MySQL 5.7에서 8.0 이관하기

1. 왜 MySQL 8.0으로 마이그레이션을 해야 하는가?

- MySQL 5.7은 2023년 10월부로 EOL 보안 패치와 버그 수정을 종료했다.
  - EOL : 알려진 / 미래 취약점에 방치된 상태

따라서 MySQL 8.0으로의 마이그레이션은 선택이 아닌 필수가 되었는데,
지원이 끝난 “DB 버전”은 명확한 마이너스 포인트이기 때문이다.

✍️ MySQL 5.7과 8.0의 차이점

- 기존의 MySQL 5.7은 MySQL 고유 문법을 사용하여 표준 SQL과는 거리가 멀었다.
  - 표준 SQL 정의 “ 관계형 데이터베이스에서 데이터를 정의(DDL), 조작(DML), 질의(Query)하기 위한 언어의 공통 규약”
  - 모든 DB가 공통으로 이해할 수 있는 기본 문법과 의미 체계인데, MySQL은 표준보단 실용을 우선시 함
  - 표준 SQL를 따르는 이유는 호환성과 이식성 때문임 → “DB가 다르더라도 같은 쿼리가 돌아가야 한다”
    - 표준의 존재 이유임.
  - **=> 그래서 MySQL 8.0으로 버전 업데이트 하면서 표준 SQL을 지원함**

기능	표준 SQL 방식	MySQL 5.7 고유 문법
문자열 결합	`CONCAT(a, b)` 또는 `a	
AUTO_INCREMENT	`GENERATED ALWAYS AS IDENTITY`	`AUTO_INCREMENT`
날짜 차이 계산	`a - b` (표준)	`DATEDIFF(a, b)`

Boolean 타입	`BOOLEAN`	실제는 `TINYINT(1)`
LIMIT	`FETCH FIRST N ROWS`	`LIMIT N`

- MySQL 5.7은 보안이 비교적 느슨했다.
  - Password
  - 기본 계정이 root 접근
  - 암호화 비활성화 상태 등 ...
  - **⇒ MySQL 8.0으로 업데이트하면서 모든 연결/저장을 기본적으로 암호화하는 구조로 변경됨**

2. 이관하기 전에 점검 할 것

1) 사전 데이터 백업 / 복구 테스트

> 이관 도중 에러 발생 시 데이터 손실 방지 및 롤백 대비를 위함
>
> → 8.0에서 데이터 딕셔너리 구조 변경으로 인해, 복구 시 5.7 포맷으로 되돌리기 어렵기 때문에 필수 과정!

☐ `mysqldump` or `mysqlpump` 로 전체 DB 백업
☐ 복구 테스트 - 백업본을 테스트 서버에 복원 검증

🪄 우리 롤백 전략에 대해 더 알아봐요~!

MySQL 5.7 에서 8.0으로 이관할때는 롤백 전략은 필수이다.

특히나 `mysqldump --all-databases` 와 같은 명령어를 사용하면 구조 + 데이터 + 트리거 + 권한까지 한번에 백업되기 때문에 가장 안전한 형태라고 할 수 있다.

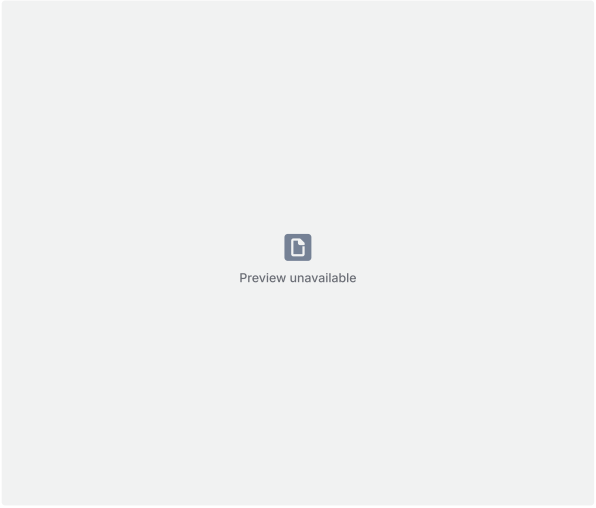
하지만..., 그만큼 단점도 명확하다.

- 데이터 정말 많다면, 덤프 생성과 복원 속도가 매우 느리다.
- 백업 중 락(Lock)이 걸린다면 서비스에 영향이 갈 수 있다.
- 바이너리 로그 기반 복원이 불가하다.

“안전한 롤백 보장”은 가능하지만, “대용량 실서비스”에는 비효율적일지도 모른다.

그렇다면, 롤백을 안전하게, 어떻게 할 수 있을까?

- Percona XtraBackup: 온라인 백업 가능 (락 거의 없음)
  - XtraBackup이 하는 일
    - MySQL이 실행 중일 때, InnoDB파일(.ibd, ibdata1) 을 복사
    - 그동안 쌓이는 redo log도 실시간으로 읽어서 함께 저장
    - 백업 후, redo log를 적용(apply)해서 데이터 정합성을 맞춤.
    - 이렇게 하면 실제 DB가 멈추지 않아도 트랜잭션 일관성이 보장된 스냅샷을 만들 수 있음.
  - InnoDB 파일 + redo log를 복제
    - **InnoDB는 트랜잭션을 빠르게 처리하기 위해 데이터를 메모리에 먼저 기록한 뒤 redo log에 "무슨 작업을 했는지"를 남긴다. 그 다음에 나중에 실제 데이터 파일 `.ibd` 에 반영(flush)를 한다.**



InnoDB 내부 구조.

- 안전하게 롤백이 되었다면, 마이그레이션 진행하기. (목차 3번에서 더 상세하게 이야기할꼐요.)
  - Percona XtraBackup을 하면서 Replication 기반 Shadow Migration을 진행
    - 기존 운영 DB(5.7)를 그대로 두고, 그 데이터를 실시간으로 복제(replication)하면서 새 버전 DB(8.0)으로 동시에 데이터를 동기화 시키는 마이그레이션 방법으로~ 마무뤼~^0^/

`mysqldump` 와 `XtraBackup` 의 차이점.

항목	`mysqldump`	`XtraBackup`
백업 방식	**논리적** (SQL로 export)	**물리적** (파일 그대로 복제)
백업 속도	느림 (INSERT문 생성)	빠름 (파일 copy 수준)
백업 시 락	필요함(락 또는 read-only)	거의 없음 (non-blocking)
데이터 크기	작을수록 유리	클수록 유리
복원 방식	SQL import	파일 복사 후 적용

2) 호환성 검사

5.7에서 사용하던 기능 중 일부는 8.0에서 제거됨

미리 확인하여 쿼리나 애플리케이션에서 에러 발생을 예방하기 위함

- ☐ 호환성 검사를 위해 `mysqlcheck` 명령어 실행 - 테이블, 컬럼, 인덱스, 스토리지 포맷, 시스템 메타데이터 등을 검사해서 8.0에서도 정상 동작 가능한지 체크해줌

```
mysqlcheck --all-databases --check-upgrade
```

- ☐ Deprecated 기능, 호환되지 않는 시스템 테이블, 컬럼 이름, 문자셋 등 확인

3) 문자셋(Character Set) 점검

MySQL 8.0은 기본 문자셋이 `utf8mb4`로 변경됨

일부 정렬(`collation`) 규칙이 달라져 인덱스 정렬 순서나 비교 결과가 달라질 수 있음

- ☐ `character_set_server`, `collation_server` 설정 확인
- ☐ `utf8` → `utf8mb4` 자동 변환 여부 확인 : DB/테이블 문자셋 확인 (`SHOW CREATE TABLE`)
- ☐ 스키마/테이블별 문자셋 확인

4) 인증 플러그인 변경 점검

기존 애플리케이션, JDBC, Python 등 클라이언트 드라이버가 구버전일 경우 로그인 실패 발생 가능

- ☐ JDBC 드라이버 버전 확인 : `mysql-connector-j 8.x` 이상 사용
- ☐ 현재 사용자(`user`, `host`)의 `plugin` 컬럼 확인

```
SELECT user, host, plugin FROM mysql.user;
```

- 8.0 기본 인증 플러그인 : `caching_sha2_password`
- 5.7 : `mysql_native_password`
- 필요 시 8.0에서도 기존 플러그인 유지 가능

```
ALTER USER 'user'@'host' IDENTIFIED WITH mysql_native_password BY 'password';
```

5) SQL 모드 (sql_mode) 확인

GROUP BY나 NULL 처리 방식이 달라져 기존 쿼리 호환성 문제 발생 가능

특히, `ONLY_FULL_GROUP_BY`로 인한 집계 쿼리 오류가 자주 발생

☐ `ONLY_FULL_GROUP_BY` 활성화 여부 확인
☐ 현재 SQL 모드 확인

```
SELECT @@GLOBAL.sql_mode;
```

6) 예약어(Reserved Words) 충돌

8.0의 윈도우 함수 추가로 기존 컬럼명이 예약어와 충돌 시 Syntax Error 발생

☐ `RANK`, `SYSTEM`, `CUME_DIST`, `WINDOW` 등 8.0에 새로 추가된 예약어가 컬럼명/테이블명으로 쓰였는지 확인

7) 스토리지 엔진 및 인덱스 점검

MyISAM은 8.0에서 기능이 제한적이고, InnoDB가 기본

FULLTEXT 인덱스의 동작 알고리즘이 개선되어 검색 결과가 달라질 수 있음

☐ `MyISAM` 테이블 존재 여부 확인
☐ `FULLTEXT`, `SPATIAL` 인덱스 정상 지원 여부
☐ `innodb_file_per_table`, `innodb_page_size` 설정 확인

8) Deprecated / 제거된 기능 확인

기존 코드가 아래 기능을 사용 중이면 런타임 오류 발생 가능

☐ `query_cache` (삭제됨)
☐ `password()` 함수 (삭제됨)
☐ `INFORMATION_SCHEMA`의 일부 뷰 변경 (`SHOW TABLES` 결과 차이)
☐ `datetime`의 0000-00-00 허용 기본값 변경 (`NO_ZERO_DATE` 기본 포함)

9) 애플리케이션 / JDBC 드라이버 호환성

5.x 드라이버는 `caching_sha2_password` 인증 미지원

SSL/TLS, timezone 설정 관련 파라미터 호환성도 달라졌음

☐ JDBC 버전은 `mysql-connector-j 8.x` 이상으로 업데이트
☐ HikariCP, Spring Boot 등 프레임워크 호환 여부 확인

10) 설정 파일(my.cnf) 검토

제거된 설정이 있으면 mysqld 시작 실패

신규 파라미터 활용 시 성능 개선 가능

☐ 5.7에서 사용하던 파라미터 중 8.0에서 제거된 항목 확인 및 삭제
☐ 8.0 신규 파라미터 (`innodb_dedicated_server` , `transaction_write_set_extraction` 등) 검토

11) 시간대(Timezone) 및 정렬 순서(Collation) 점검

> 8.0에서는 `utf8mb4_0900_ai_ci` 가 기본 정렬로,
>
> 기존 `utf8_general_ci` 와 정렬 결과가 다를 수 있음

☐ 서버 / DB / 세션의 `@@time_zone` 확인
☐ 정렬(`collation`) 변경으로 문자열 비교 결과 확인

12) 모니터링 및 로그 설정

> 업그레이드 후 초기 부하나 쿼리 오류 감지 용이
>
> 8.0은 성능 스키마를 통한 세밀한 성능 진단이 가능

☐ `performance_schema` 활성화
☐ `slow_query_log` , `general_log` , `error_log` 설정 확인

3. 마이그레이션 방법 선택

MySQL 마이그레이션 방법

구분	방식	특징	다운타임	권장 상황
In-Place Upgrade (직접 업그레이드)	기존 데이터 디렉토리를 그대로 업그레이드	빠르고 단순하지만 되돌리기 어려움	수분~수십분	DB 서버가 1대뿐이고 데이터량이 적을 때
Logical Dump & Restore (논리 백업/복원)	`mysqldump` / `mysqlpump` 로 데이터 덤프 후 복원	완전한 백업, 가장 안전함	데이터 크기에 비례	구조 변경 많거나 클린 업그레이드 원할 때
Replication 기반 업그레이드	새 8.0 서버를 슬레이브로 붙여서 동기화 후 전환	다운타임 거의 없음	수초~분	서비스 무중단 전환 필요할 때

(이중화 마이그레이션)				

1) In-Place Upgrade (직접 업그레이드)

기존 MySQL 5.7 서버에서 바이너리 데이터를 그대로 유지한 채 8.0로 업그레이드하는 방식

장점	• 빠르고 간단 (데이터 복사 불필요) • 설정 그대로 유지 가능
단점	• 되돌리기 불가능 (8.0 데이터 딕셔너리 구조 변경) • 업그레이드 중 오류 시 복구 어려움 • 다운타임이 필요
권장 조건	• 데이터 크기가 작고 • 테스트 환경에서 충분히 검증된 경우

2) Logical Dump & Restore (논리 백업/복원)

`mysqldump` 또는 `mysqlpump` 로 SQL 문 형태로 데이터를 내보내고,

새로운 8.0 서버에 깨끗이 다시 삽입하는 방식

장점	• 가장 안전 (백업/복원 구조가 명확) • 5.7 → 8.0 간 호환성 자동 변환 • 불필요한 메타데이터 정리 가능
단점	• 데이터가 많을수록 시간이 오래 걸림 • 완전한 다운타임 필요 (서비스 중지 후 복원)
권장 조건	• 데이터 10~50GB 이하 • 클린 마이그레이션 선호 • 새로운 서버로 교체 예정

3) Replication 기반 업그레이드 (이중화 마이그레이션)

MySQL의 복제(Replication) 기능을 이용해

5.7 → 8.0 서버 간 실시간 동기화 후, 장애 없이 전환하는 방식

장점	• 다운타임 최소화 (수초 수준) • 검증 후 안전하게 스위칭 가능 • 대용량 DB에서도 실무에서 자주 사용
단점	• 구성 복잡 (binary log, GTID 등 설정 필요) • 서버 2대 이상 필요 • 복제 동기화 중 트랜잭션 차이 주의
권장 조건	• 운영 중단이 거의 불가능한 서비스 • 복제 환경에 익숙한 운영자

선택 가이드

항목	In-Place	Dump & Restore	Replication
데이터 크기	소량 (≤ 10GB)	중간 (≤ 50GB)	대형 (수백 GB~TB)
다운타임 허용	있음	김	거의 없음
안전성	중간	높음	높음
복구 용이성	낮음	높음	높음
서버 수	1대	2대	2대 이상
구조 변경 허용	낮음	높음	중간

현실적인 조합 (추천 시나리오)

환경	권장 방법
개발/테스트 환경	In-Place Upgrade
데이터 정리·최적화 필요	Logical Dump & Restore
운영 서비스(무중단 전환)	Replication 기반 업그레이드
대용량 + 다운타임 최소화	Replication + Failover 전환

4. 마이그레이션 후 점검 방법

1) 기본 동작 점검

☐ 서버 기동 확인

```
systemctl status mysqld
```

☐ MySQL 버전 확인

```
SELECT VERSION();
```

☐ 데이터베이스 목록 및 용량 비교

```
SELECT table_schema, ROUND(SUM(data_length + index_length)/1024/1024,2)
FROM information_schema.tables
GROUP BY table_schema;
```

☐ 시스템 테이블 확인

```
SHOW TABLES FROM mysql;
```

2) 데이터 무결성 / 일관성 점검

☐ 테이블 수 일치

```
SELECT COUNT(*) FROM information_schema.tables WHERE table_schema NOT IN
```

☐ 데이터 행 수 비교

```
SELECT COUNT(*) FROM your_table;
```

☐ CHECKSUM 비교 (정확 검증)

```
CHECKSUM TABLE your_table;
```

☐ 트리거 / 뷰 / 프로시저 / 이벤트 확인

```
SHOW TRIGGERS;
SHOW PROCEDURE STATUS;
SHOW EVENTS;
```

3) 애플리케이션 연결 및 인증 점검

☐ JDBC 등 클라이언트 연결 테스트

```
SELECT user, host, plugin FROM mysql.user;
```

☐ 커넥션 풀 / ORM 호환성 테스트

☐ 권한(Role) 및 GRANT 검증

```
SHOW GRANTS FOR 'app_user'@'%';
```

4) 쿼리/성능/SQL 모드 점검

☐ SQL 모드 확인

```
SELECT @@GLOBAL.sql_mode;
```

☐ 주요 쿼리 성능 비교

```
EXPLAIN SELECT * FROM orders WHERE user_id = 100;
```

☐ 인덱스 통계 갱신

```
ANALYZE TABLE your_table;
OPTIMIZE TABLE your_table;
```

☐ 정렬 및 비교 결과 확인

5) 로그 및 모니터링 점검

☐ 에러 로그 확인

```
cat /var/log/mysql/error.log | grep -i "error"
```

☐ Slow Query Log 활성화

```
SET GLOBAL slow_query_log = 'ON';
SET GLOBAL long_query_time = 1;
```

☐ Performance Schema / Sys Schema 점검

```
SELECT * FROM performance_schema.setup_instruments LIMIT 3;
SELECT * FROM sys.schema_table_statistics LIMIT 3;
```

☐ 모니터링 툴 연결 확인

5. 주의사항

스프링

- MySQL 8.0을 사용하려면 mariadb 드라이버를 사용하면 안됨!
  - MariaDB Connector/J는 MySQL 8.0을 위한 완전한 호환성을 제공하지 않는다.

- 기존 버전

```
// Gradle
implementation('org.mariadb.jdbc:mariadb-java-client:2.3.0')


// Spring application.yml
spring:
  datasource:
    url: jdbc:mysql://mydb:3306/appdb?useSSL=false&allowPublicKeyRetrie
    username: app
    password: ****
    driver-class-name: org.mariadb.jdbc.Driver
```

- 수정 버전

```
// Gradle
implementation 'mysql:mysql-connector-j:8.3.0'  // 최신 버전
implementation 'mysql:mysql-connector-j:8.0.33' // LTS 계열



// Spring application.yml
```

```yaml
spring:
  datasource:
    url: jdbc:mysql://mydb:3306/appdb?useSSL=false&allowPublicKeyRetrie
    username: app
    password: ****
    driver-class-name: com.mysql.cj.jdbc.Driver
```

> driver-class-name를 생략해도 될까요 ?

생략해도 괜찮은게, 스프링 부트가 JDBC URL → Driver 자동 매핑 기능이 동작한다.

근데 근데 명시해야 할 경우가 있는데, 아래와 같은 환경일 때는 명시가 필요하다...

① MariaDB 드라이버와 MySQL 드라이버가 혼재된 환경

- 이런 경우 자동 인식이 헷갈릴 수 있어 명시하는 게 안정적이다.

② 특수 커넥션 풀 라이브러리(Hikari 외)

- 예: 일부 레거시 톰캣 DBCP에서 자동 인식이 안 될 때.

③ 다중 Datasource 환경(멀티 DB)

- 두 개 이상의 데이터소스를 구성할 때는 driver-class-name을 명확히 지정함.

MySQL

- Replication이 정상적으로 따라잡고 있고, Replica(8.0)의 지연이 **수 밀리초 수준**일 때 Read DB로 사용한다.
- MySQL 8.0 업데이트 전, 새로 만든 MySQL 5.7 서버를 종료해야 한다.
  - 5.7이 실행 중인 상태에서는 8.0 binary로 덮어씌울 수 없다.

6. SpringBoot 코드 수정

- [x] Gradle 수정
- [x] application.yml datasource 드라이버 수정
- [ ] 네이티브 쿼리 전체적으로 확인 (QueryDSL은 크게 문제가 없다고 함)
- [ ] zero date 처리 확인 : MySQL 5.x에서는 '0000-00-00 00:00:00' 허용했으나 8.x에서는 기본적으로 예외 발생

```
// 대응방법 1: URL 파라미터 추가
url: jdbc:mysql://localhost:3306/dbname?zeroDateTimeBehavior=convertToNu

// 대응방법 2: Entity에서 nullable 처리
```

```
@Column(nullable = true)
private LocalDateTime createdAt;
```

- [ ] GROUP BY 절에 SELECT 컬럼이 모두 포함되어 있는지 (ONLY_FULL_GROUP_BY 모드 위반 때문)
- [ ] MySQL 8.0에 새로 추가된 예약어 사용 중인지 확인

```
예약어 체크 (MySQL 8.0에서 추가된 예약어)
-- RANK, DENSE_RANK, ROW_NUMBER, LEAD, LAG 등
-- 테이블명이나 컬럼명으로 사용 시 백틱(`) 처리 필요
```

- [ ] 인증 방식 변경 확인

```
[버전별 인증 방식]
MySQL 5.7 이하: mysql_native_password (SHA1 기반)
MySQL 8.0 이상: caching_sha2_password (SHA256 기반, 더 안전)

[문제점]
MySQL 8.0 Driver는 caching_sha2_password를 지원
MySQL 5.4 서버는 caching_sha2_password를 지원하지 않음

[추천 해결 방법]
-> JDBC URL에 인증 방식 명시 (가장 안전)
properties# MySQL 5.4 서버에 연결 시 (현재 상황)
spring.datasource.url=jdbc:mysql://localhost:3306/dbname?\
  useSSL=false&\
  allowPublicKeyRetrieval=true&\
  serverTimezone=Asia/Seoul&\
  defaultAuthenticationPlugin=mysql_native_password
```

7. docker-compose 테스트

수정 전 코드

- 보안 때문에 container_name, environment 값 바꿔서 작성했습니다! 🤫

```yaml
version: '3.6'
services:
    mysql:
        container_name: test-local-mysql
        image: mysql:5.7
        platform: linux/amd64
        volumes:
            - ./my.cnf:/etc/mysql/my.cnf
        environment:
            - MYSQL_ROOT_PASSWORD=1234test
            - MYSQL_DATABASE=test
            - MYSQL_USER=local
            - MYSQL_PASSWORD=local
        ports:
            - 3306:3306
        command: ['mysqld', '--character-set-server=utf8mb4', '--collat
```

수정 후 코드

- 변경된 부분
  - image
  - volumes
  - command

```yaml
version: '3.6'
```

```yaml
services:
    mysql:
        container_name: test-local-mysql
        image: mysql:8.0.33
        platform: linux/amd64
        volumes:
            - ./my80.cnf:/etc/mysql/conf.d/my80.cnf
        environment:
            MYSQL_ROOT_PASSWORD: 1234test
            MYSQL_DATABASE: test
            MYSQL_USER: local
            MYSQL_PASSWORD: local
        ports:
            - "3306:3306"
        command:
            - --default-authentication-plugin=mysql_native_password
            - --character-set-server=utf8mb4
            - --collation-server=utf8mb4_0900_ai_ci
            - --lower_case_table_names=1
            - --explicit_defaults_for_timestamp=true
```

- infra 디렉토리에 my80.cnf 파일 생성

```
[mysqld]
default_authentication_plugin=mysql_native_password
character-set-server=utf8mb4
collation-server=utf8mb4_0900_ai_ci
lower_case_table_names=1
```

기존 my.cnf 파일을 8.0에서 사용하면 안 되는 이유

- my.cnf는 MySQL의 전체 설정 파일
- MySQL 서버가 실행될 때 읽어들이는 모든 설정이 담긴 파일 (MySQL이 어떻게 실행될지 결정)
  - character-set 설정
  - collation 설정
  - log 관련 설정
  - deprecated 옵션
  - 인증 플러그인
  - 정렬 규칙
  - 인덱스 동작 관련 설정
  - 등등

- 5.7 옵션의 상당수가 8.0에서 삭제되어 5.7 옵션이 남아있으면 에러 발생
- 8.0 기본값과 충돌하는 설정이 포함될 수 있음

→ 따라서, 새로운(my80.cnf) 파일을 만들어야 함

→ my.cnf 파일에 덮어씌워도 되지만 문제가 발생했을 때 복구가 어렵고 설정 충돌이 날 수 있음

command 설정

```yaml
command:
  - --default-authentication-plugin=mysql_native_password
  - --character-set-server=utf8mb4
```

```
  - --collation-server=utf8mb4_0900_ai_ci
  - --lower_case_table_names=1
  - --explicit_defaults_for_timestamp=true
```

- `--default-authentication-plugin=mysql_native_password`
  - 8.0 기본 인증 플러그인 : caching_sha2_password
  - 위 플러그인은 SpringBoot + MySQL 드라이버 조합과 안 맞아서 Connection 실패가 많이 발생함
  - `mysql_native_password`를 강제로 적용하면 SpringBoot가 안정적으로 접속할 수 있음
  - MySQL 8.0 전환 시 필수 옵션

- `--character-set-server=utf8mb4`
  - 명시적으로 utf8mb4 설정을 해주어서 깨짐 이슈 방지

- `--collation-server=utf8mb4_0900_ai_ci`
  - 8.0에서 디폴트 collation이 변경됨
  - 이를 명시적으로 설정

- `--lower_case_table_names=1`
  - macOS 파일 시스템(HFS+, APFS)은 기본적으로 대소문자 구분 안함
  - MySQL은 파일 시스템에 테이블을 파일로 저장하기 때문에 8.0에서 `lower_case_table_names=0` (대소문자 구분)으로 실행 시
    macOS에서 테이블명 충돌로 아예 MySQL 실행이 안 될 수 있음
  - macOS에서 안전하게 실행하기 위함

- `--explicit_defaults_for_timestamp=true`
  - MySQL 5.7 → 8.0에서 TIMESTAMP 기본값 처리 방식이 변경됨
  - 명시적으로 “timestamp 기본값 자동 정의 허용” 옵션을 넣어 안정적으로 실행하도록 함

이슈

아래 에러가 발생할 경우, volume mount 확인 후 삭제하고 재기동하면 됨

```
Upgrade is not supported after a crash or shutdown with innodb_fast_shut
This redo log was created with MySQL 5.7.44
DD Storage Engine failed
Data Dictionary initialization failed
```