파일 시스템

InnoDB

트랜잭션

MySQL의 유령

Transaction과 Phantom 살펴보기

Phantom Read

Phantom OF MySQL

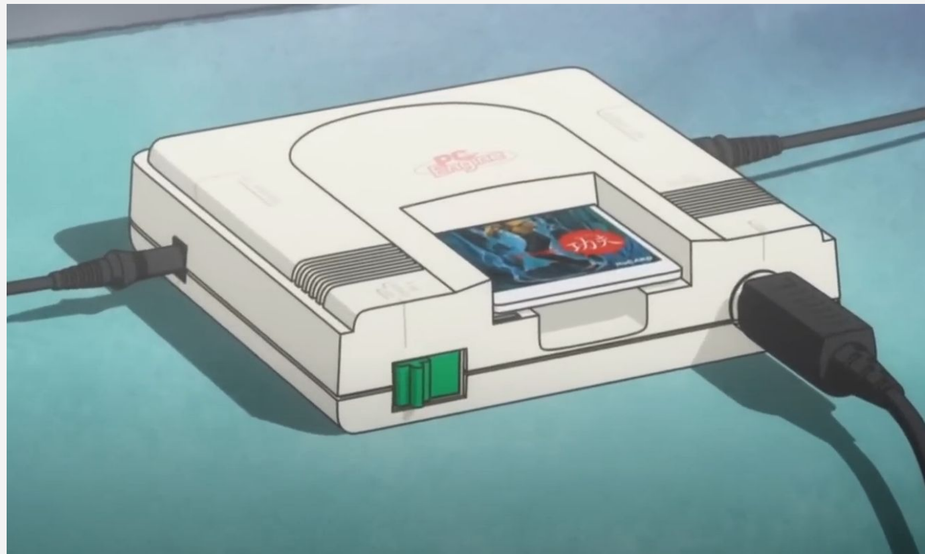
Table of Contents

About me

발표자 소개

- 1987년 세운상가표 APPLE][로 BASIC, 6502 Assembly로 코딩 시작
- 미래소년 코난을 보고 배터리를 만들고 싶어서 재료공학 전공
- 현대물리가 너무 어려워서 포기하고 데이터베이스 분야 박사학위 취득

- LG 전자 소프트웨어 플랫폼 연구소 선임연구원
- NHN NEXT 교수
- Amazon Web Services Senior Technical Trainer
- (현) 교육기업 코드스쿼드 창업 7년차
- **int *p;**
- **절대 고수 아님**



유령의 집 터전

Linux File System

Linux

Linux

리눅스 커널을 기반으로 한 오픈소스 OS

시장 점유율: 2%

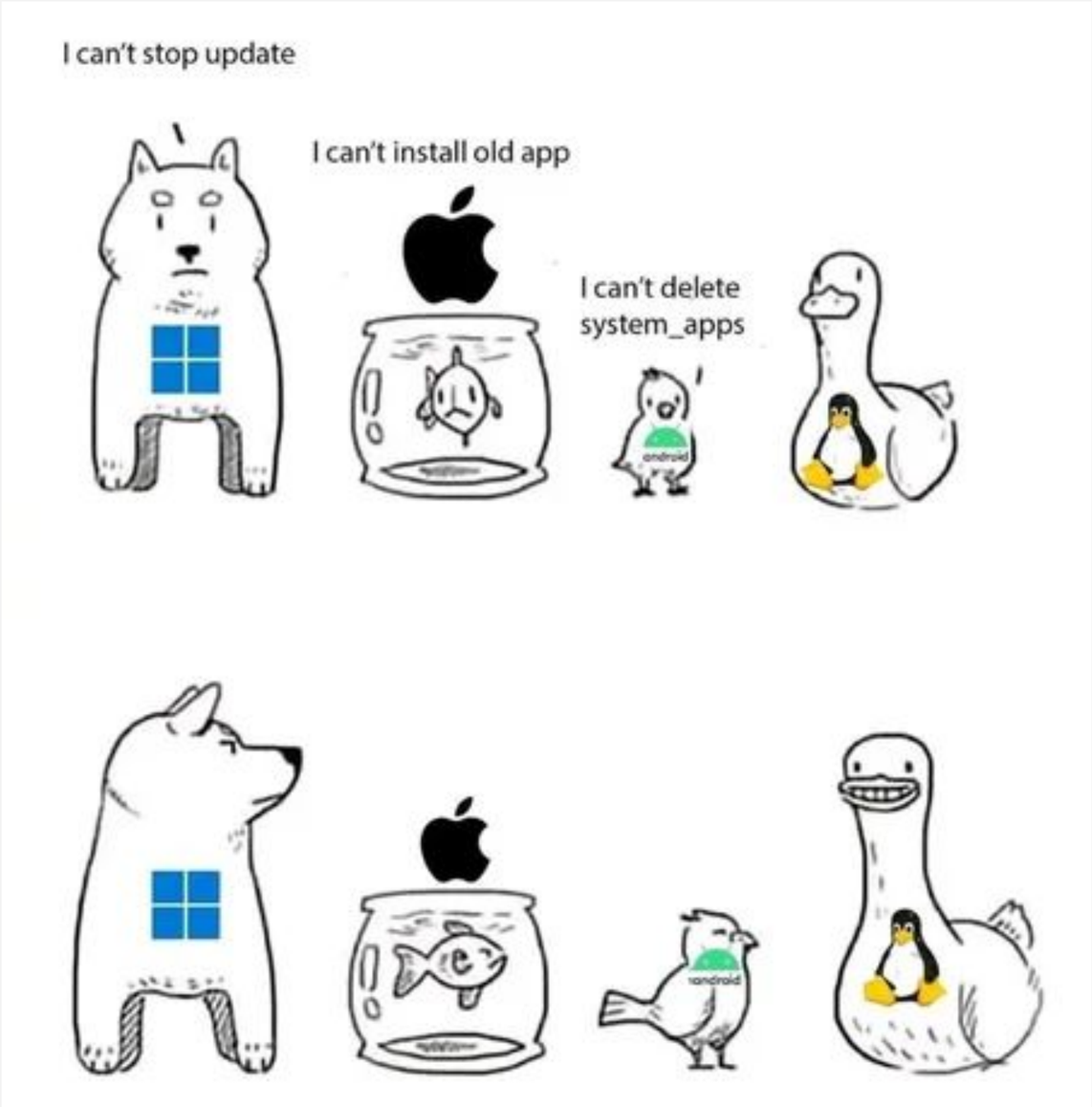
Linux

Linux

서버 환경의 90%

40억대의 안드로이드 기기

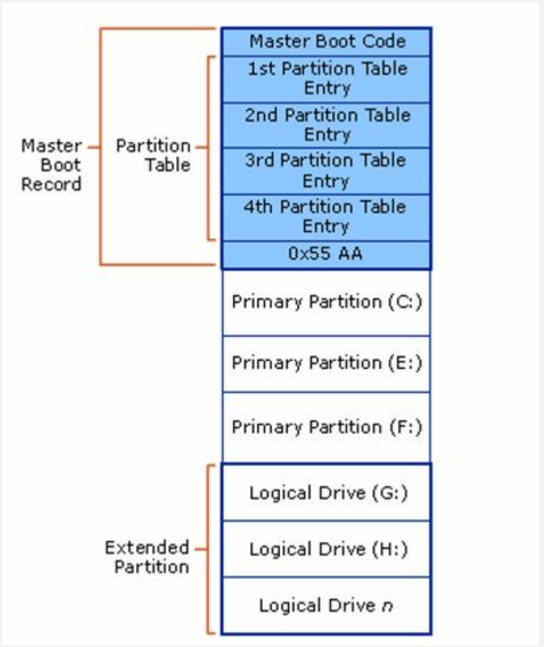
그냥 아주 매우 멋짐!

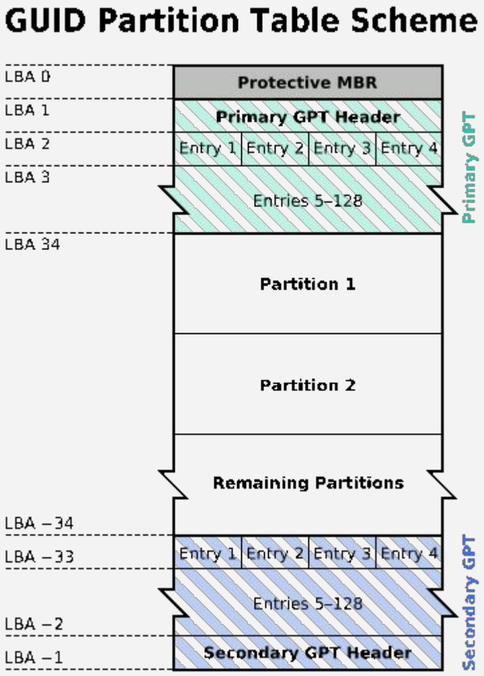


Linux File System

SSD 부착 후 초기화
파티션 정보 생성

```
$ lsblk
$ fdisk -l
$ fdisk /dev/sdb
# gdisk /dev/sdb
```



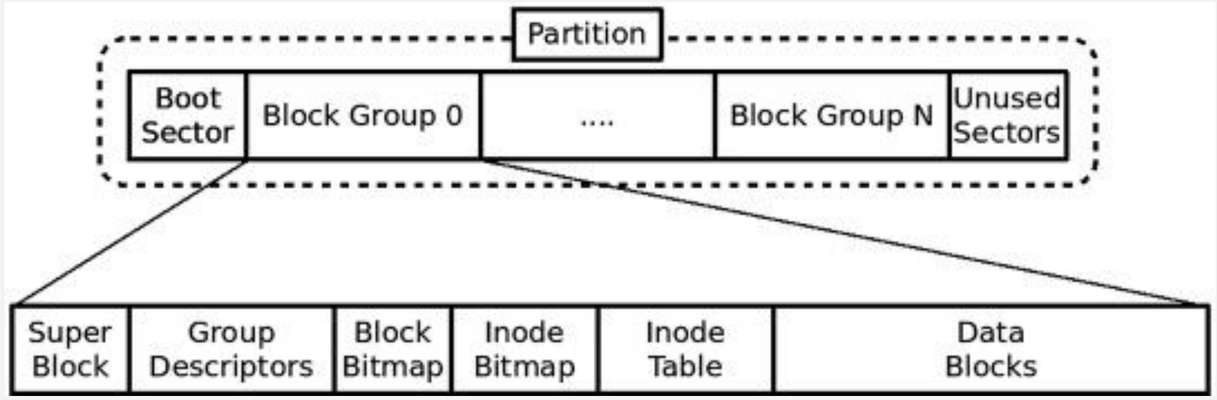


https://s905060.gitbooks.io/site-reliability-engineer-handbook/content/the_differences_between_mbr_and_gpt.html

Linux File System

포맷 및 파일 시스템 연결
mkfs.ext4 && mount

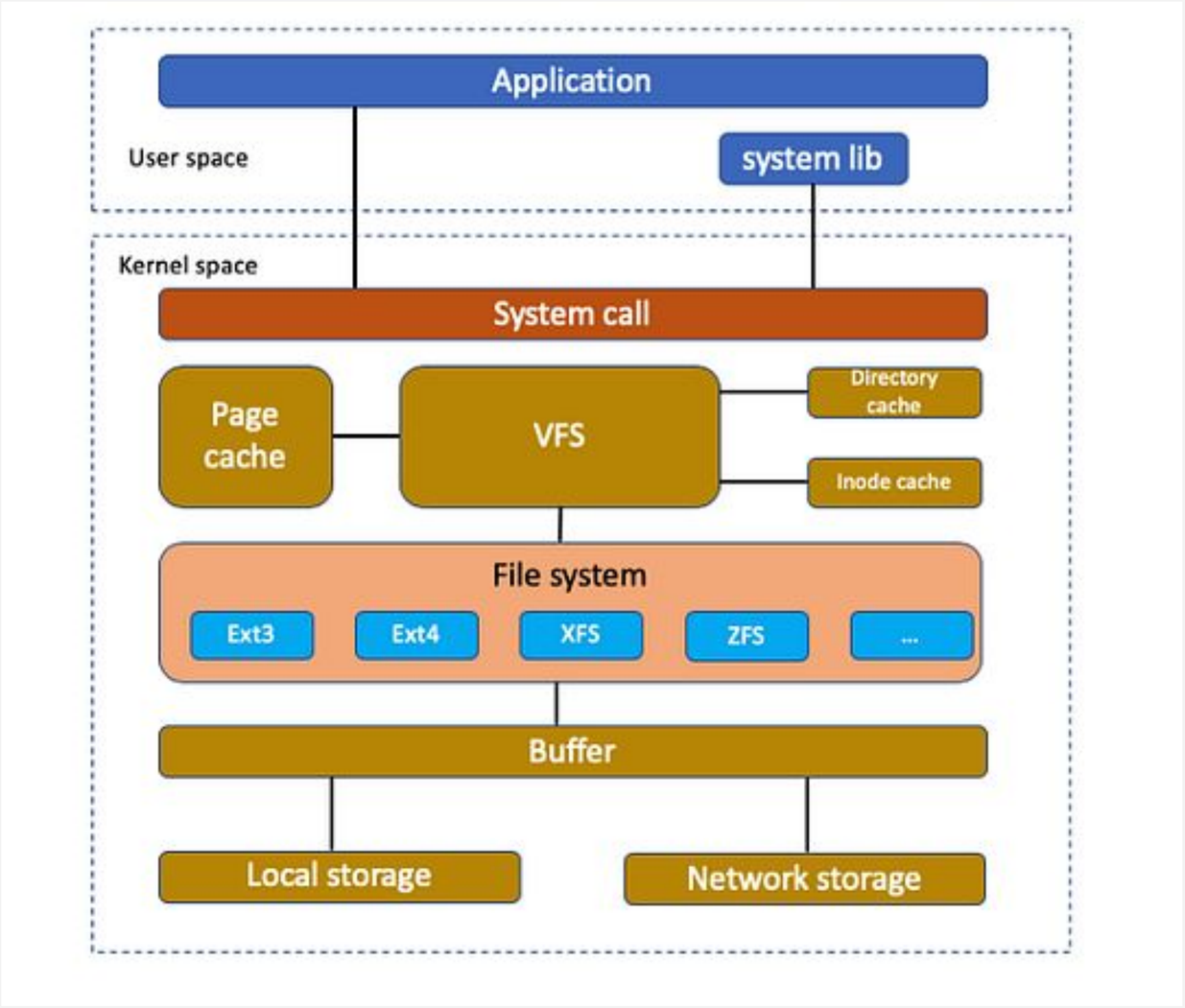
```
$ mkfs.ext4 /dev/sdb1
$ mount /dev/sdb1 /media/db
```



Linux File System

돌발 퀴즈

Java 또는 다른 언어에서 시스템 콜을 호출하려면 어떻게 해야 하나요?

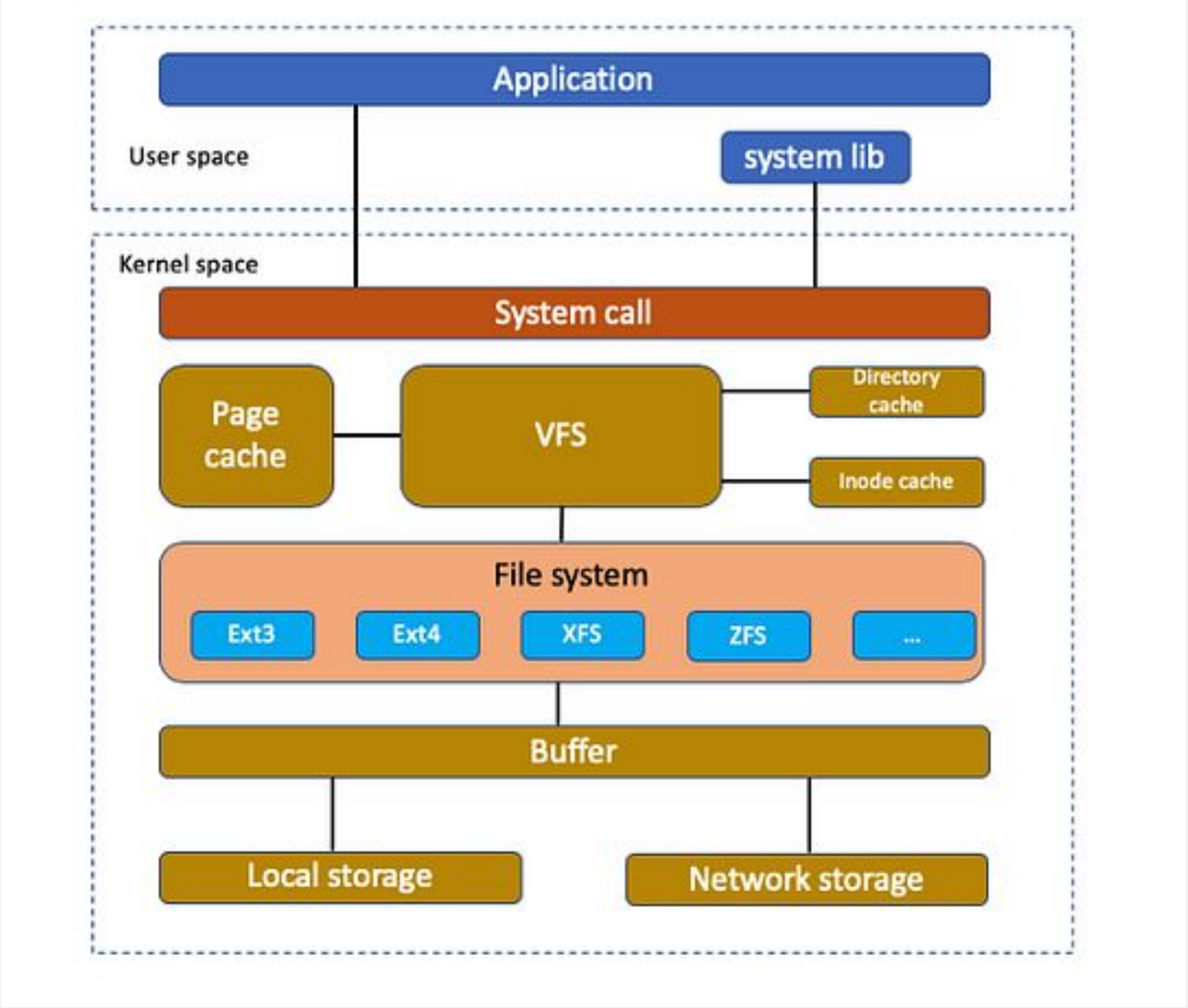


Linux File System

VFS && Ext4

- VFS - IO 서브시스템에서 추상화 담당

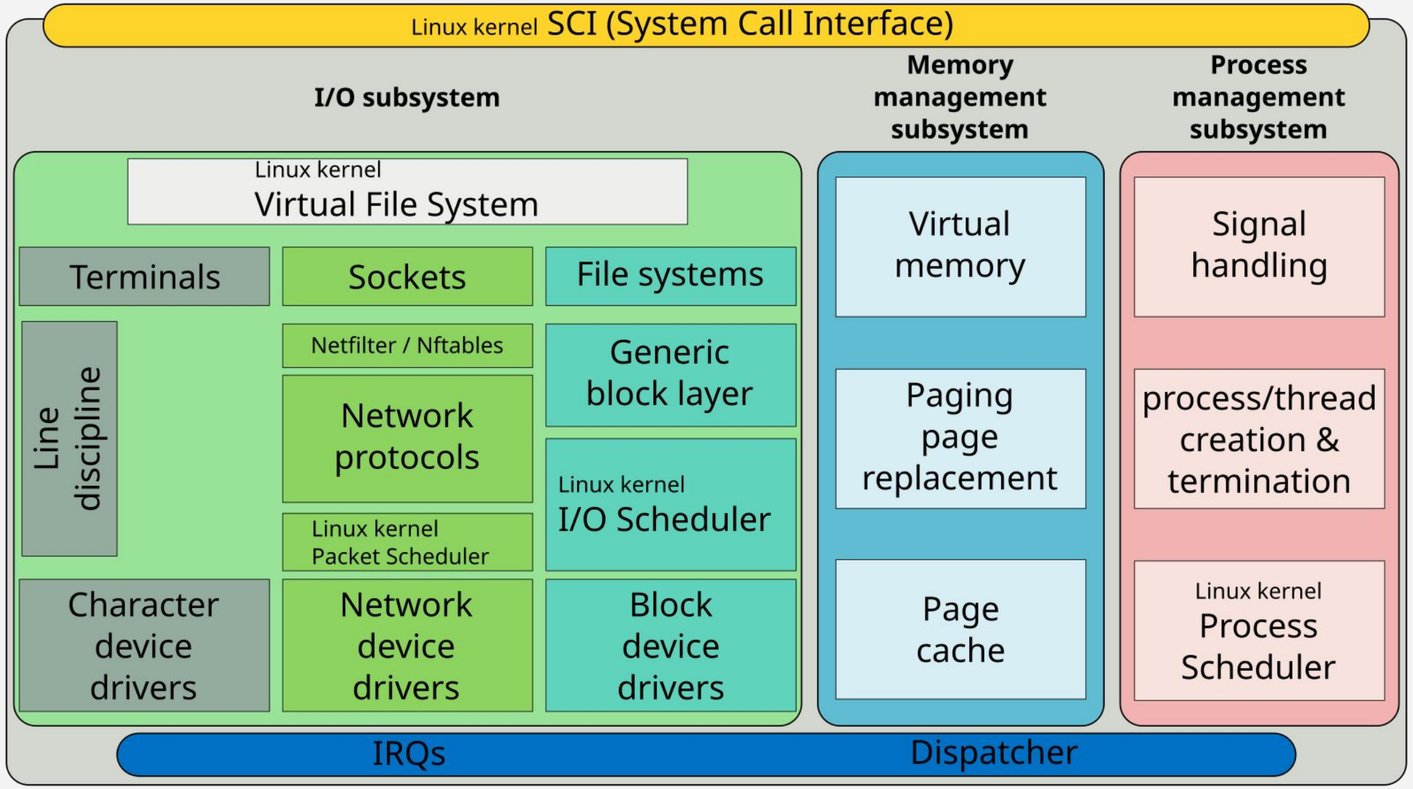
- 파일 시스템간의 차이를 숨기고
  어플리케이션이 일관된 방식으로
  파일을 처리할 수 있게 해 줌



Linux File System

리눅스 커널과 서브시스템
kernel, SCI, I/O subsystem, VFS, ...

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



https://ko.m.wikipedia.org/wiki/%ED%8C%8C%EC%9D%BC:Simplified_Structure_of_the_Linux_Kernel.svg

집을 지어 봅시다

InnoDB

MySQL

MySQL 설치

```
$ sudo apt-get install build-essential cmake libncurses5-dev libssl-dev bison
$ git clone https://github.com/mysql/mysql-server.git
$ mkdir build; cd build
$ cmake .. -DDOWNLOAD_BOOST=1 -DWITH_BOOST=/path/to/boost
$ make; make install
$ sudo systemctl enable mysqld
```

create database

데이터베이스 및 테이블 생성

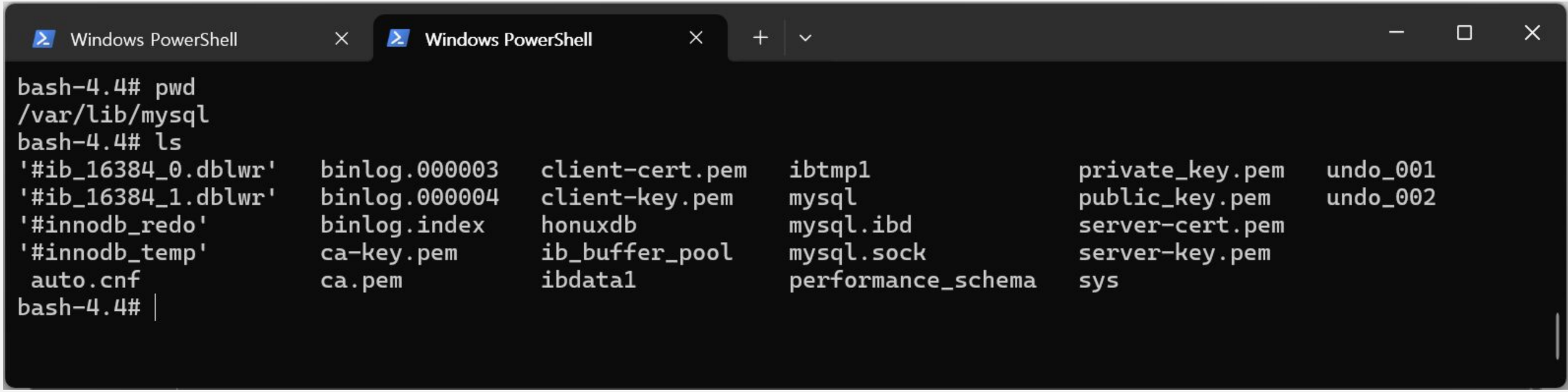
```
mysql> create database honuxdb;

mysql> use honuxdb;

mysql> CREATE TABLE HTEST (ID INT PRIMARY, ...);
```

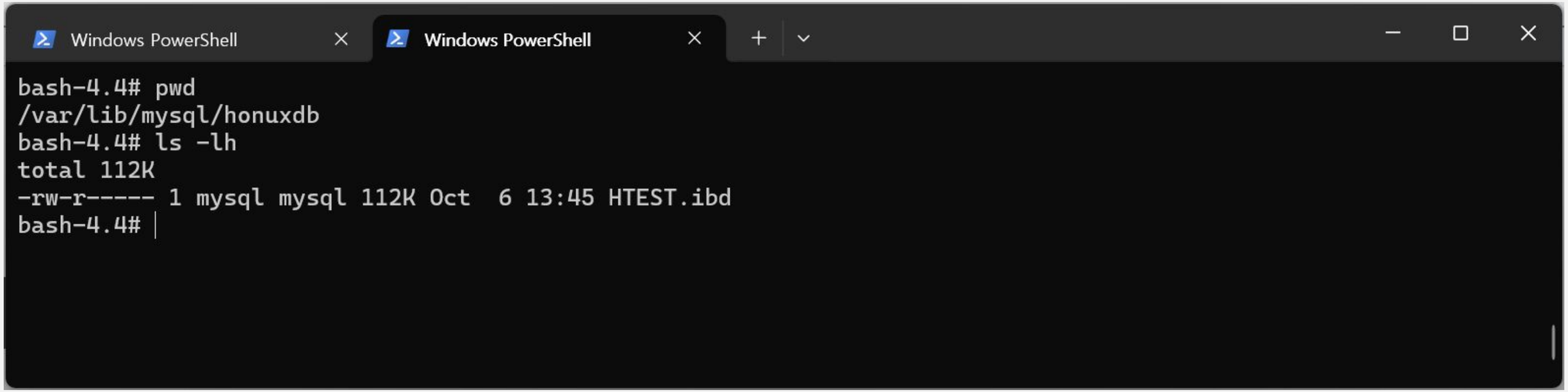
InnoDB

파일 시스템 계층에는 무슨 일이 일어나나?



InnoDB

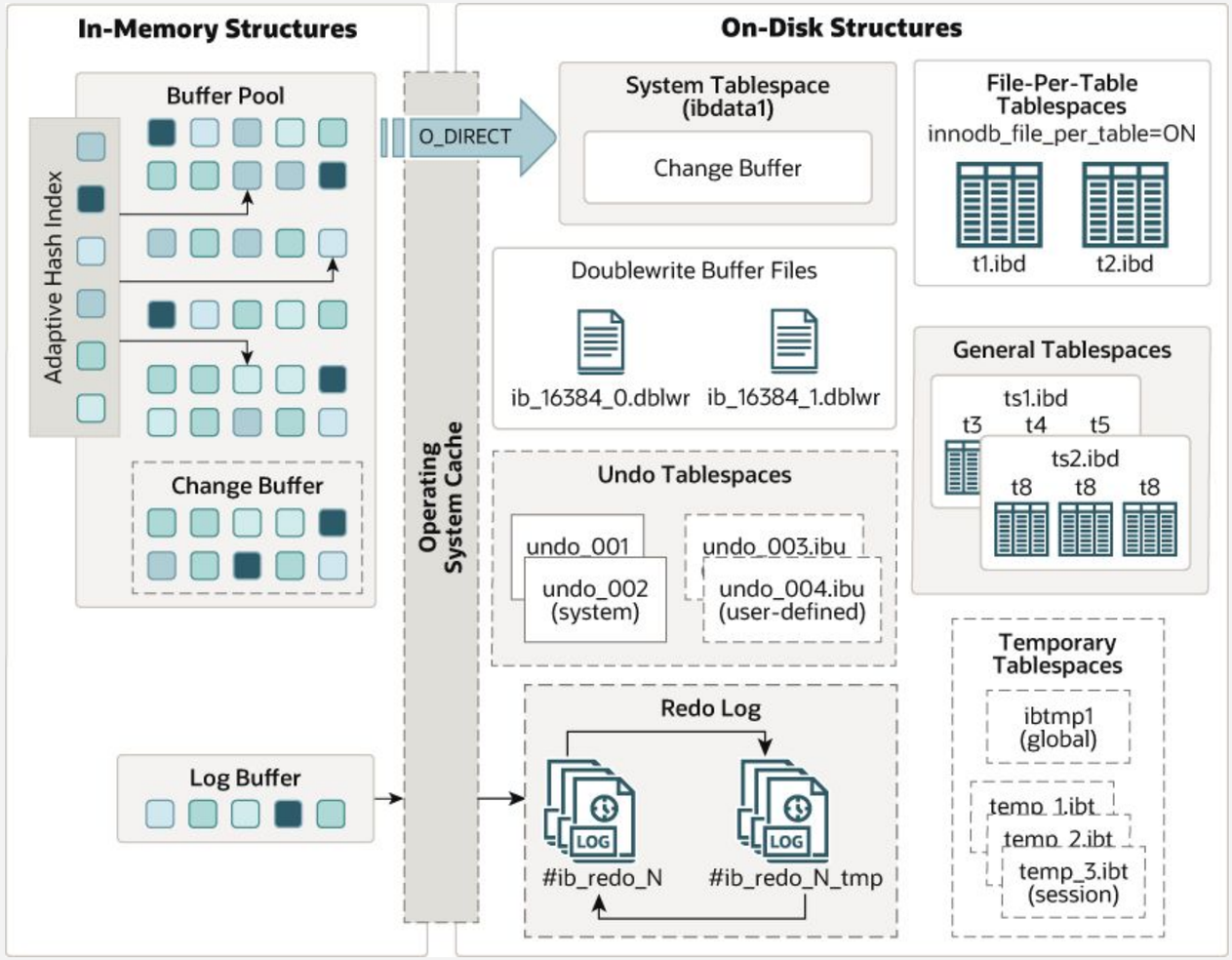
파일 시스템 계층에는 무슨 일이 일어나나?



InnoDB

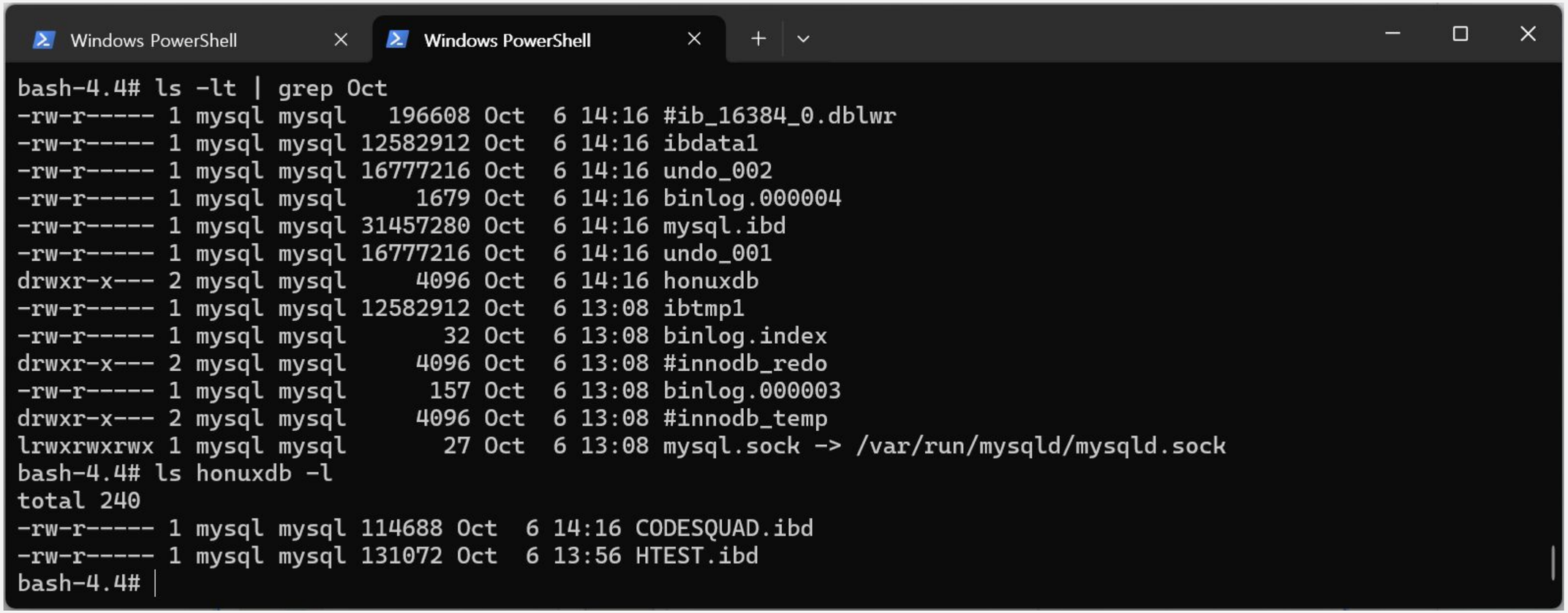
MySQL 8.X InnoDB

Buffer Pool
Change Buffer
System Tablespace (O_DIRECT)
Doublewrite Buffer
Undo Tablespaces
Log Buffer
Redo Log - WAL
General Tablespaces
Temporary Tablespaces



InnoDB

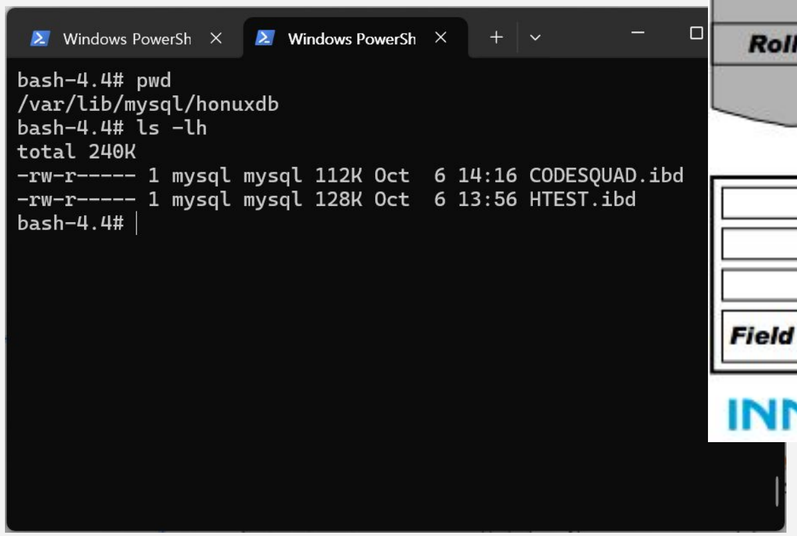
MySQL 8.X InnoDB

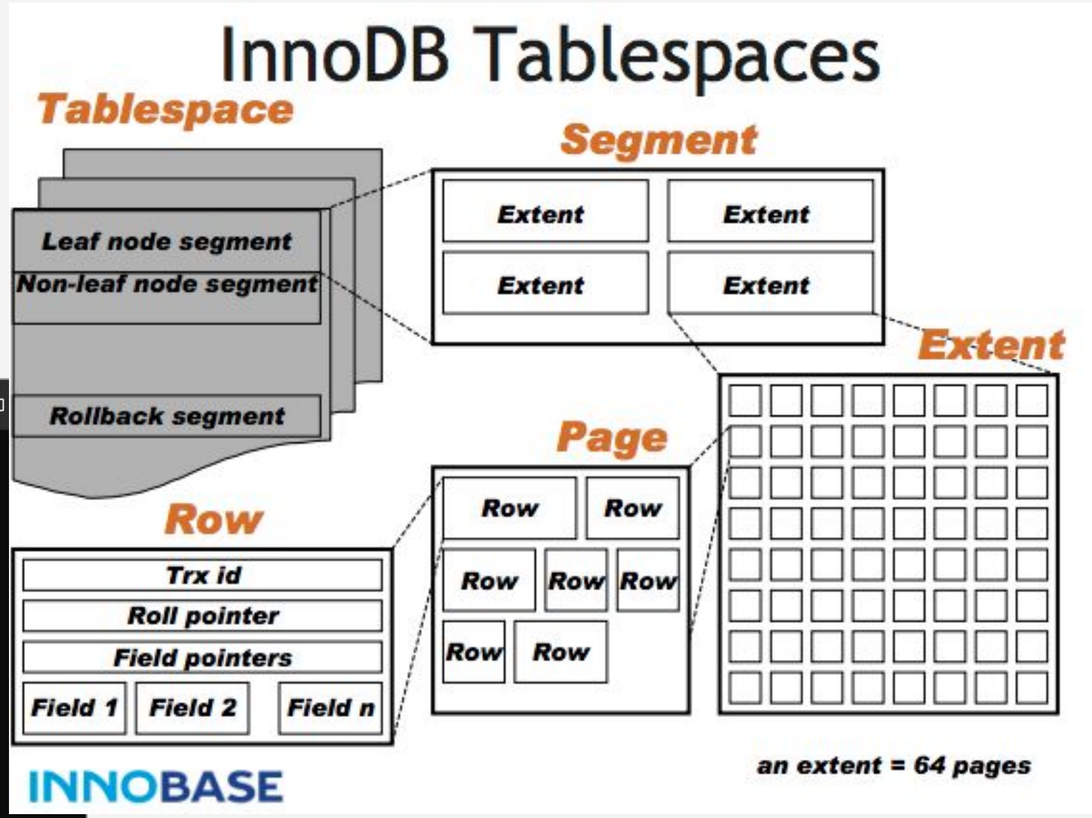


InnoDB

Tablespace

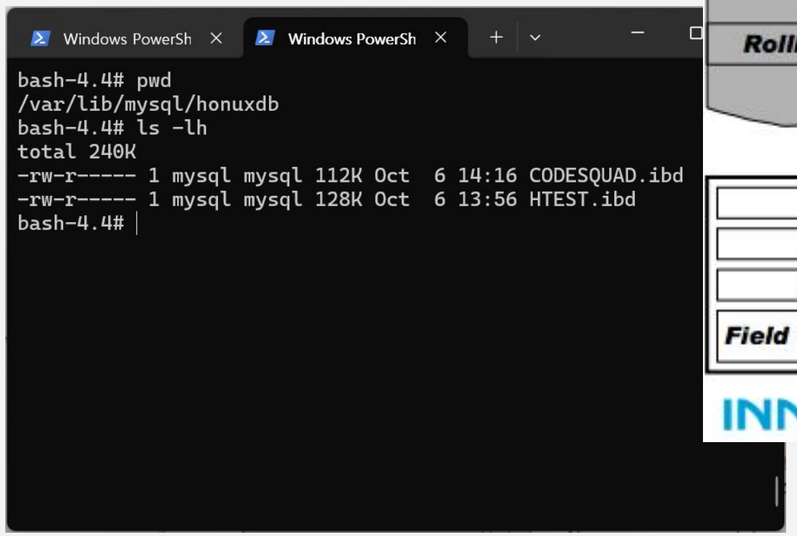
- System
- File-Per-Table
- General
- Temporary
- Undo

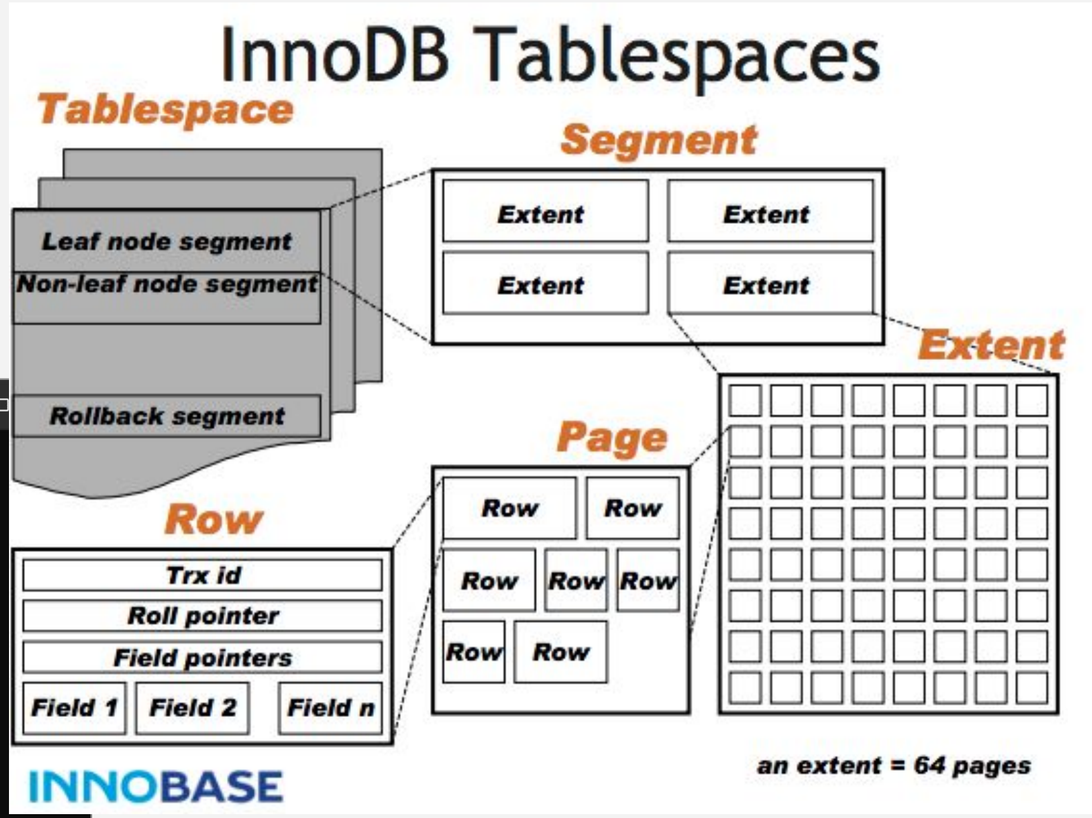




InnoDB

질문: drop table HTEST; 하면 무슨 일이 벌어질까요?

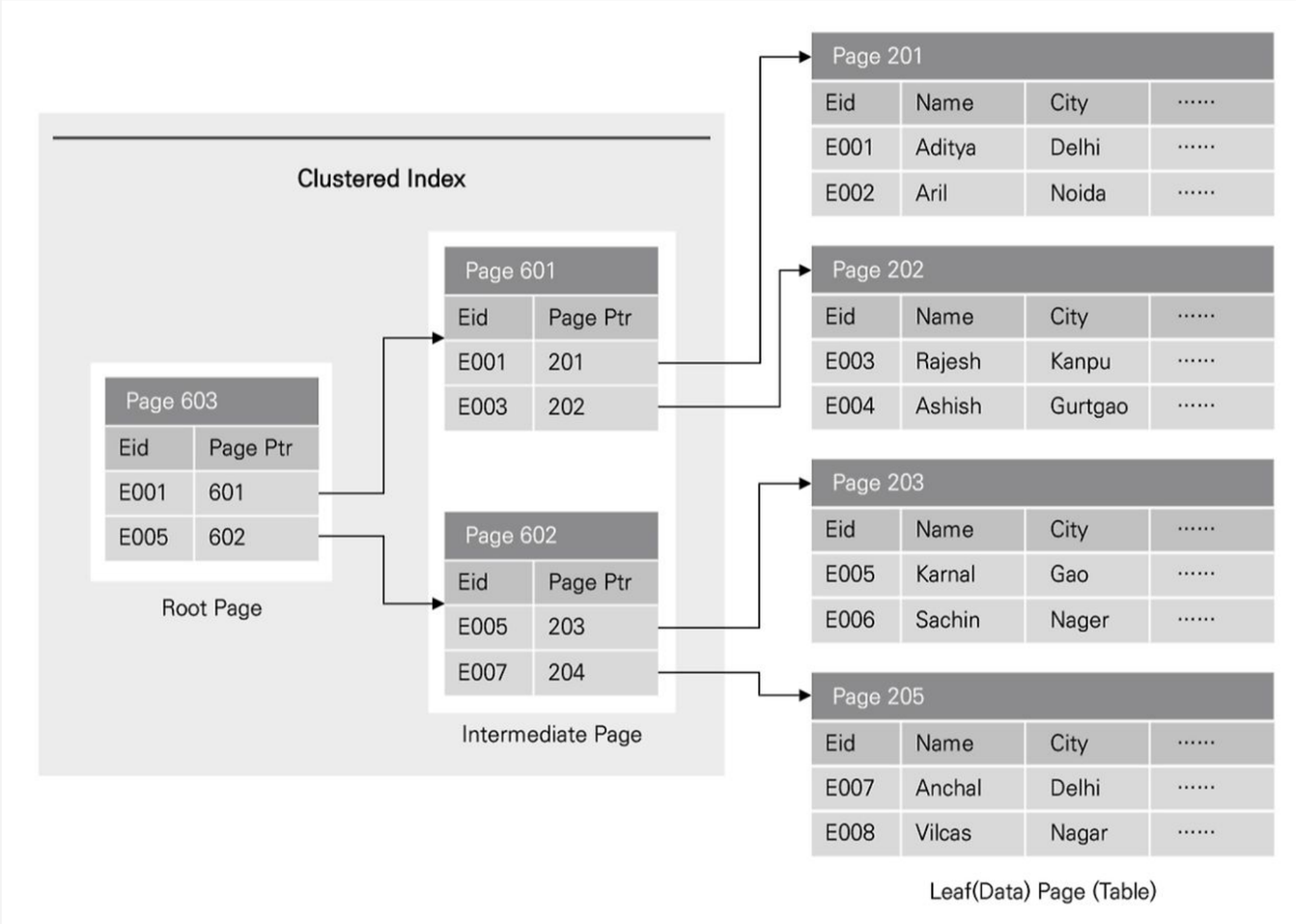




InnoDB

Clustered Index

- 테이블 생성시 무조건 생성됨
- B+ tree
- K = PK, V = 레코드
- 데이터는 리프 노드에만 존재
- 데이터는 PK 기준으로 정렬되어 있음



https://blog.ex-em.com/1699

InnoDB

Secondary Index

CREATE INDEX 명령으로 생성

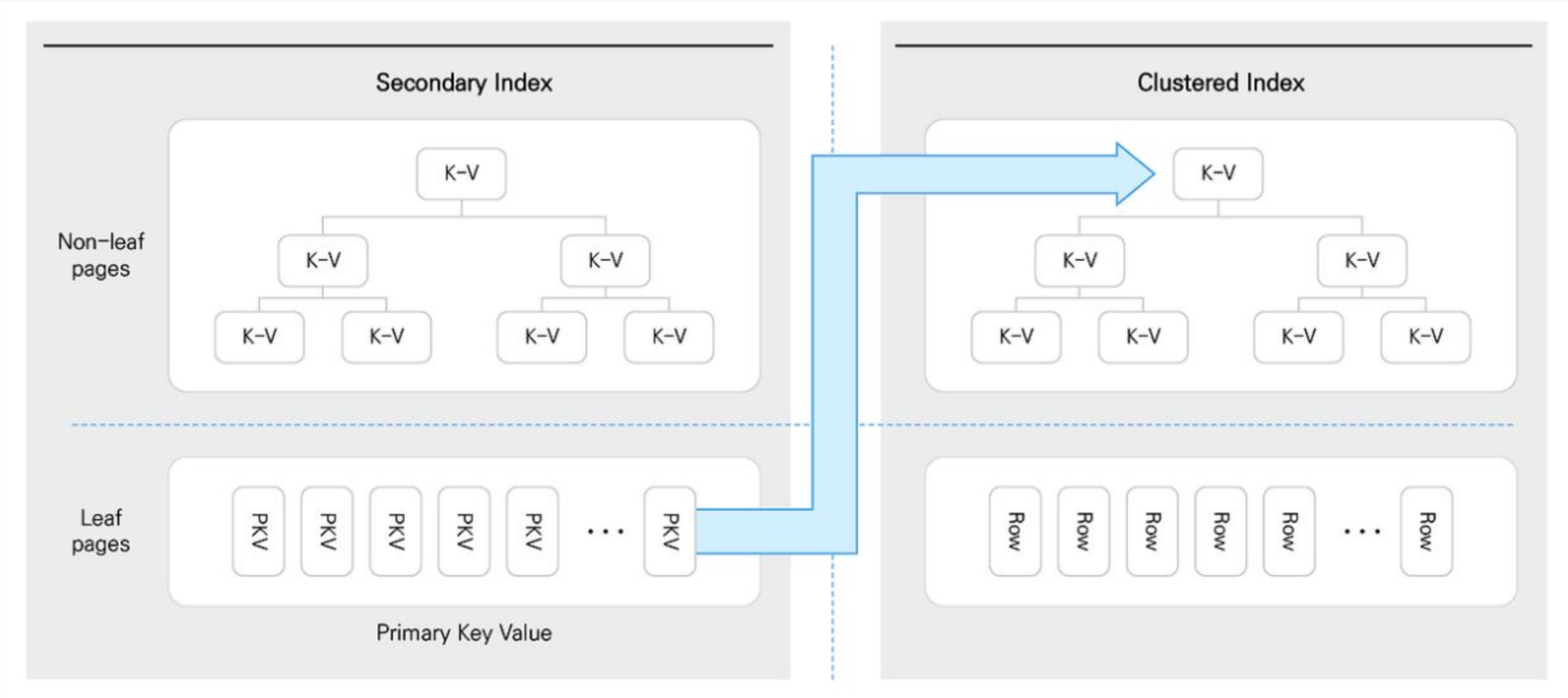
B+ tree

Inverted Index

R-Tree

Hash Table

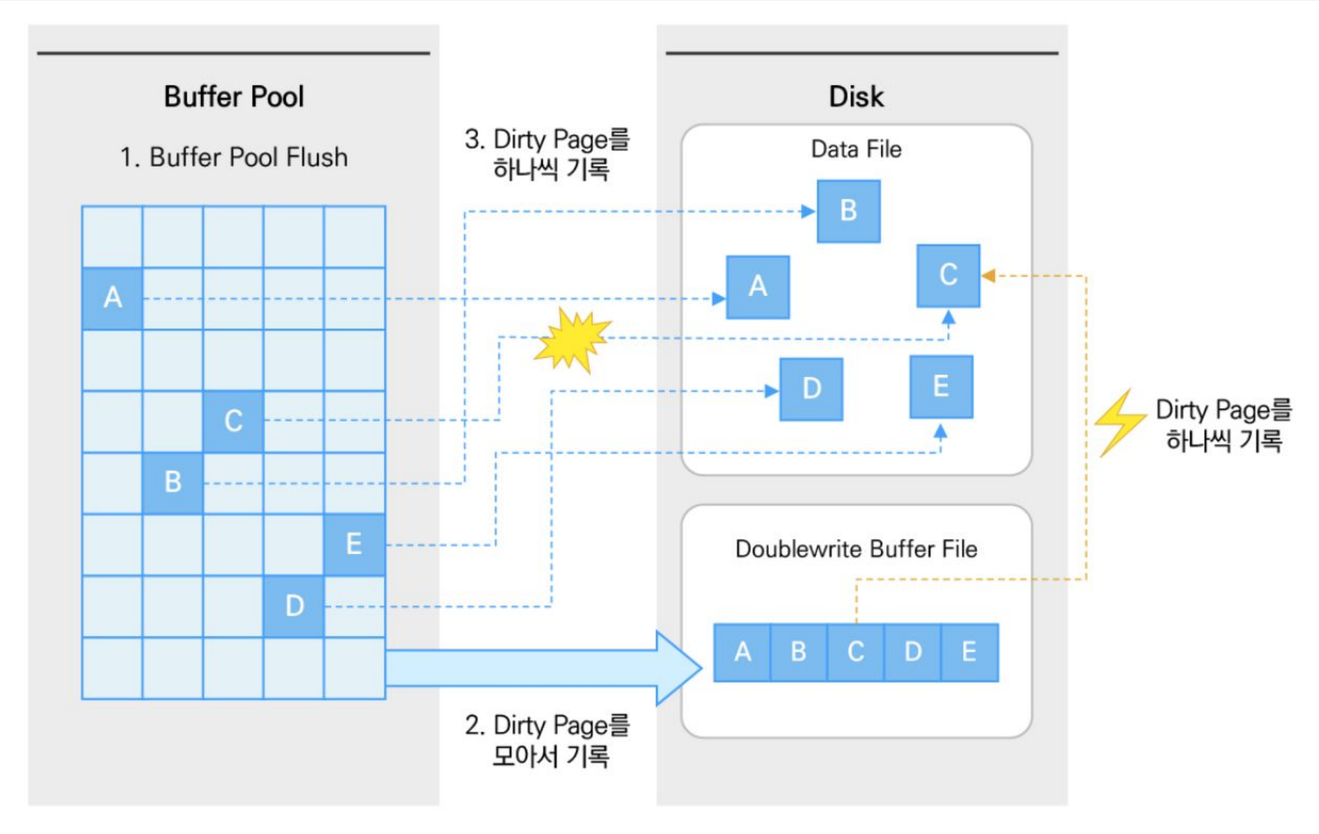
K = 지정 컬럼, V = PK



InnoDB

Double Write Buffer

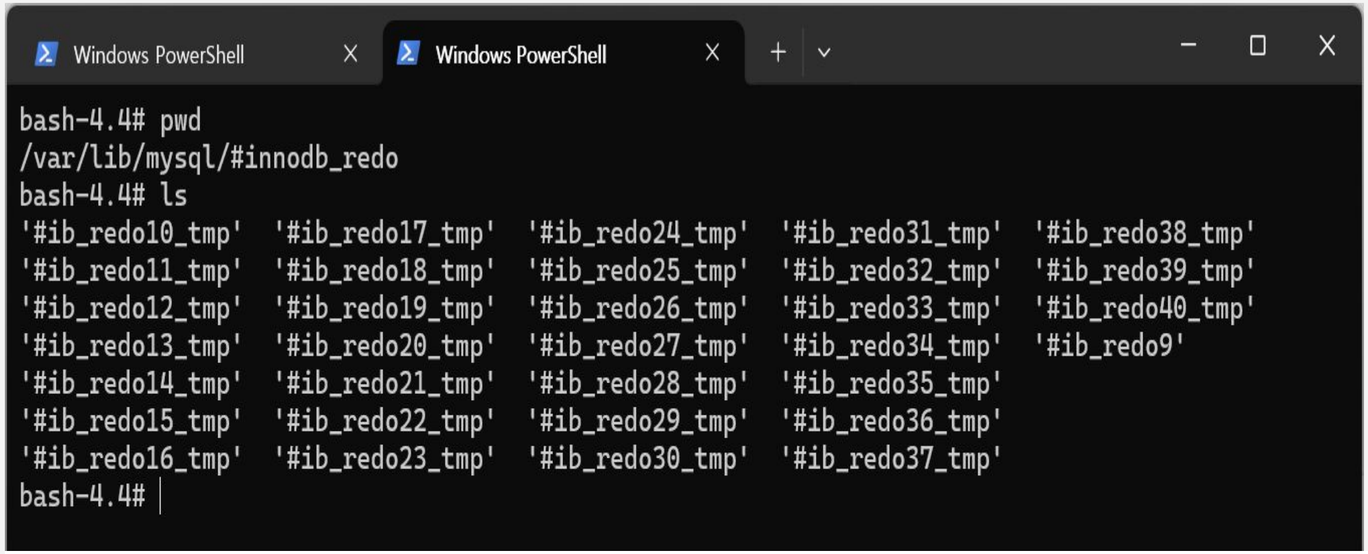
쓰기 성능 및 안정성 향상



InnoDB

Redo Log

- ib_redoXX_tmp
- WAL (Write Ahead Log)
- 버퍼의 변경사항은 로그에 우선 저장됨
- **8.X 부터 일시적 비활성화 가능!**



ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging

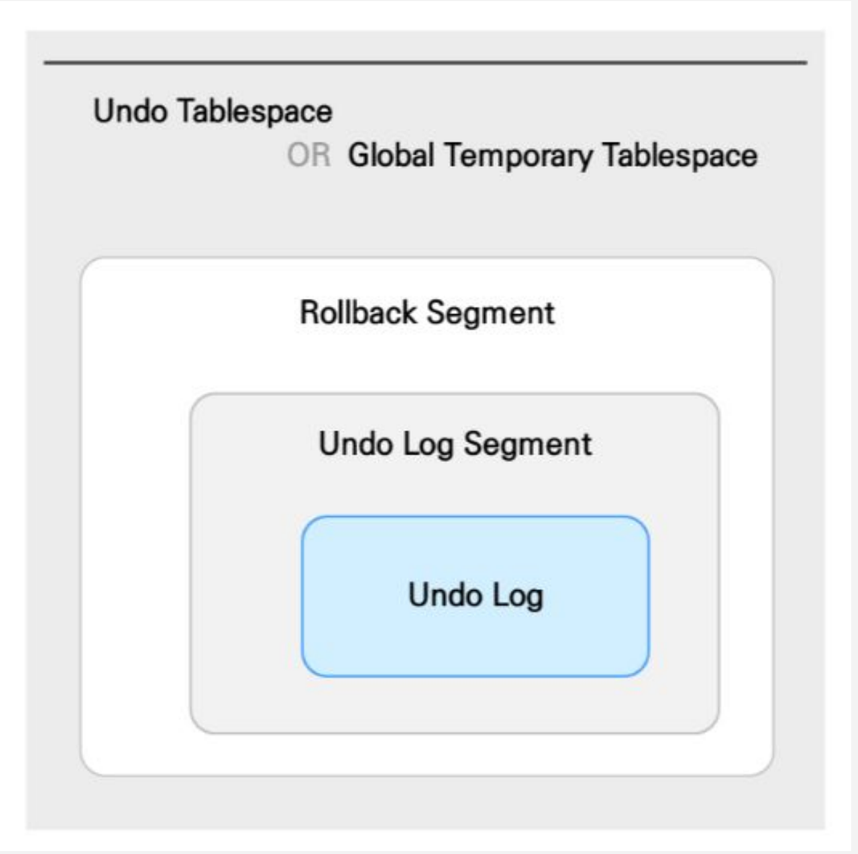
InnoDB

Undo log

Undo 테이블 스페이스 내의 롤백 세그먼트에 Undo Log 저장

데이터의 이전 버전을 Undo 로그를 통해 읽을 수 있음

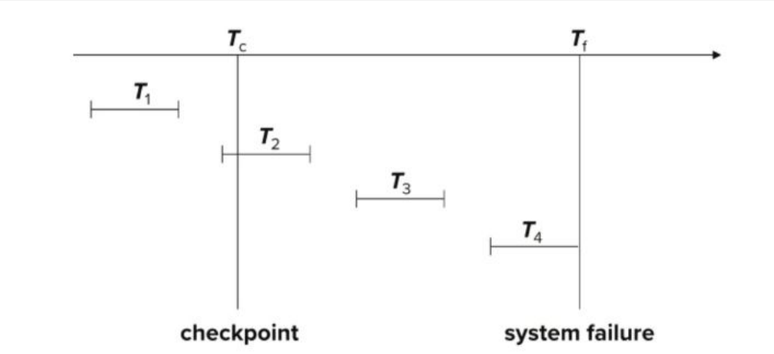
MVCC, 롤백에 활용됨

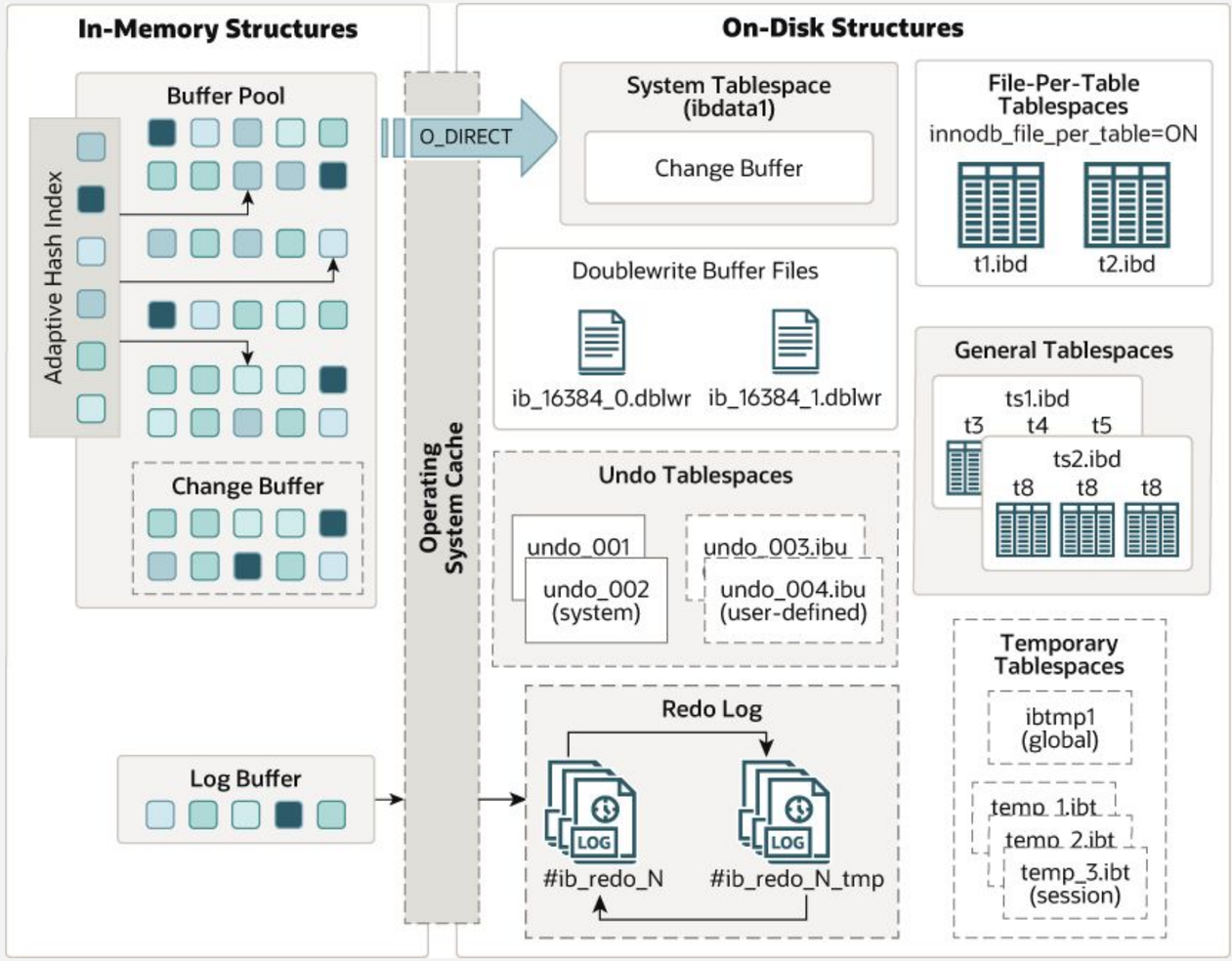


InnoDB

InnoDB 다시 살펴 보기

트랜잭션 시작
레코드 변경
체크포인트 (Fuzzy Checkpoint)
커밋
롤백
InnoDB 구조와 함께 설명해 보자





Transaction Isolation Mode

Transaction

MySQL Lock

- **Shared and Exclusive Locks**
- Intention Locks
- Record Locks
- Gap Locks
- Next-Key Locks
- Insert Intention Locks
- AUTO-INC Locks
- Predicate Locks for Spatial Indexes

Transaction

SQL92 Transaction Isolation Level

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

Transaction

동시성 문제

- **Lost Update Problem (P0)**
- Dirty Read Problem (P1)
- Non Repeatable Read Problem (P2)
- Phantom Read Problem (P3)

Transaction

Transaction Isolation Mode

SQL-92 Isolation Levels

	Isolation Level	Dirty Read	Non-Repeatable Read	Phantom Read
1	READ UNCOMMITTED	Yes	Yes	Yes
2	READ COMMITTED	No	Yes	Yes
3	REPEATABLE READ	No	No	Yes
4	SERIALIZABLE	No	No	No

Transaction

MySQL의 트랜잭션 고립레벨

- 기본 모드는 Repeatable Read
- Read Committed, Repeatable Read → **MVCC 사용**
- **SQL 92 Repeatable Read - 팬텀 발생**
- **MySQL의 Repeatable Read에서는**
  Phantom Read가 발생하지 않는다!

SQL-92 Isolation Levels

	Isolation Level	Dirty Read	Non-Repeatable Read	Phantom Read
1	READ UNCOMMITTED	Yes	Yes	Yes
2	READ COMMITTED	No	Yes	Yes
3	REPEATABLE READ	No	No	Yes
4	SERIALIZABLE	No	No	No

Transaction

질문: Lost Update Problem은 어떻게 처리해야 하나요?

- **Lost Update Problem (P0)**
- Dirty Read Problem (P1)
- Non Repeatable Read Problem (P2)
- Phantom Read Problem (P3)

Transaction

Read Uncommitted

Transaction A

1) START TRANSACTION;

3) SELECT name FROM emp WHERE id=1

-- 결과: Alice

5) SELECT name FROM emp WHERE id=1

-- 결과: Ali

7) SELECT name FROM emp WHERE id=1

-- 결과: Alice

Transaction B

2) START TRANSACTION;

4) UPDATE emp SET name = “Ali” WHERE id = 1

6) ROLLBACK;

Transaction

Read Uncommitted

- 커밋되지 않은 데이터도 모두 읽을 수 있음
- MVCC를 사용하지 않고 데이터를 직접 읽음
- **어디에서 데이터를 읽나? Buffer Pool**

  → Dirty Read 발생 O

  → Non Repeatable Read 발생 O

  → Phantom Read 발생 O

Transaction

Read Committed

Transaction A

1) START TRANSACTION;

3) SELECT name FROM emp WHERE id=1

-- 결과: Alice

5) SELECT name FROM emp WHERE id=1

-- 결과: Alice

7) SELECT name FROM emp WHERE id=1

-- 결과: Ali

Transaction B

2) START TRANSACTION;

4) UPDATE emp SET name = “Ali” WHERE id = 1

6) Commit;

Transaction

Read Committed

- 커밋된 데이터를 즉시 읽음
- MVCC 사용
- **어디에서 데이터를 읽나? Buffer Pool + Undo log (Read view)**
- **트랜잭션 ID로 데이터의 정합성 판단**

  → Dirty Read 발생 X

  → Non Repeatable Read 발생 O

  → Phantom Read 발생 O

Transaction

Repeatable Read

Transaction A

1) START TRANSACTION;

3) SELECT name FROM emp WHERE id=1

-- 결과: Alice

5) SELECT name FROM emp WHERE id=1

-- 결과: Alice

7) SELECT name FROM emp WHERE id=1

-- 결과: Alice

Transaction B

2) START TRANSACTION;

4) UPDATE emp SET name = “Ali” WHERE id = 1

6) Commit;

Transaction

Repeatable Read

- 트랜잭션 시작시점의 스냅샷 활용
- MVCC를 사용함
- **어디에서 데이터를 읽나? Buffer Pool + Undo log (Snapshot)**

  → Dirty Read 발생 X

  → Non Repeatable Read 발생 X

  → Phantom Read 발생 X

Transaction

Serializable

- MVCC를 사용하지 않음
- 어디에서 데이터를 읽나? 버퍼 풀
- **S Lock과 X Lock을 사용함 → 데드락과 동시성 저하 유발**

  → Dirty Read 발생 X

  → Non Repeatable Read 발생 X

  → Phantom Read 발생 X

MySQL과 Phantom Read

Phantom Read

(1) 특정 조건으로 데이터를 조회

(2) 다른 트랜잭션이 데이터를 삽입하거나 삭제

(3) 원래 트랜잭션에서 같은 조건으로 다시 조회할 때

새로운 레코드가 나타나거나 사라진 상황 발생!!

Phantom Read

Phantom Read - 예제 테이블

employees

id	name	department
1	Alice	HR
2	Bob	IT
3	Carol	IT
4	David	Finance

Phantom Read

Transaction A (Repeatable Read)

START TRANSACTION;

SELECT * FROM employees WHERE department = 'IT';

-- 결과: Bob(IT), Carol(IT)

SELECT * FROM employees WHERE department = 'IT'; -- 결과: Bob(IT), Carol(IT), Eve(IT)

Transaction B (Repeatable Read)

START TRANSACTION;

INSERT INTO employees (id, name, department) VALUES (5, 'Eve', 'IT');

COMMIT;

id	name	department
1	Alice	HR
2	Bob	IT
3	Carol	IT
4	David	Finance

InnoDB

InnoDB와 Phantom READ

To prevent phantoms, InnoDB uses an algorithm called next-key locking that combines index-row locking with gap locking.

If one session has a shared or exclusive lock on record R in an index, another session cannot insert a new index record in the gap immediately before R in the index order.

InnoDB

InnoDB와 Phantom READ

“MVCC와 Next key 락을 사용하기 때문에

InnoDB에서는 Phantom READ가 발생하지 않는다.”

cf.) 표준 SQL DBMS는

record lock만 사용하므로 레코드 삽입이 가능하고

Phantom Read의 원인이 된다.

Gap Lock

Next key Lock && Gap Lock

- 인덱스와 레코드 사이의 간격을 잠금
- 또는 첫 번째 또는 마지막 인덱스 레코드 앞뒤의 범위를 잠금
- SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE;

  → t.c1 열에 15 삽입 불가능
- **Read Committed 레벨에서는 Gap Lock 이 사용되지 않음**

InnoDB

InnoDB와 Phantom READ

“Next key 락을 사용하기 때문에

InnoDB에서는 Phantom READ가 발생하지 않는다.”

단 REPEATABLE READ와, SERIALIZABLE 모드에서만.

InnoDB

REPEATABLE READ에서는 Phantom Read가 생기지 않는다?

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from emp where id >= 2;
+----+-------+-------+
| id | name  | dept  |
+----+-------+-------+
|  2 | kk    | apple |
|  3 | crong | golf  |
|  4 | pobi  | java  |
+----+-------+-------+
3 rows in set (0.00 sec)

mysql> select * from emp where id >= 2 for update;
+----+-------+-------+
| id | name  | dept  |
+----+-------+-------+
|  2 | kk    | apple |
|  3 | crong | golf  |
|  4 | pobi  | java  |
|  5 | ivy   | aos   |
+----+-------+-------+
4 rows in set (0.00 sec)

mysql>
```

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into emp values(5, 'ivy', 'aos');
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

InnoDB

InnoDB와 Phantom READ

REPEATABLE READ는 정말 안전한 거겠죠?

SELECT → INSERT → SELECT FOR UPDATE 를 하면 유령이 생깁니다!!

이럴수가!!

InnoDB

REPEATABLE READ에서는 Phantom Read가 생기지 않는다???? (2)

```
mysql> select * from emp;
+----+-------+-------+
| id | name  | dept  |
+----+-------+-------+
|  1 | honux | it    |
|  2 | kk    | apple |
|  3 | crong | golf  |
|  4 | pobi  | java  |
+----+-------+-------+
4 rows in set (0.00 sec)

mysql> update emp set dept= 'aos' where name = 'ivy';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from emp;
+----+-------+-------+
| id | name  | dept  |
+----+-------+-------+
|  1 | honux | it    |
|  2 | kk    | apple |
|  3 | crong | golf  |
|  4 | pobi  | java  |
|  5 | ivy   | aos   |
+----+-------+-------+
5 rows in set (0.00 sec)
```

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into emp values (5, 'ivy', 'ios');
Query OK, 1 row affected (0.00 sec)

mysql> select * from emp;
+----+-------+-------+
| id | name  | dept  |
+----+-------+-------+
|  1 | honux | it    |
|  2 | kk    | apple |
|  3 | crong | golf  |
|  4 | pobi  | java  |
|  5 | ivy   | ios   |
+----+-------+-------+
5 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

InnoDB

InnoDB와 Phantom READ

REPEATABLE READ는 정말 안전한 거겠죠?

억지로 만들려고 마음만 먹으면 유령도 만들수 집니다!

모든 일은 마음먹기에 달려 있다 (아님)

InnoDB

InnoDB와 Phantom READ

“MVCC와 Next-key 락을 사용하기 때문에

InnoDB에서는 Phantom READ가 발생하지 않는다.”

단 REPEATABLE READ와, SERIALIZABLE 모드에서만.

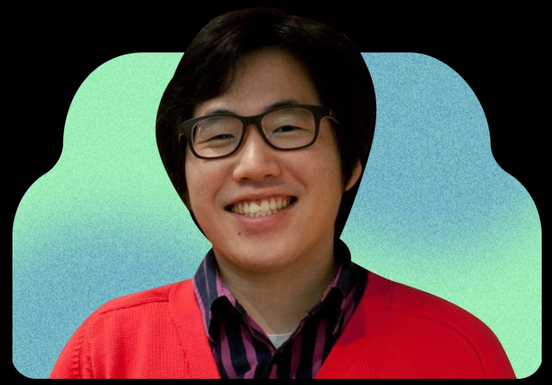
그런데 SELECT FOR UPDATE 등을 쓰다 보면 유령이 나타날 수 있다!

InnoDB

MySQL의 유령

유령은 InnoDB의 아주 깊은 곳에 슬며시 살고 있었다...

Thank You



정호영

백엔드 마스터 / 코드스쿼드

honux77@gmail.com