[12주차] 16.7 ~ 17전

태그	Done
날짜	@2024년 4월 16일 → 2024년 4월 23일
제목	복제

📌 16장 복제

✅ 16.7 복제 고급 설정

16.7.1 지연된 복제(Delayed Replication)

복제의 목적은 최대한 빠르게 동기화해서 소스, 레플리카 서버간 데이터를 동일한 상태로 만드는 목적도 있지만, 의도적으로 레플리카 서버로의 복제를 지연시키는 경우도 있습니다.

- 소스 서버에서 실수로 삭제 쿼리를 했을때 복구해야 하는 경우
- 데이터 반영에 지연이 있을 때 어떻게 서비스가 동작하는지 테스트 할 경우

5.6에 처음 등장하고 8.0버전에 와서 여러 부분들이 개선됐습니다.

`CHANGE MASTER TO MASTER_DELAY=86400` 은 레플리카 서버의 복제를 하루 늦추는 명령어 입니다.

레플리카 서버는 SOURCE_DELAY(or MASTER_DELAY) 옵션에 값이 지정되면 타임스탬프 값을 참조해서 각 트랜잭션별로 실행을 지연시킬지 말지 여부를 결정하는데 이때 `ICT`, `OCT`라는 타임스탬프 값을 참조합니다.

- original_commit_timestamp(OCT)

  트랜잭션이 원본 소스 서버(Original Source)에서 커밋된 시각으로, 밀리초 단위의 유닉스 타임스탬프 값으로 저장된다.

- immediate_commit_timestamp(ICT)

  트랜잭션이 직계 소스 서버(Immediate Source)에서 커밋된 시각으로, 밀리초 단위의 유닉스 타임스탬프 값으로 저장된다.

- `원본 소스`: 가장 위에 존재하는 소스 서버, 트랜잭션이 제일 처음 실행됐던 소스 서버를 말함
- `직계 소스`: 가장 하위에 있는 레플리카 서버 기준 바로 위의 소스 서버를 말함
- 원본 소스는 OCT, ICT값이 일치하고, 바이너리 로그를 사용하고 `log_slave_updates`가 활성화된 레플리카 서버의 경우 ICT에는 복제된 트랜잭션이 커밋된 시점으로 값이 저장됨

기존의 문제였던 원본 소스 서버 시각 기준 지연 계산은 ICT를 사용함에 따라 문제점이 사라졌습니다.

16.7.2 멀티 스레드 복제(Multi-threaded Replication)

복제된 트랜잭션들을 하나의 스레드가 아닌 여러 스레드로 처리할 수 있게하는 멀티 스레드 복제 기능을 제공합니다.

5.6에 처음 도입됐으며 다음과 같은 형태로 처리됩니다.

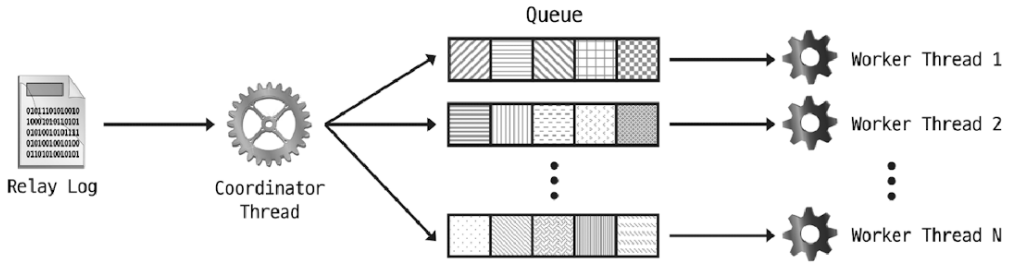

그림 16.23 멀티 스레드 복제 동기화 방식

- 기존 SQL 스레드는 코디네이터 스레드로 생각하면 됩니다. 코디네이터 스레드는 실제로 이벤트를 실행하는 워커 스레드와 협업해 동기화를 진행합니다.

- 코디네이터 스레드는 릴레이 로그 파일에서 이벤트를 읽고 설정 방식에 따라 스케줄링 하여 워커 스레드에게 이벤트를 할당합니다.
- 이후 워커 스레드의 큐에 적재되면 워커 스레드가 큐에서 이벤트를 하나씩 꺼내어 레플리카 서버에 적용합니다.

어떻게 병렬 처리할 것인가에 따라 `DB기반`, `LOGICAL CLOCK 기반 처리 방식`으로 나뉩니다. 또한 시스템 변수를 통해 워커 스레드의 수를 설정하고, 처리할 수 있는 이벤트 사이즈를 조절할 수 있습니다.

만약 이벤트 하나의 크기가 설정 값을 넘어선다면 큐가 비워질 때까지 대기 후 해당 이벤트를 처리하므로 주의해야 합니다.

---

1. 데이터베이스 기반 멀티 스레드 복제

스키마 기반 처리 방식이라고도 합니다. 한마디로 mysql내 데이터베이스 단위로 병렬 처리를 수행하는 방식을 말합니다. (따라서 한개의 DB밖에 없다면 장점이 없음)

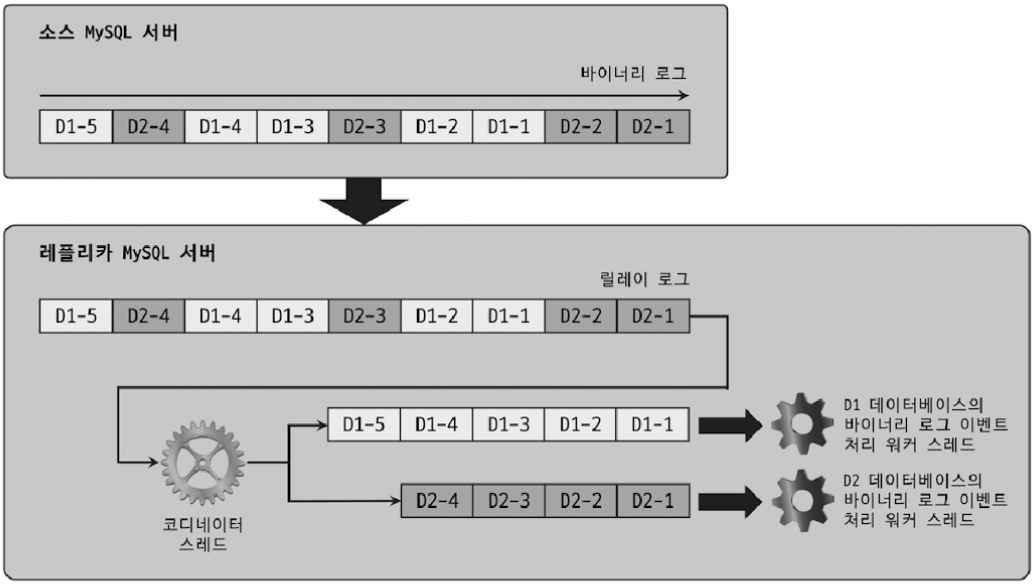

그림 16.24 데이터베이스 기반 멀티 스레드 복제의 동작 방식

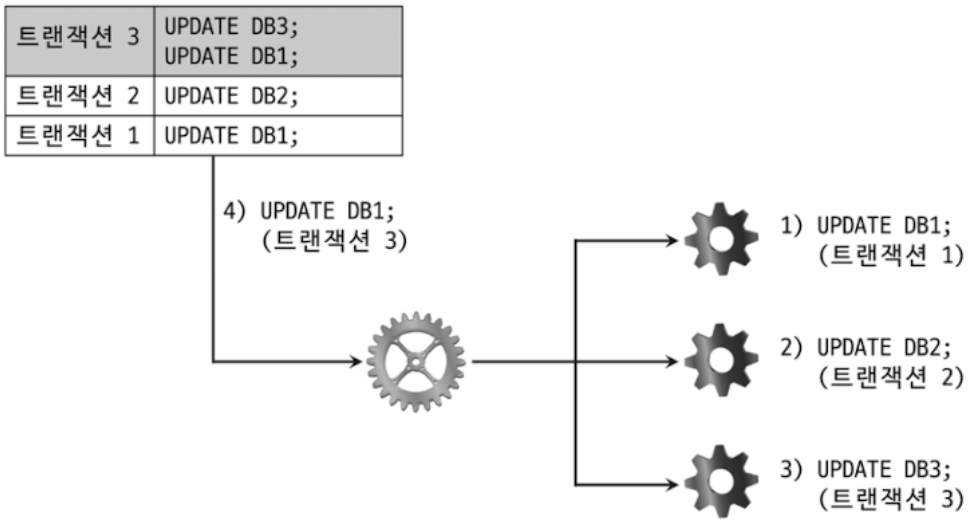

그림 16.25 데이터베이스 기반 멀티 스레드 복제의 예시

- 다만 테이블, 레코드 수준의 충돌까지 고려하지 않으므로 DB가 동일한지 아닌지만 비교하여 병렬 처리를 수행합니다.
- 즉, 이미 DB1이 처리중일때 새로운 DB1에 대한 트랜잭션이 들어오면 기존 DB1 트랜잭션이 끝나길 기다린 후 처리됩니다.
- 또한 log_slave_updates 옵션이 활성화돼 있을때 소스 서버의 바이너리 로그 트랜잭션 순서와 레플리카 서버의 바이너리 로그 트랜잭션 순서는 위와 같은 특성때문에 다를 수 있습니다.
- 따라서 소스 서버의 특정 트랜잭션 이전에 실행된 모든 트랜잭션이 레플리카 서버에 전부 적용됐다고 보장할 순 없습니다.

---

2. **LOGICAL CLOCK 기반 멀티 스레드 복제**

소스 서버에서 넘어온 전체 트랜잭션들을 데이터베이스에 종속되지 않고 멀티 스레드로 처리할 수 있습니다. (데이터베이스 내에서 멀티 스레드 동기화 처리가 가능해짐)

- LOGICAL CLOCK은 발생 순서만 파악할 수 있고 나열된 시각들간 소요 시간을 알 수 없습니다. 멀티 스레드 혹은 분산 시스템에서 상태의 동기화에 많이 사용됩니다.

트랜잭션의 순번 값을 바탕으로 병렬 처리 여부를 판단하는 Commit-parent, 개선된 잠금(Lock)기반, 8.0에서 도입된 WriteSet 기반 방식이 도입됐습니다.

(Commit-parent, LOCK 기반은 생략)

2-1 WriterSet 기반 LOGICAL CLOCK 방식

8.0.1 버전에 도입된 방식, 트랜잭션이 변경한 데이터를 기준으로 병렬 처리 가능여부를 결정합니다.

```
Tx1 : ---P-------C---------------->
Tx2 : --------------P-------C----->
* P = Prepare
* C = Commit
```

Commit-parent, LOCK 기반은 위와같이 트랜잭션 시점이 다르다면 병렬 실행이 불가능합니다. 하지만, WriteSet 기반 방식은 동일한 데이터를 변경하지 않는 트랜잭션들은 시점이 달라도 모두 병렬로 실행될 수 있습니다.

같은 세션에서 실행된 트랜잭션들의 병렬 처리 여부에 따라 다음과 같이 나뉩니다.

- `COMMIT_ORDER`
  - 과거 5.7버전의 잠금 기반 방식과 동일하게 동작, 커밋 시점이 겹치는 트랜잭션들은 모두 병렬 처리됨
- `WRITESET`
  - 서로 다른 데이터들을 변경한 트랜잭션들은 모두 병렬로 처리됨
- `WRITESET_SESSION`
  - 동일한 세션에서 실행된 트랜잭션들은 병렬로 처리될 수 없고 그 외에는 WRITESET과 동일함

트랜잭션에서 변경 데이터를 기준으로 병렬 처리를 위한 트랜잭션들의 종속 관계를 정의합니다.

이를 구현하기위해 내부적으로 트랜잭션에 의해 변경된 데이터들의 목록을 해시값으로 표현하고 관리합니다. `이런 해싱된 데이터를 WriteSet이라고 합니다`.

- 아래와 같은 조합으로 WriteSet이 생성됨(유니크한 키의 개수만큼 만들어짐)

```
WriteSet = hash(index_name, db_name, db_name_length, table_name, table_name_length, value, value_length)
```

1. 트랜잭션들의 WriteSet은 MySQL 서버 메모리에서 해시맵 테이블로 히스토리가 관리됨
    - 변경된 데이터 해시값 WriteSet이 Key, 트랜잭션의 sequence_number가 Value로 저장됨
2. 트랜잭션이 커밋되면 바이너리 로그에 트랜잭션 정보와 함께 last_committed, sequence_number값이 기록됩니다.

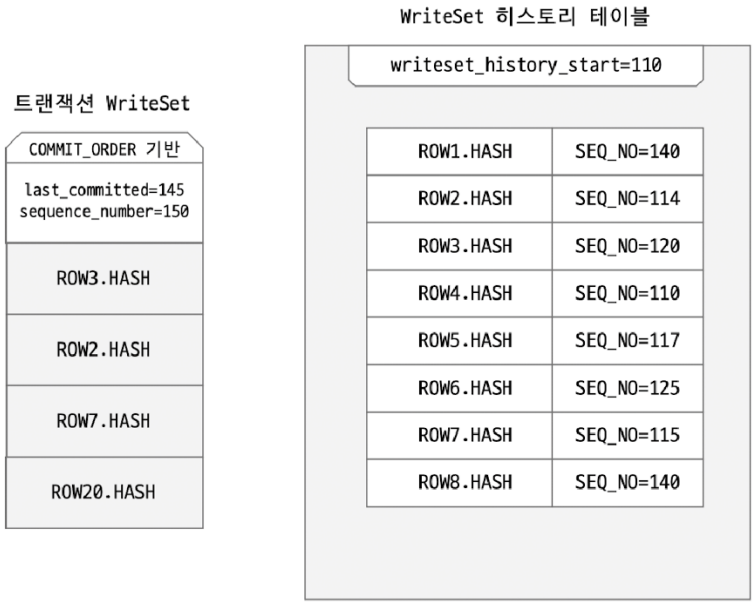

그림 16.30 COMMIT_ORDER 기반으로 설정된 트랜잭션 정보와 현재 WriteSet 히스토리 테이블 데이터

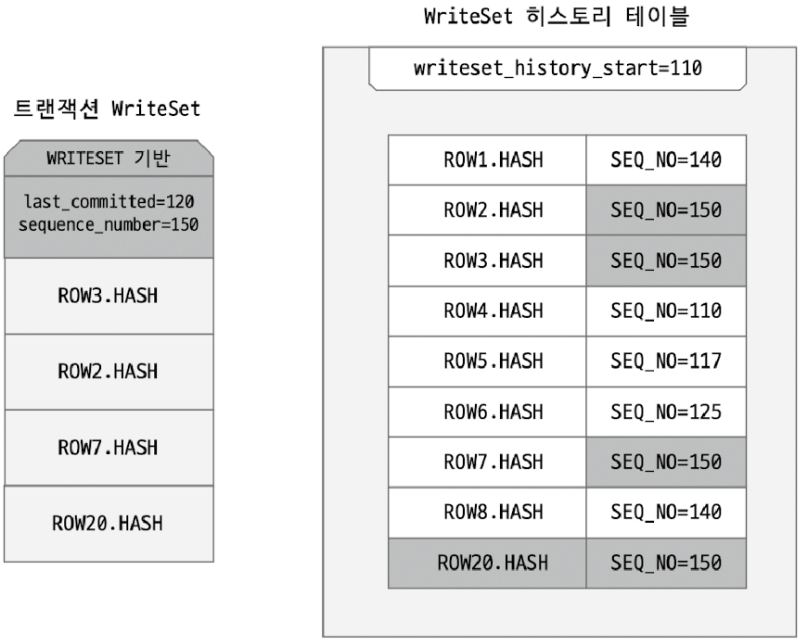

그림 16.31 WRITESET 기반으로 재설정된 트랜잭션 정보와 갱신된 WriteSet 히스토리 테이블 데이터

- 트랜잭션의 `last_committed` 값을 1차적으로 `COMMIT_ORDER` 타입 기반으로 설정함
- 이후 ROW3, ROW2, ROW7, ROW20 순서로 해시테이블에 찾아서 비교를 수행함
  - 충돌하는 KEY가 있다면 해당 KEY의 VALUE를 자신의 sequence_number로 변경
  - 기존 sequence_number를 자신의 last_committed 값으로 설정하는데 이때 기존 last_committed값이 더 크다면 변경하지 않습니다.
  - 충돌하지 않는다면 새로운 데이터를 삽입함
- 만약 여러 `WriteSet`이 충돌되고 다른 `sequence_number`를 가진다면 트랜잭션의 `last_committed`에는 이 `sequence_number` 값 중 가장 큰 값이 저장됩니다.
- `번외`
  - 충돌하는 `WriteSet`이 하나도 없다면 해당 트랜잭션이 가진 WriteSet들이 히스토리 테이블에 새로 저장됩니다. 이후 트랜잭션의 last_committed에는 히스토리 테이블에 존재하는 WriteSet 데이터 중 가장 작은 sequence_number 값이 저장됩니다.
    - `WRITE_SESSION` 타입에선 결정된 last_committed값이 있다면, 같은 세션에서 커밋된 마지막 트랜잭션의 sequence_number 값과 한번 더 비교해서 둘 중 더 큰값을 last_committed에 저장합니다.

    - `WRITESET`, `WRITESET_SESSION` 모두 변경된 데이터들이 속하는 테이블에 유니크한 키를 하나도 가지고 있지 않다면 WriteSet은 생성되지 않습니다. 이때는 last_committed에는 COMMIT_ORDER 타입 기반으로 결정된 값이 유지되어 최종 사용됩니다.
    - 변경된 데이터들이 속하는 테이블의 유니크한 키들이 다른 테이블에서 외래키로 참조되는 경우에도 COMMIT_ORDER 타입 기반으로 결정된 last_committed값을 사용합니다.

- `정리`
    - 동시에 커밋되는 트랜잭션의 수에 의존하지 않으므로 굳이 동시에 커밋되는 트랜잭션 수를 늘리기 위해 속도를 저하시킬 필요가 없습니다.
    - WriteSet 기반 방식은 레플리카 서버에서 병렬 처리성을 높이는 방식이지만, 트랜잭션 커밋 시 추가적인 메모리 공간이 필요하고 비교에 따른 오버헤드가 발생합니다.

---

3. **멀티 스레드 복제와 복제 포지션 정보**

멀티 스레드 복제에선 각 워커 스레드들이 실행한 바이너리 로그 이벤트의 포지션 정보는 특정 파일들에 각 스레드별로 저장됩니다.

워커 스레드들은 이벤트를 실행 완료할때마다 해당 데이터를 갱신합니다.

현재 복제 이벤트의 처리 현황을 보여주는 Applier 메타데이터에는 워커 스레드들이 실행한 이벤트들에서 Low Watermark에 해당하는 이벤트의 포지션 값이 저장됩니다.

이 값은 코디네이터 스레드(SQL 스레드)가 수행하는 체크포인트 작업에 의해 주기적으로 갱신됩니다.

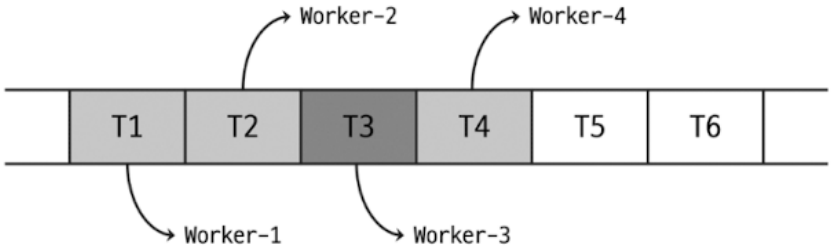

그림 16.32 트랜잭션을 처리 중인 워커 스레드

- T3가 오래 걸리는 이벤트고 나머지 이벤트들이 먼저 실행 완료된 상황에서
- 코디네이터 스레드에서 체크포인트를 수행하면 T4가 완료되어도 T3가 완료되지 않았으므로 T2가 로우 워터마크 이벤트가 됩니다.
    - Applier 메타데이터에서 소스 서버의 바이너리 로그 및 릴레이 로그 포지션 값이 T2 이벤트에 해당하는 포지션 값으로 업데이트 됩니다.
- T2, T4 사이에 생겨난 포지션 간격은 Gap(갭)이라고 합니다. 체크포인트 지점은 항상 갭 이전에 실행 완료된 이벤트에서만 나타납니다.
    - 또한 시스템 변수의 설정을 통해 애초에 Gap이 발생하지 않도록 할 수 있습니다.

16.7.3 크래시 세이프 복제(Crash-safe Replication)

복제를 했더라도, 비정상 종료 후 재실행시 소스 서버와의 동기화가 실패할 수 있습니다. 서버 장애이후 mysql 에서 문제없이 복제가 진행되는 크래시 세이프 복제를 실현할 수 있는데

이는 특정 옵션을 활성화 하는게 아니라 여러 가지 복제 관련 옵션들을 복제 형태에 따라 적절히 설정했을 때 얻게 되는 효과입니다.

---

1. 서버 장애와 복제 실패

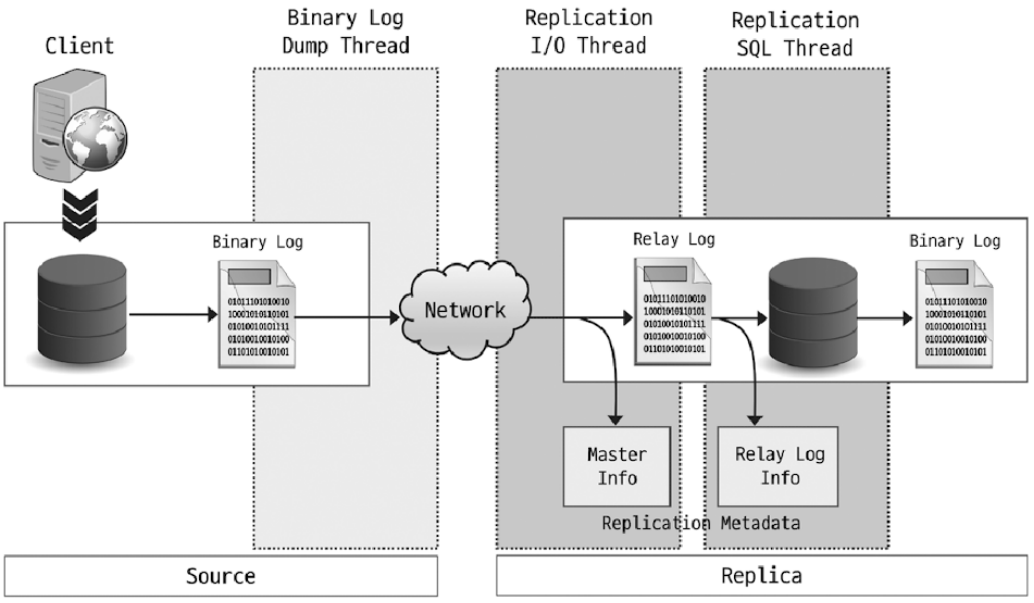

그림 16.33 MySQL의 복제 동기화 과정

I/O스레드와 SQL 스레드는 각각 자신이 어느 시점까지의 바이너리 로그를 가져왔는지, 어느 트랜잭션까지 재실행했는지에 대한 포지션 정보를 남깁니다.
(이후 재시작시 어느 시점부터 다시 시작할지 판단할때 사용함)

위와 같이 진행되는 복제 과정에서 실행 포지션 정보가 `FILE` 혹은 `TABLE` 형태로 관리되는것에 따라 문제가 발생할 수 있습니다.

- FILE 형태의 경우 각 스레드가 동작할 때 실제 자신이 처리 중인 내용과 포지션 정보를 원자적으로 동기화된 상태로 관리할 수 없습니다.
  - mysql 서버가 비정상 종료하는 경우 처리한 내역과 포지션 정보 간에 불일치가 발생할 수 있습니다.

    - I/O 스레드가 릴레이 로그에 이벤트를 기록한 후 아직 포지션 정보 파일에 업데이트를 하지 않은 상태에서 MySQL이 비정상 종료되면 MySQL을 재구동할 때 릴레이 로그에 동일한 이벤트가 기록될 수 있다.
    - SQL 스레드가 릴레이 로그에 기록된 트랜잭션을 커밋한 후 아직 포지션 정보 파일에 업데이트를 하지 않은 상태에서 MySQL이 비정상 종료되면 MySQL을 재구동할 때 동일한 트랜잭션이 재실행될 수 있다.

- TABLE 형태는 InnoDB 엔진을 사용하므로 SQL 스레드가 트랜잭션 적용과 포지션 정보 업데이트를 한 트랜잭션으로 묶어 원자적으로 처리할 수 있게 됐습니다.
- 그래도 여전히 I/O스레드가 릴레이 로그 파일에 이벤트를 쓰는 작업, 포지션 정보 업데이트 작업이 원자적으로 처리되지 않으므로 여전히 불일치 문제가 발생할 가능성이 있

습니다.

```
relay_log_recovery=ON
relay_log_info_repository=TABLE
```

- 다만 위의 `최소 옵션`을 통해 I/O 스레드의 포지션을 SQL 스레드가 마지막으로 실행했던 포지션으로 초기화하고, 새로운 릴레이 로그 파일을 생성해서 SQL 스레드가 읽어야 할 릴레이 로그 포지션 위치를 초기화 합니다.
  - 다만 운영체제의 비정상 종료에는 무용지물일 수 있습니다.

---

2. 복제 사용 형태별 크래시 세이프 복제 설정 (자세한 내용은 책을 참조)

크게 4가지의 방식이 존재합니다.

1. 바이너리 로그 파일 위치 기반 복제 + 싱글 스레드 동기화
   - 최소 옵션 설정 셋과 동일
2. 바이너리 로그 파일 위치 기반 복제 + 멀티 스레드 동기화
   - 레플리카 서버에서 복제된 트랜잭션들의 커밋 순서가 소스 서버에서와 동일하도록 설정됐느지 여부에 따라 옵션 셋이 달라짐
3. GTID 기반 복제 + 싱글 스레드 동기화
   - mysql.gtid_executed 테이블 데이터가 복제된 트랜잭션이 적용될 때마다 매번 함께 갱신되는지 여부에 따라 옵션 셋이 달라집니다.
4. GTID 기반 복제 + 멀티 스레드 동기화
   - 싱글 스레드로 동기화 되는 경우와 동일하지만, 서버가 비정상적으로 종료된 후 relay_log_recovery=ON 설정으로 재구동하면 트랜잭션 갭을 메우는 작업이 수행됩니다.
   - 하지만 GTID에선 불필요하기에 자동으로 생략되도록 코드가 수정됐습니다.

16.7.4 필터링된 복제(Filtered Replication)

mysql 복제에선 소스 서버의 특정 이벤트만 레플리카 서버에 적용될 수 있도록 필터링 기능을 제공합니다.

필터링의 주체는 소스, 레플리카 서버 둘 다 될 수 있습니다.

- 소스 서버에서의 필터링은 데이터베이스 단위로만 가능함
  (설정 변경시 반드시 재시작 필요)

  - binlog-do-db
    바이너리 로그에 기록할 데이터베이스명을 지정한다. 이 옵션에 지정된 데이터베이스에 대한 이벤트들만 바이너리 로그에 기록된다.
  - binlog-ignore-db
    binlog-do-db 옵션과는 상반되는 옵션으로, 바이너리 로그에 기록하지 않을 데이터베이스명을 지정한다. 이 옵션에 지정된 데이터베이스에 대한 이벤트들은 바이너리 로그에 기록되지 않는다.

- 레플리카 서버는 소스 서버에서보다 더 유연한 형태로 필터링 설정이 가능합니다.
  - 재시작 없이 동적으로 필터링 설정을 변경할 수 있습니다.
  - 릴레이 로그에 저장된 이벤트들을 실행하는 시점에 적용됩니다.
  - 모든 이벤트들을 가져온 다음 이벤트를 실행할 때 필터링을 적용합니다.
  - `CHANGE REPLICATION FILTER` 구문에서 필터링을 적용합니다.

    표 16.5 CHANGE REPLICATION FILTER 구문에서 사용 가능한 필터링 옵션 목록

옵션	커맨드 라인 및 설정 파일 옵션	설명
REPLICATE_DO_DB	replicate-do-db	복제 대상 데이터베이스를 지정한다.
REPLICATE_IGNORE_DB	replicate-ignore-db	복제에서 제외할 데이베이스를 지정한다
REPLICATE_DO_TABLE	replicate-do-table	복제 대상 테이블을 지정한다.
REPLICATE_IGNORE_TABLE	replicate-ignore-table	복제에서 제외할 테이블을 지정한다
REPLICATE_WILD_DO_TABLE	replicate-wild-do-table	복제 대상 테이블을 와일드카드(%) 패턴을 사용해 지정한다.
REPLICATE_WILD_IGNORE_TABLE	replicate-wild-ignore-table	복제에서 제외할 테이블을 와일드카드(%) 패턴을 사용해 지정한다.
REPLICATE_REWRITE_DB	replicate-rewrite-db	특정 데이터베이스에 대한 이벤트들을 지정한 데이터베이스로 치환해서 적용한다.

  - 복제된 이벤트를 이벤트가 발생한 원래의 데이터베이스가 아닌 다른 데이터베이스에 적용하도록 설정할 수 있습니다.
- 복제 필터링이 적용된 레플리카 서버는 다음과 같이 필터링을 진행합니다.
  - 데이터베이스 수준으로 설정된 필터링 옵션들을 바탕으로 1차적으로 필터링함

    - 다음으로 테이블 수준으로 설정된 필터링 옵션들을 체크해서 최종적으로 이벤트의 적용 여부를 결정하게 됩니다.
    - 데이터베이스 수준의 필터링 옵션들은 복제되어 넘어온 이벤트의 바이너리 로그 포맷에 따라 같은 이벤트라도 필터링 처리 결과가 달라집니다.
    - `Statement 포맷`은 USE 문에 의해 지정된 디폴트 DB를 바탕으로 필터링이 적용됨
    - `Row 포맷`인 경우 DML 이벤트들은 변경된 테이블이 속한 데이터베이스를 바탕으로 필터링이 적용됩니다. 즉, 실행을 할 때 해당 테이블이 속한 DB를 바탕으로 필터링을 합니다. (DDL은 Statement이므로 Statement 포맷과 동일)
- 테이블 수준 필터링 옵션또한 복제 대상 테이블과 복제 제외 대상 테이블을 함께 변경하는 형태의 쿼리의 경우 바이너리 로그 포맷에 따라 필터링 결과가 달라질 수 있으므로 주의해야 합니다.

    - ROW 포맷 사용 시

      DDL문에 대해 USE 문을 사용해 디폴트 데이터베이스가 설정되게 하고 쿼리에서 데이터베이스명을 지정하지 않는다.

    - STATEMENT 또는 MIXED 포맷 사용 시

      DML 및 DDL문 모두 USE 문을 사용해 디폴트 데이터베이스가 설정되게 하고 쿼리에서 데이터베이스명을 지정하지 않는다. 또한 복제 대상 테이블과 복제 제외 대상 테이블을 모두 변경하는 DML을 사용하지 않는다.