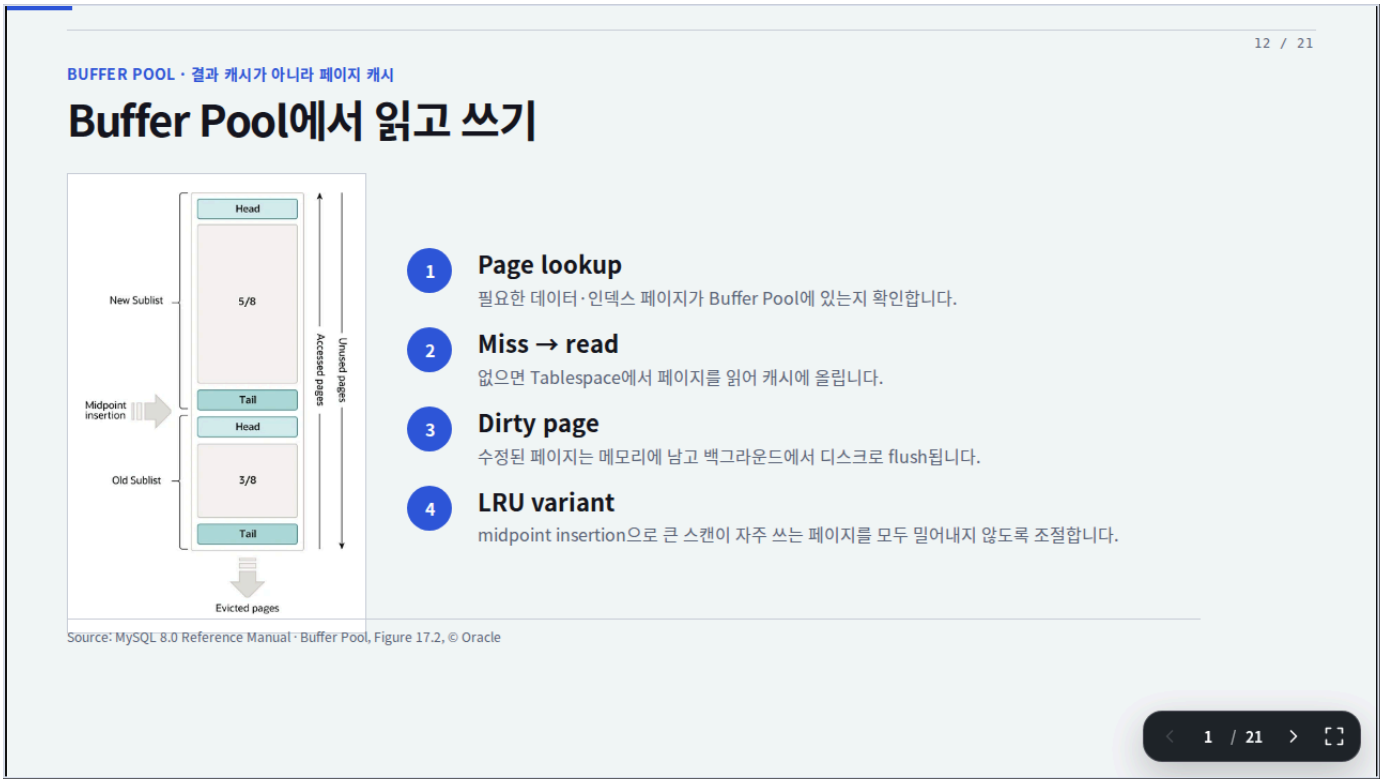
발표에서는 SQL 한 문장의 읽기 경로와 직접 맞닿은 Buffer Pool, Tablespace, Page에 초점을 맞춥니다. 트랜잭션 격리, MVCC, Undo·Redo 복구도 같은 아키텍처에 속하지만 별도 발표가 필요할 만큼 범위가 큽니다. 전체 구성은 InnoDB Architecture에서 확인하십시오.

Adaptive Hash Index는 전체 B-tree를 별도 해시 인덱스로 복제하지 않습니다. 반복되는 검색 패턴을 관찰해 hot page의 일부 경로를 보조합니다. 경합이 큰 워크로드에서는 이득이 줄거나 오히려 비용이 생길 수 있습니다.

확인 포인트

- 메모리 구조와 디스크 구조의 역할을 나눈다.
- Adaptive Hash Index의 효과가 workload 의존적임을 확인한다.

12. Buffer Pool에서 읽고 쓰기



핵심 개념

Buffer Pool은 InnoDB가 디스크의 테이블·인덱스 페이지를 메모리에 캐시하는 공간입니다. 읽으려는 페이지가 Buffer Pool에 있으면 메모리에서 처리합니다. 없으면 해당 페이지를 Tablespace에서 읽어 빈 프레임에 올립니다. 변경된 페이지는 dirty page가 되며, 체크포인트와 백그라운드 플러시 과정에서 디스크에 기록됩니다.

흐름과 의미

페이지 교체에는 LRU에 가까운 목록을 쓰지만 단순한 한 줄짜리 LRU는 아닙니다. 목록을 young 영역과 old 영역으로 나누고 새 페이지를 midpoint 부근에 넣습니다. 한 번만 훑는 큰 스캔이 오래 쓰이던 hot page를 한꺼번에 밀어내는 현상을 줄이려는 설계입니다. `innodb_old_blocks_pct`와 `innodb_old_blocks_time`은 이 동작을 조절합니다.

슬라이드가 생략한 세부 사항

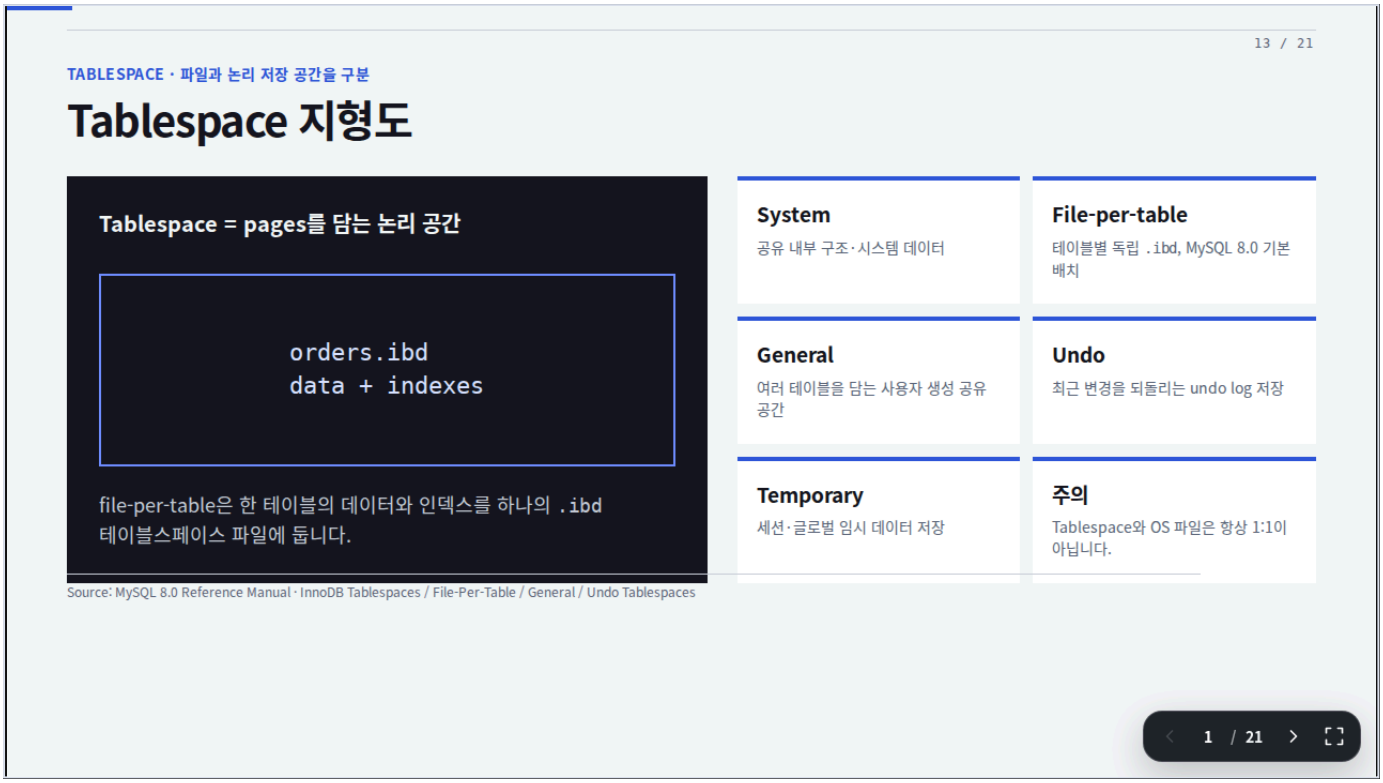
Buffer Pool은 SQL 결과 집합을 저장하는 result cache가 아닙니다. 행과 인덱스가 들어 있는 InnoDB 페이지를 캐시합니다. Buffer Pool hit가 높더라도 쿼리가 너무 많은 페이지를 훑으면 CPU와 메모리 대역폭을 많이 쓸 수 있습니다. 반대로 miss가 많다면 디스크 I/O가 병목이 될 가능성이 커집니다. InnoDB Buffer Pool을 참고하십시오.

Buffer Pool은 쿼리 결과 캐시가 아닙니다. Page를 캐시하며 dirty page는 checkpoint와 background flush 흐름에서 디스크로 기록됩니다. hit rate가 높아도 너무 많은 Page를 훑으면 CPU와 메모리 대역폭이 병목이 됩니다.

확인 포인트

- cache hit과 cache miss의 I/O 경로를 그린다.
- dirty page와 result cache를 혼동하지 않는다.

13. Tablespace 지형도



핵심 개념

Tablespace는 InnoDB의 페이지와 세그먼트가 놓이는 논리적 저장 공간입니다. 하나의 Tablespace가 반드시 하나의 파일과 일치하는 것은 아닙니다. System Tablespace처럼 여러 파일로 구성할 수 있는 공간도 있고, file-per-table Tablespace처럼 일반적으로 테이블 하나가 자체 `.ibd` 파일을 갖는 경우도 있습니다.

흐름과 의미

MySQL 8.0에서 자주 만나는 공간은 System Tablespace, File-per-table Tablespace, General Tablespace, Undo Tablespace, Temporary Tablespace입니다. System Tablespace에는 change buffer와 일부 시스템 데이터가 놓입니다. MySQL 8.0의 데이터 딕셔너리는 InnoDB에 통합됐지만 이를 단순히 “모든 메타데이터가 system tablespace 한 파일에 있다”고 설명하면 정확하지 않습니다.

슬라이드가 생략한 세부 사항

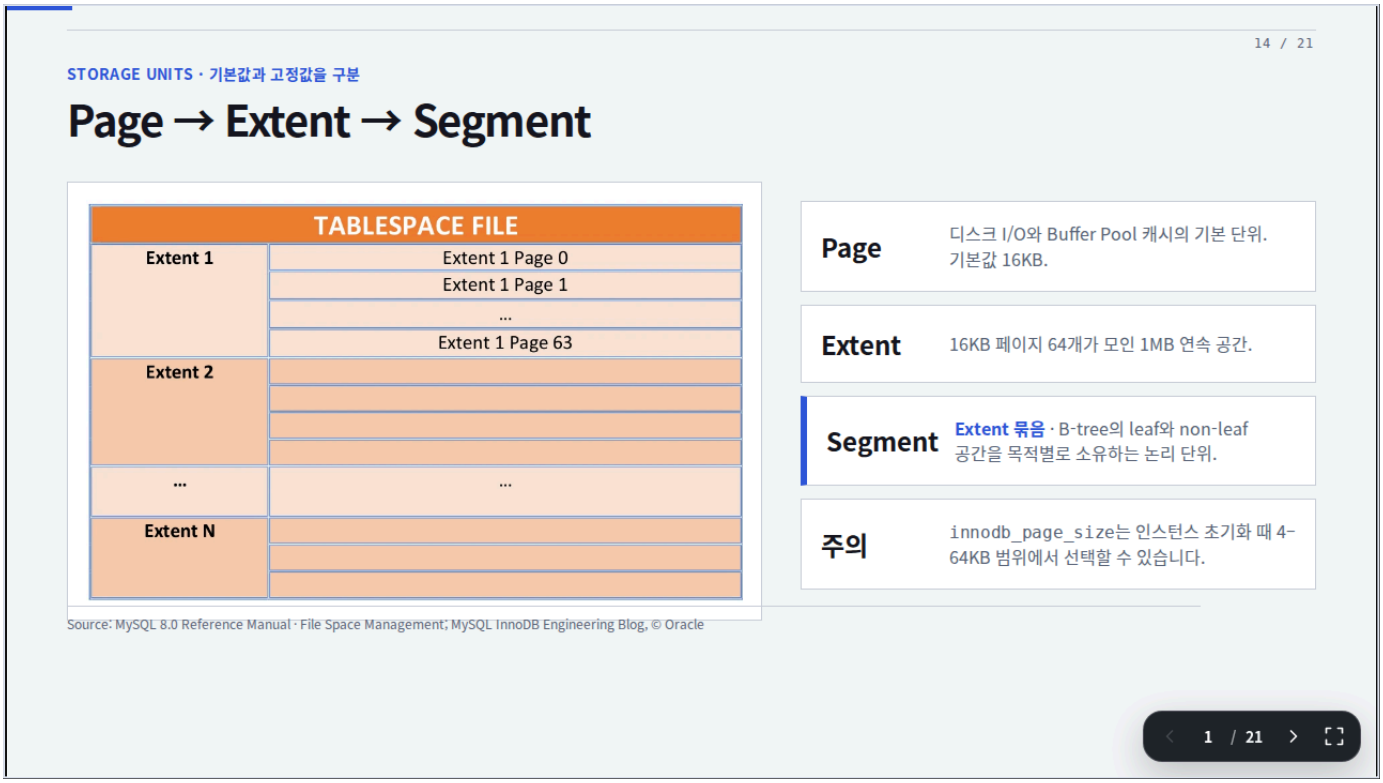
File-per-table은 테이블별 이동·회수·관리에 유리합니다. General Tablespace는 여러 테이블을 하나의 공유 공간에 둡니다. Undo Tablespace는 MVCC와 rollback에 필요한 undo record를 담고, Temporary Tablespace는 내부·사용자 임시 테이블을 저장합니다. 구조와 제약은 InnoDB Tablespaces와 File Space Management에서 확인하십시오.

Tablespace와 파일은 항상 일대일이 아닙니다. System Tablespace는 여러 data file로 구성할 수 있고 General Tablespace에는 여러 테이블이 들어갑니다. MySQL 8.0의 통합 Data Dictionary를 구버전 system tablespace 설명과 섞지 않습니다.

확인 포인트

- Tablespace와 data file의 대응 관계를 유형별로 비교한다.
- Undo·Temporary Tablespace의 목적을 구분한다.

14. Page → Extent → Segment



핵심 개념

Page는 InnoDB가 디스크와 Buffer Pool 사이에서 읽고 쓰는 기본 단위입니다. 기본값은 16KB입니다. 인스턴스를 초기화할 때 `innodb_page_size`로 4KB, 8KB, 16KB, 32KB, 64KB 가운데 하나를 고릅니다. 운영 중인 인스턴스에서 가볍게 바꾸는 설정은 아닙니다.

흐름과 의미

기본 16KB 페이지 기준으로 연속된 64개 페이지가 1MB Extent를 이룹니다. 4KB·8KB·16KB 페이지 크기에서도 Extent는 1MB이고, 32KB 페이지에서는 2MB, 64KB 페이지에서는 4MB입니다. Segment는 한 인덱스의 leaf page나 non-leaf page처럼 같은 목적의 페이지·Extent를 묶어 관리하는 논리 단위입니다.

슬라이드가 생략한 세부 사항

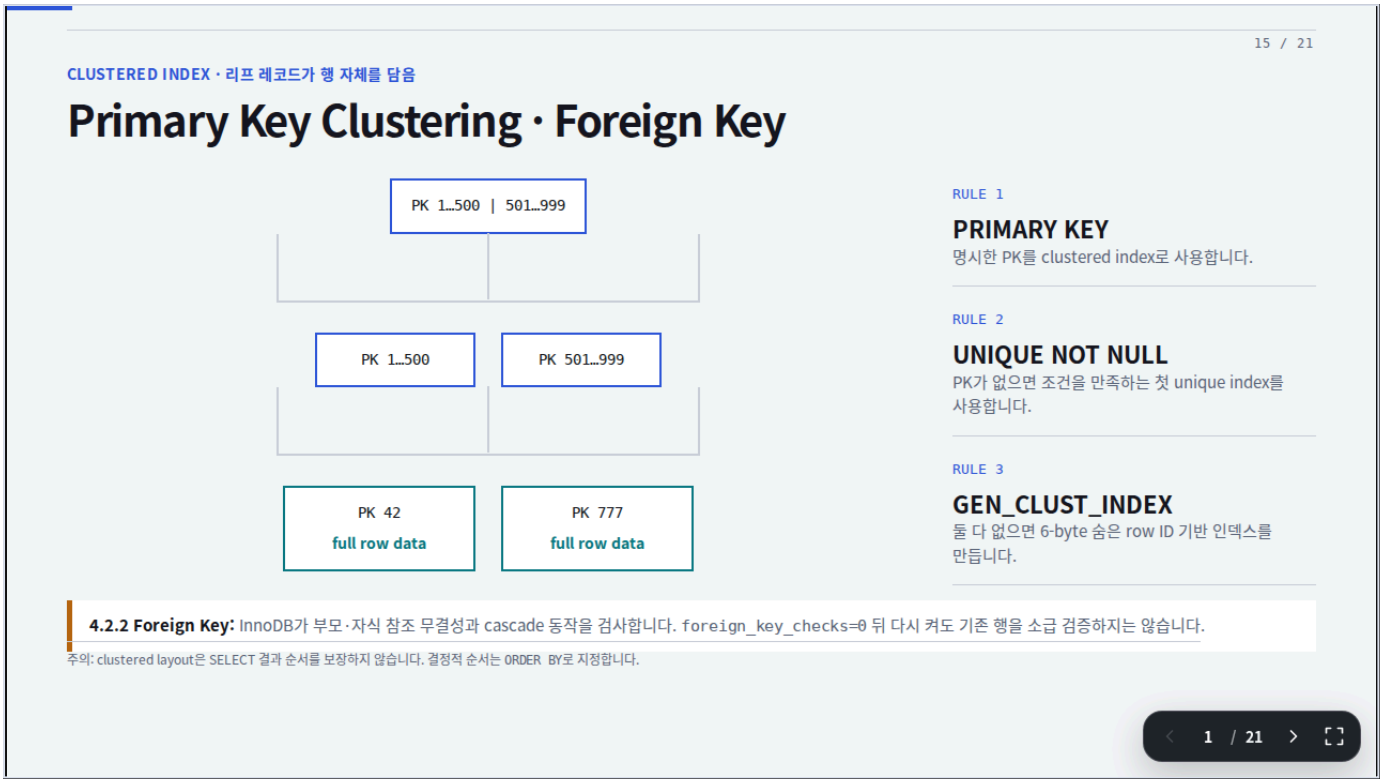
한 Page가 곧 한 Row는 아닙니다. 일반적으로 하나의 index page 안에 여러 record가 들어갑니다. 큰 가변 길이 열은 일부 데이터를 overflow page에 둘 수 있습니다. Page Type도 index, undo log, inode, system 등 여러 종류가 있으므로 “페이지는 테이블 행을 담는 상자” 정도로만 기억하면 실제 구조를 놓치게 됩니다.

16KB는 기본 Page 크기이지 고정값이 아닙니다. 4KB·8KB·16KB Page의 Extent는 1MB지만 32KB에서는 2MB, 64KB에서는 4MB입니다. Page 크기는 인스턴스 초기화 뒤 바꿀 수 없습니다.

확인 포인트

- Page 크기별 Extent 크기를 다시 계산한다.
- Segment가 leaf와 non-leaf 공간을 논리적으로 묶는 이유를 설명한다.

15. Primary Key Clustering · Foreign Key



핵심 개념

InnoDB의 clustered index leaf page에는 행 데이터가 함께 저장됩니다. 테이블에 명시적인 Primary Key가 있으면 그 키를 clustered index로 씁니다. Primary Key가 없으면 모든 열이 `NOT NULL`인 첫 번째 `UNIQUE` 인덱스를 선택합니다. 그런 인덱스도 없으면 InnoDB가 6바이트 row ID를 만들고 `GEN_CLUST_INDEX`라는 숨은 clustered index를 구성합니다.

흐름과 의미

Primary Key로 찾은 leaf record에서 행의 나머지 열을 바로 읽는 구조입니다. 반면 Primary Key가 길면 모든 secondary index leaf record에 복사되는 값도 길어집니다. 짧고 안정적이며 증가 방향이 비교적 일정한 키가 공간과 쓰기 패턴에 유리한 경우가 많은 이유입니다. 그렇다고 모든 서비스가 단조 증가 정수만 써야 하는 것은 아닙니다. 분산 ID, 보안 요구, 업무 키의 성격을 함께 봅니다.

슬라이드가 생략한 세부 사항

InnoDB의 외래 키는 부모·자식 테이블의 참조 무결성을 검사하고 `CASCADE`, `SET NULL`, `RESTRICT` 같은 참조 동작을 지원합니다. 관련 열에는 인덱스가 필요합니다. `foreign_key_checks=0`은 적재 순서를 유연하게 만들 때 쓰기도 하지만 다시 `1`로 바꿔도 이미 들어간 행을 소급 검사하지 않습니다. 비활성화 구간의 데이터 품질을 별도로 검증해야 합니다.

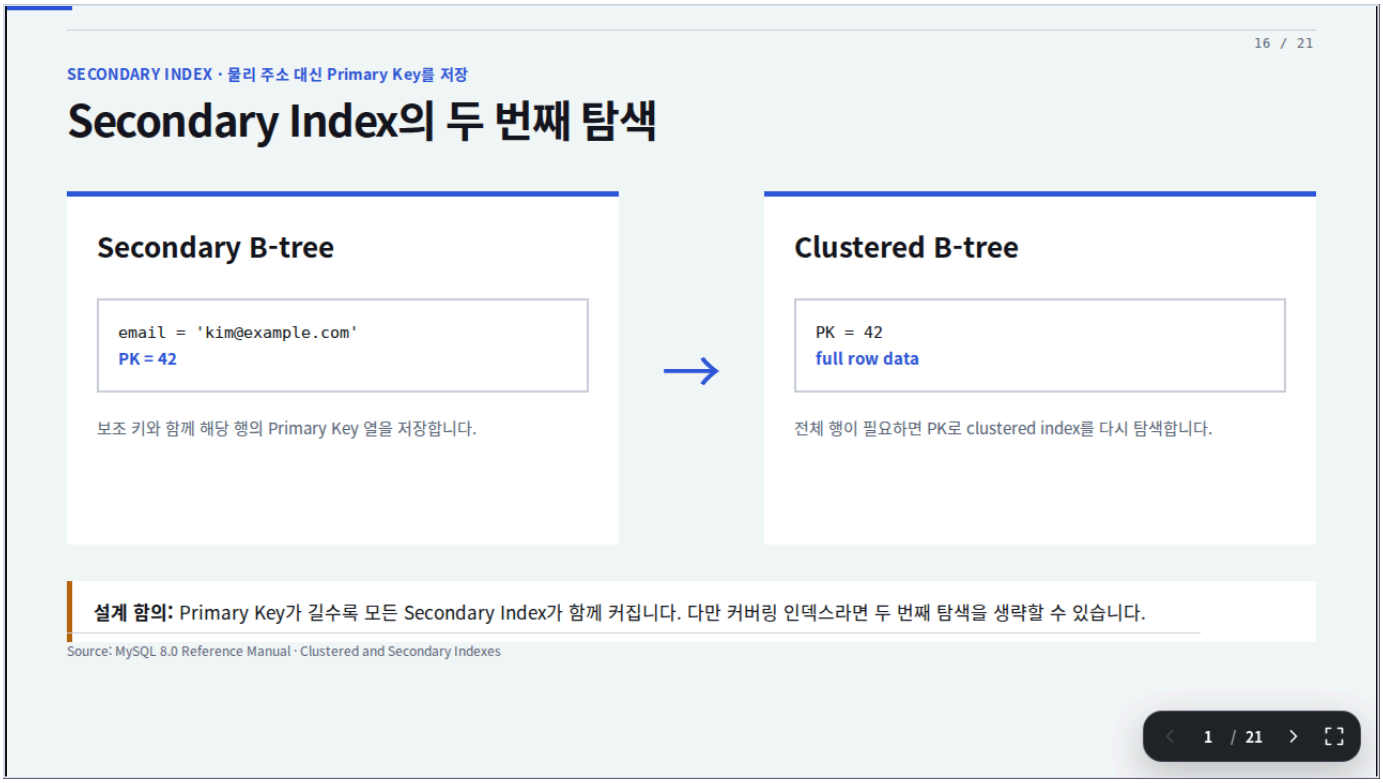
Clustered index의 물리적 배치가 조회 결과의 정렬 순서를 보장하지는 않습니다. SQL 결과 순서가 필요하면 반드시 `ORDER BY`를 사용합니다. 자세한 선택 규칙은 Clustered and Secondary Indexes에 정리돼 있습니다.

Clustered layout은 물리 배치 특성일 뿐 SQL 결과 순서를 보장하지 않습니다. 외래 키 검사를 잠시 껐다가 다시 켜도 비활성화 구간에 들어온 기존 행은 소급 검증되지 않습니다.

확인 포인트

- PK 선택 규칙 세 단계를 순서대로 말한다.
- 외래 키 비활성화 뒤 기존 행이 소급 검증되지 않음을 기억한다.

16. Secondary Index의 두 번째 탐색



핵심 개념

InnoDB의 secondary index leaf record에는 secondary key와 해당 행의 Primary Key가 저장됩니다. secondary index로 조건에 맞는 항목을 찾은 뒤 행의 다른 열이 필요하면 그 Primary Key로 clustered index를 다시 탐색합니다. 흔히 이를 두 번째 탐색 또는 back-to-table lookup이라고 설명합니다.

흐름과 의미

예를 들어 `orders(status, created_at)` 인덱스로 주문 후보를 찾았는데 결과에 `total_amount`가 필요하고 그 열이 인덱스에 없다면 clustered index에서 행을 다시 읽습니다. 후보가 많을수록 두 번째 탐색 비용도 커집니다. 쿼리에 필요한 열이 모두 secondary index에 있으면 covering index가 되어 clustered index 접근을 피합니다.

슬라이드가 생략한 세부 사항

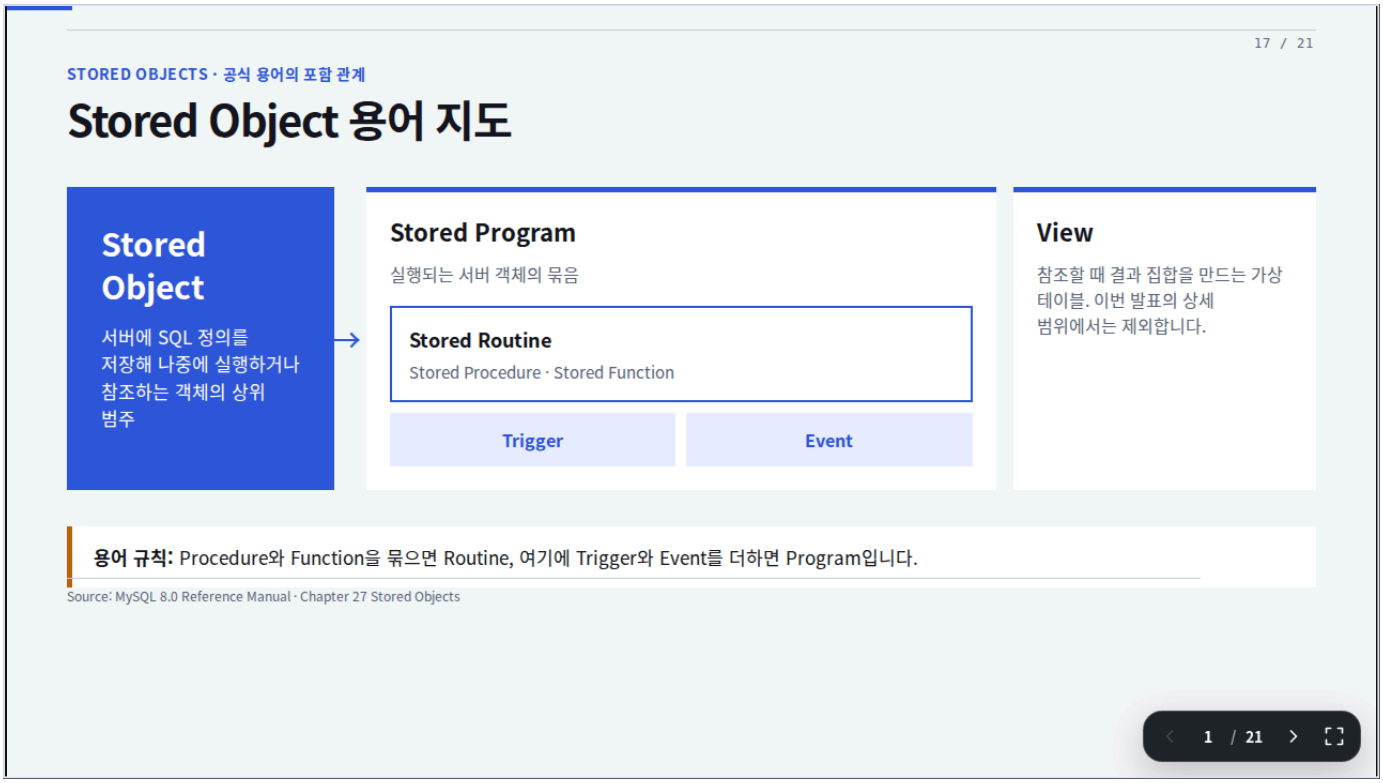
Primary Key가 모든 secondary index에 포함된다는 점은 설계 비용으로 돌아옵니다. Primary Key가 길수록 secondary index가 커지고, 같은 Buffer Pool 공간에 담기는 entry 수가 줄어듭니다. 인덱스 수가 많을수록 쓰기 비용도 늘어납니다. “인덱스가 있으면 빠르다”보다 접근 경로와 저장 비용을 함께 봐야 합니다.

Secondary index가 필요한 열을 모두 담는 covering index라면 clustered index 재탐색을 피합니다. 다만 열을 무작정 더 넣으면 인덱스 크기와 쓰기 비용이 늘어납니다.

확인 포인트

- secondary leaf의 PK가 clustered lookup에 쓰임을 설명한다.
- covering index의 이득과 인덱스 비대화 비용을 함께 본다.

17. Stored Object 용어 지도



핵심 개념

MySQL의 stored object는 서버에 정의를 저장하고 서버 안에서 실행하는 객체를 가리킵니다. Stored Procedure와 Stored Function을 합쳐 Stored Routine이라고 부릅니다. Trigger와 Event는 실행 계기가 다르지만 함께 Stored Object 범주에서 설명합니다.

흐름과 의미

Procedure는 애플리케이션이 `CALL`로 명시적으로 호출하며 `IN`, `OUT`, `INOUT` 매개변수를 씁니다. Function은 값을 반환하고 SQL 식 안에서 호출합니다. Trigger는 특정 테이블의 `INSERT`, `UPDATE`, `DELETE` 전후에 행마다 실행됩니다. Event는 Event Scheduler가 지정된 시각이나 반복 주기에 맞춰 실행합니다.

슬라이드가 생략한 세부 사항

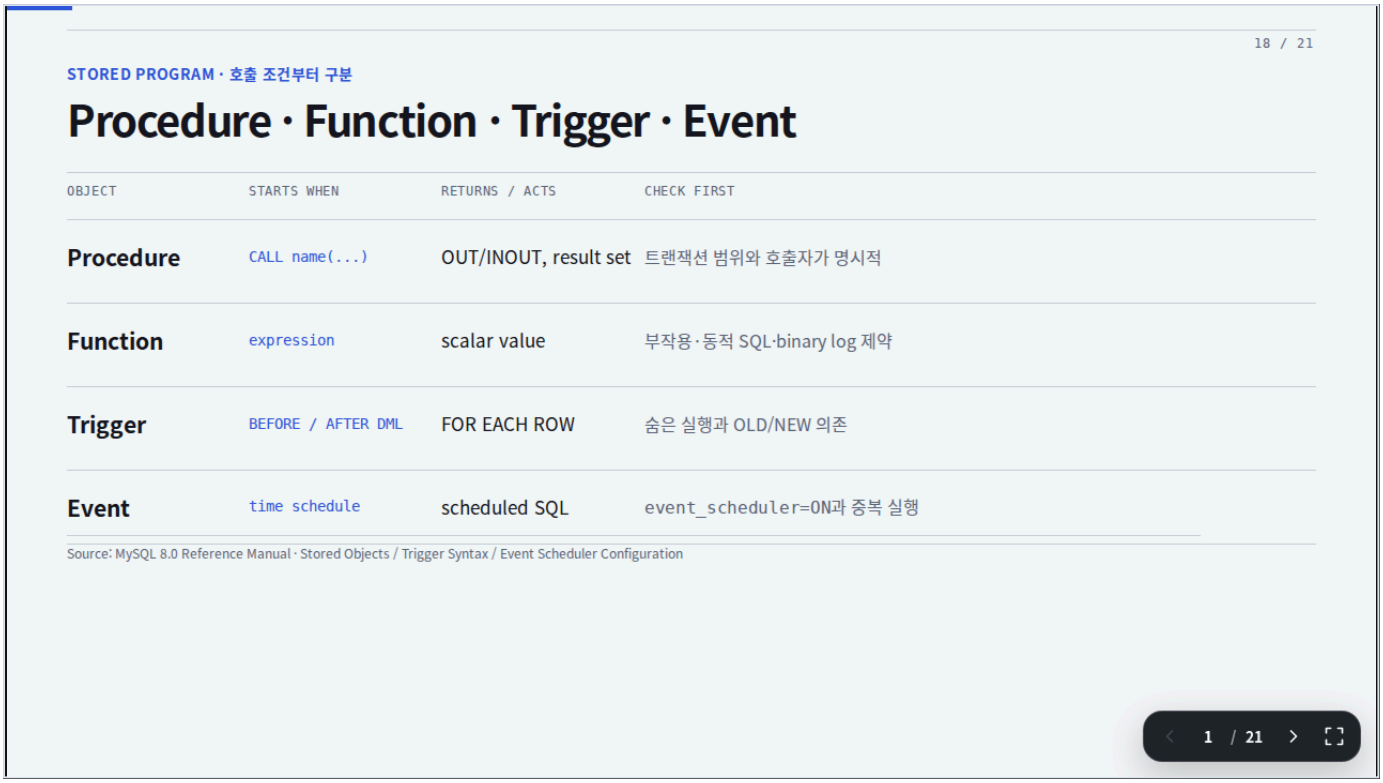
이 객체들을 쓰면 비즈니스 규칙을 데이터 가까이에 둘 수 있습니다. 대신 배포·버전 관리·테스트·관찰은 애플리케이션 코드보다 까다롭습니다. 팀의 운영 방식과 장애 대응 절차까지 보고 선택합니다. 공식 개요는 Stored Objects에 있습니다.

Stored Object에는 Stored Program뿐 아니라 View도 포함됩니다. 이번 발표는 실행 주체와 시점을 비교하려고 Procedure, Function, Trigger, Event에 범위를 맞췄습니다.

확인 포인트

- Routine·Program·Object의 포함 관계를 그린다.
- 네 객체를 호출 주체와 실행 시점으로 분류한다.

18. Procedure · Function · Trigger · Event



핵심 개념

Procedure는 여러 SQL 문장을 하나의 호출 단위로 묶거나 결과 집합과 출력 매개변수를 돌려줄 때 적합합니다. Function은 반드시 값을 반환하며 SQL 식 안에서 사용합니다. 데이터 변경과 외부 상태 의존이 큰 로직을 Function에 넣으면 평가 횟수와 부작용을 추적하기 어려워집니다.

흐름과 의미

Trigger는 테이블 이벤트에 묶여 자동 실행됩니다. MySQL Trigger는 `FOR EACH ROW` 방식이므로 한 문장이 10만 행을 바꾸면 트리거 본문도 행마다 실행됩니다. 호출자가 Trigger의 존재를 몰라도 동작합니다. 감사 열 기록이나 단순한 불변식 보강에는 유용하지만 숨은 비용과 재귀적 영향 범위를 주의합니다. 문법과 실행 조건은 Trigger Syntax and Examples에서 확인하십시오.

슬라이드가 생략한 세부 사항

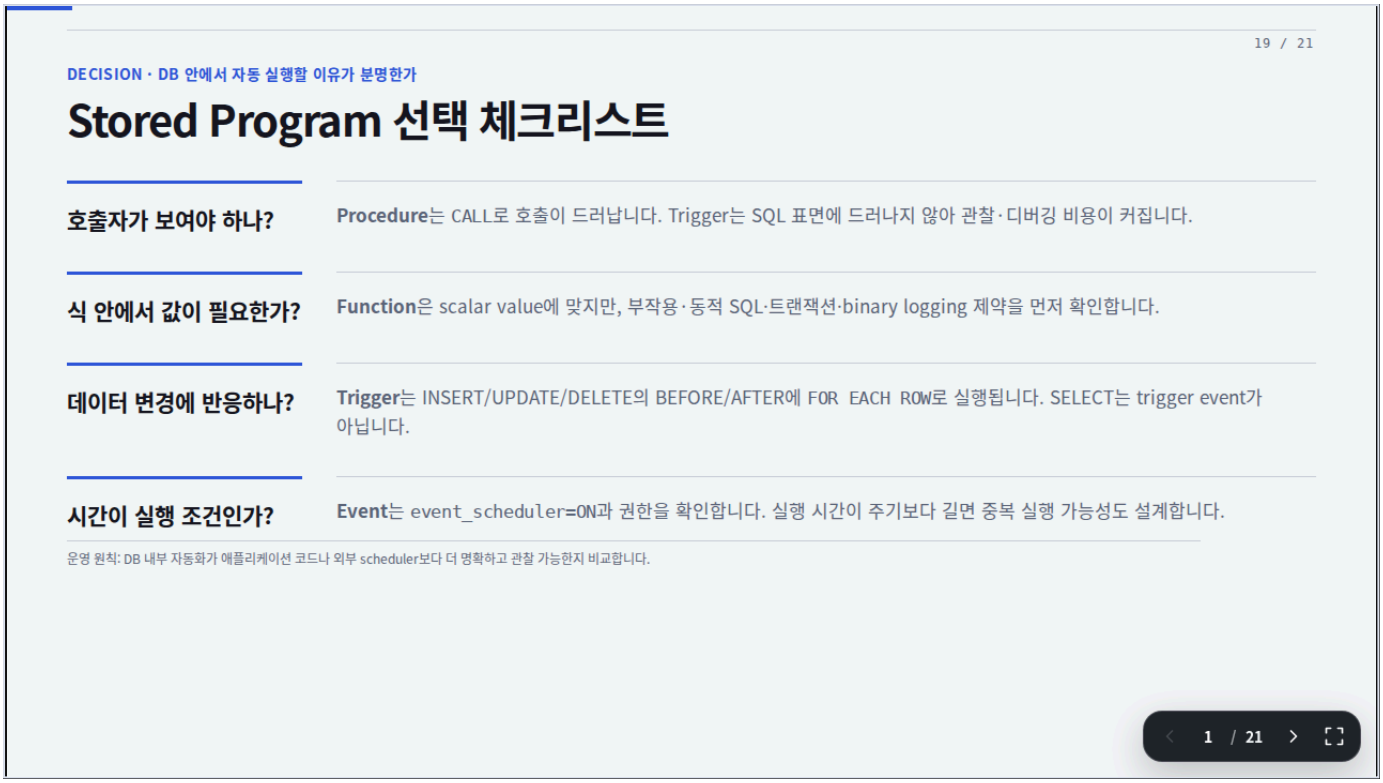
Event는 한 번 또는 반복 일정으로 실행하는 서버 내부 작업입니다. Event를 정의해도 `event_scheduler`가 `ON`이 아니면 실행되지 않습니다. 실행 주체, 실패 기록, 시간대, 중복 실행 가능성까지 운영 절차에 넣습니다. Event Scheduler Configuration과 Event Scheduler Overview을 함께 보십시오.

Trigger는 statement 단위가 아니라 `FOR EACH ROW`로 실행됩니다. Event는 정의만으로 동작하지 않고 `event_scheduler=ON`이 필요합니다. 두 객체 모두 호출 표면에서 보이지 않는 비용을 만들 수 있습니다.

확인 포인트

- Trigger의 `FOR EACH ROW` 비용을 변경 행 수와 연결한다.
- Event 실행 전에 `event_scheduler` 상태를 확인한다.

19. Stored Program 선택 체크리스트



핵심 개념

선택 기준은 실행 계기부터 잡으면 됩니다. 애플리케이션이 명시적으로 시작하는 다단계 작업은 Procedure가 자연스럽습니다. SQL 식에서 재사용할 계산이 필요하고 부작용을 통제할 수 있다면 Function을 검토합니다. 특정 테이블의 행 변경에 반드시 붙어야 하는 규칙은 Trigger 후보입니다. 정해진 시각이나 주기에 실행하는 내부 작업은 Event 후보입니다.

흐름과 의미

그다음에는 관찰 가능성과 실패 경계를 점검합니다. 누가 언제 실행했는지, 실행 시간과 변경 행 수를 어떻게 기록할지, 오류가 났을 때 transaction이 어디까지 rollback되는지 확인합니다. 권한과 `DEFINER` 계정의 수명, 백업·복구 시 객체 포함 여부, 배포 순서도 빠뜨리기 쉽습니다.

슬라이드가 생략한 세부 사항

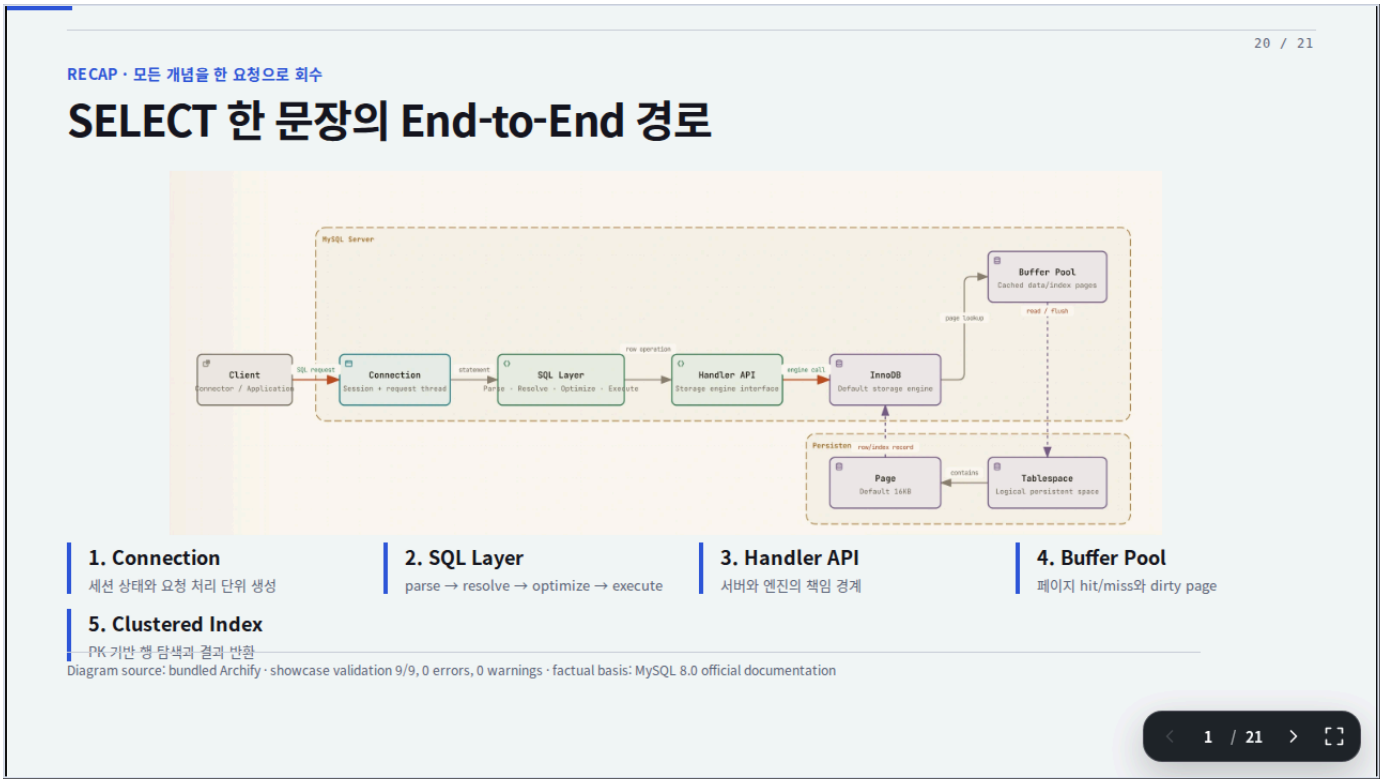
애플리케이션 job scheduler가 이미 표준이라면 Event를 더하기보다 기존 체계를 따르는 편이 운영에 유리합니다. 데이터베이스 안에서 실행해야 원자성과 데이터 근접성이 보장되는 작업도 있습니다. 어느 쪽을 고르든 로직의 존재가 숨지 않도록 소스 관리, 모니터링, 테스트 경로를 마련합니다.

DB 내부 로직은 스키마 백업, 권한, `DEFINER`, 복제, 장애 복구, 배포 순서에 함께 묶입니다. 기능 적합성만 보고 선택하면 운영 중 변경과 관찰이 어려워집니다.

확인 포인트

- 호출 가시성·실패 경계·권한을 선택 기준에 넣는다.
- 소스 관리와 모니터링 경로가 있는지 확인한다.

20. SELECT 한 문장의 End-to-End 경로



핵심 개념

마지막으로 `SELECT` 한 문장을 처음부터 다시 따라가 봅니다. Client가 Connection을 열고 인증을 마치면 Session 상태가 준비됩니다. 요청을 처리하는 Thread가 SQL 문자열을 받아 Parser와 Resolver/Preparation으로 넘깁니다. 문법과 객체·권한 확인이 끝나면 Optimizer가 통계와 비용을 바탕으로 실행 계획을 고릅니다.

흐름과 의미

Executor는 계획의 iterator를 실행하며 Handler API로 필요한 행을 요청합니다. InnoDB는 선택된 인덱스의 페이지를 Buffer Pool에서 찾습니다. cache miss가 나면 Tablespace에서 Page를 읽어 메모리에 올립니다. secondary index를 사용했고 필요한 열이 leaf record에 없다면 Primary Key로 clustered index를 다시 찾습니다.

슬라이드가 생략한 세부 사항

조건을 통과한 행은 Executor의 조인·필터·정렬·집계 단계를 거쳐 결과가 됩니다. 서버는 결과를 Connection으로 보내고 Client가 이를 받습니다. 느린 쿼리를 볼 때도 같은 길을 거꾸로 확인하면 됩니다. 연결 대기, 세션 메모리, 계획 오류, 많은 row 접근, Buffer Pool miss 가운데 어느 경계에서 시간이 늘었는지 찾습니다.

실제 SELECT에는 트랜잭션 가시성 확인, 잠금, 네트워크 패킷 전송이 더해집니다. 도식은 Parser부터 Page까지의 책임 경계를 선명하게 보이려고 이 세부 단계를 접었습니다.

확인 포인트

- Client에서 Page까지 정방향으로 한 번 설명한다.
- 병목 진단 때 같은 경로를 역방향으로 추적한다.

21. 세 가지 렌즈로 MySQL을 읽는다

핵심 개념

첫 번째 렌즈는 **책임 경계**입니다. MySQL 서버는 SQL과 실행 흐름을 맡고, InnoDB는 트랜잭션을 지원하는 레코드·인덱스·페이지 저장을 맡습니다. Handler API가 두 영역을 연결합니다.

흐름과 의미

두 번째 렌즈는 **수명과 범위**입니다. Connection과 Session, Thread는 비슷해 보여도 수명과 소유 상태가 다릅니다. Global Memory와 Session Memory도 공유 범위와 할당 시점이 다릅니다. 설정값의 최댓값만 곱하지 말고 실제 할당 조건을 확인합니다.

슬라이드가 생략한 세부 사항

세 번째 렌즈는 **접근 단위**입니다. SQL은 행을 요구하지만 InnoDB는 Page를 읽고 Buffer Pool에 캐시합니다. Clustered index와 secondary index는 그 페이지를 어떤 경로로 찾을지 결정합니다. 세 렌즈를 겹치면 연결 문제, 실행 계획 문제, I/O 문제를 같은 그림 안에서 구분할 수 있습니다.

발표 뒤에는 실제 쿼리 하나를 골라 `EXPLAIN ANALYZE`, Performance Schema, InnoDB Buffer Pool 지표를 함께 확인해 보십시오. 아키텍처 용어가 운영 지표와 연결될 때 비로소 문제 해결 도구가 됩니다.

세 렌즈는 정답을 자동으로 주는 체크리스트가 아닙니다. 같은 증상도 버전, 에디션, 설정, 데이터 분포에 따라 원인이 달라집니다. 측정값으로 가설을 확인하는 순서를 유지합니다.

확인 포인트

- Boundary·Plan·Page 세 질문으로 실습 쿼리를 점검한다.
- 실제 버전과 workload 측정값으로 가설을 검증한다.