[11주차] 16.4 ~ 16.6

태그	Done
날짜	@2024년 4월 9일 → 2024년 4월 16일
제목	복제

📌 16장 복제

✅ 16.4 복제 데이터 포맷

레플리카 서버는 소스 서버의 바이너리 로그 이벤트를 내부적 가공 없이 그대로 실행해 자신의 데이터에 적용하기에 복제시 어떤 데이터 포맷을 사용하는지는 중요합니다.

크게 `Statement`, `Row`, `Mixed`가 존재합니다. 이는 `binlog_format` 시스템 변수를 통해 설정할 수 있습니다.

16.4.1 Statement 기반 바이너리 로그 포맷

- 변경 이벤트에 대해 이벤트를 발생시킨 SQL 문을 바이너리 로그에 기록하는 방식입니다.
- 즉, 실행된 SQL문이 그대로 바이너리 로그에 저장돼 있습니다.

- 장점
  - 여러 레코드를 수정해도 SQL문 하나만 기록되기에, 용량이 적습니다.
  - 원격으로 바이너리 로그를 백업하거나 원격 레플리카 서버에 복제할 때도 좀 더 빠르게 처리됨
  - 감사등의 목적으로 확인할때 SQL문을 확인할 수 있어 용이함
- 단점
  - 비확정적(Non-Deterministic)으로 처리될 수 있는 쿼리의 경우 소스 - 레플리카 간 데이터가 달라질 수 있음

    - DELETE/UPDATE 쿼리에서 ORDER BY 절 없이 LIMIT 사용
    - SELECT ... FOR UPDATE 및 SELECT ... FOR SHARE 쿼리에서 NOWAIT이나 SKIP LOCKED 옵션 사용
    - LOAD_FILE(), UUID(), UUID_SHORT(), USER(), FOUND_ROWS(), RAND(), VERSION() 등과 같은 함수를 사용하는 쿼리
    - 동일한 파라미터 값을 입력하더라도 결괏값이 달라질 수 있는 사용자 정의 함수(User-defined Function)나 스토어드 프로시저(Stored Procedure)를 사용하는 쿼리

  - Row포맷으로 복제될 때 보다 데이터에 더 많은 락을 검
    - INSERT INTO ... SELECT가 대표적
    - 적절한 인덱스가 없어 풀 테이블 스캔을 유발하는 UPDATE 쿼리
  - Row의 경우 변경 데이터 자체가 넘어가므로 동일 데이터 변경일지라도 락을 더 적게 점유하고 속도도 빠름
- 제약사항
  - 트랜잭션 격리 수준이 `REPEATABLE-READ` 이상이어야 함

16.4.2 Row 기반 바이너리 로그 포맷

- 5.1 버전부터 도입됨
- 데이터 변경시 변경된 값 자체가 바이너리 로그에 기록되는 방식임
- 소스 서버와 레플리카 서버의 데이터를 일관되게 하는 가장 안전한 방식
  (5.7.7 버전부터는 기본 포맷임)

- 위와 같은 특성 덕분에 비확정적인 함수를 실행해도 데이터를 기록하므로 문제가 되지 않습니다.

- 주의사항
  - 데이터가 기록된다 == 바이너리 로그의 볼륨이 단시간에 커질 수 있다
  - 실행된 변경 내역을 SQL문 형태로 확인하려면 레플리카 서버의 릴레이 로그나 바이너리 로그(log_slave_updates 옵션 활성화 돼 있는 경우)를 mysqlbinlog 프로그램을 사용해 변환해야 합니다.
    (
    `세부사항은 472page 확인`)
- Row 포맷은 모든 트랜잭션 격리 수준에서 사용 가능합니다.
- 다만 Row 포맷이어도 `사용자 계정 생성`, `권한 부여 및 회수`, `테이블과 뷰 및 트리거 생성` 등과 같은 DDL문은 `Statement 포맷` 형태로 바이너리 로그에 기록됩니다.

16.4.3 Mixed 포맷

- 두 가지 바이너리 로그 포맷을 혼합해 사용합니다.
- 쿼리 대부분은 Statement 포맷, 안전하지 못한(비확정적인 쿼리 등) 쿼리는 Row 포맷으로 변환되어 기록합니다.
- mysql 서버가 내부 설정 기준과 기술적인 측면을 고려해 자동으로 두 포맷을 번갈아 사용하기에 사용자가 예상했던 결과와 다르게 처리될 수 있음에 유의해야 합니다.

16.4.4 Row 포맷의 용량 최적화

- Row 기반의 바이너리 포맷을 사용할때 가장 큰 걱정은 바이너리 로그 파일의 용량이 다른 포맷보다 많이 커질 수 있습니다.
- 따라서 Row 포맷을 사용할 때 바이너리 로그 파일의 용량을 줄일 수 있는 두 가지 방법을 제공합니다.

---

1. `바이너리 로그 Row 이미지`

- Row 포맷을 사용하는 경우 Statement 포맷보다 더 많은 저장 공간과 네트워크 트래픽을 유발할 가능성이 있습니다.
- `binlog_row_image` 시스템 변수를 통해 저장되는 변경 데이터의 칼럼 구성을 제어할 수 있습니다.
- 바이너리 로그에는 각 변경 데이터마다 `변경 전 레코드`(Before-Image)와 `변경 후 레코드`(After-Image)가 함께 저장됩니다.
- `binlog_row_image`는 3개의 옵션 중에 하나를 설정해 어떤 칼럼을 기록할지 결정합니다.
  (기본값은 full)

  - full

    특정 칼럼에서만의 변경 여부와 관계없이 변경이 발생한 레코드의 모든 칼럼들의 값을 바이너리 로그에 기록하는 방식이다. INSERT, UPDATE, DELETE 문장별로 바이너리 로그 파일에 기록되는 정보는 달라진다. INSERT 문장의 경우 새롭게 INSERT된 레코드의 모든 칼럼들만 바이너리 로그 파일에 기록되며, UPDATE의 경우에는 변경 전의 레코드와 변경 후의 레코드 모두 전체 칼럼들의 셋으로 바이너리 로그에 기록된다. DELETE 문장의 경우에는 변경 전의 레코드의 전체 칼럼들만 바이너리 로그에 기록된다.

  - minimal

    변경 데이터에 대해 꼭 필요한 칼럼들의 값만 바이너리 로그에 기록한다. INSERT, UPDATE, DELETE 문장별로 정확히 어떤 칼럼들이 바이너리 로그에 기록되는지는 아래에서 다시 자세히 살펴보겠다.

  - noblob

    full 옵션을 설정한 것과 동일하게 작동하지만 레코드의 BLOB이나 TEXT 칼럼에 대해 변경이 발생하지 않은 경우 해당 칼럼들은 바이너리 로그 파일에 기록하지 않는다.

- binlog_row_image 시스템 변수가 minimal로 설정됐을 때 INSERT, UPDATE, DELETE 문장에 따라 바이너리 로그의 변경 전후 레코드 기록은 아래와 같습니다.

표 16.3 binlog_row_image 시스템 변수가 minimal일 때 이벤트 종류별 바이너리 로그에서의 변경 전후 레코드 기록 내용

이벤트 종류	변경 전 레코드(Before Image)	변경 후 레코드(After Image)
INSERT	(없음)	INSERT 시 값이 명시됐던 모든 칼럼과 Auto-Increment 값 (테이블에 Auto-Increment 칼럼이 있는 경우에 한함)
UPDATE	PKE	UPDATE 시 값이 명시됐던 모든 칼럼
DELETE	PKE	(없음)

  - `Primary Key Equivalent`는 PK 혹은 PK 역할을 하는 칼럼 조합을 말합니다. PK가 없으면 유니크 인덱스로 그것도 없으면 모든 칼럼 조합이 PKE로 취급됩니다.

2. `바이너리 로그 트랜잭션 압축`

- mysql 서버의 바이너리 로그는 안정적인 복제를 위해 일정 기간 동안 보관되도록 설정하며, 시점 복구(Point-In-Time Recovery)를 고려하는 경우 원격 스토리지 서버에 바이너리 로그를 백업하기도 합니다.
- 디스크 저장 공간이나 네트워크 대역폭 사용량 절약을 위해 로그 보관 주기를 더 짧게 설정하는 경우도 있습니다.
    - 하지만, 원격 레플리카 서버로 바이너리 로그를 전송함에 따라 소비되는 네트워크 대역폭 사용량은 사용자가 줄일 수 없고, 보관 주기를 짧게 하는것도 한계가 있습니다.

- 8.0.20 버전부터는 Row 포맷으로 기록되는 트랜잭션에서 변경한 데이터를 압축해서 바이너리 로그에 기록할 수 있습니다.

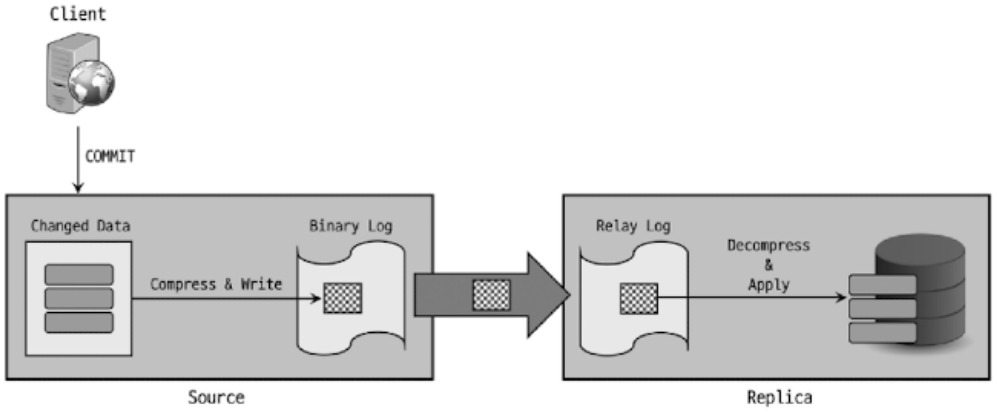

그림 16.7 압축된 바이너리 로그 트랜잭션의 복제

    - 트랜잭션에서 변경한 데이터들을 변경한 데이터들을 특정 알고리즘(zstd와 같은)을 통해 압축하고 하나의 이벤트로 바이너리 로그에 기록합니다.
    - 이는 레플리카 서버로 복제될 때도, 레플리카 서버의 레플리케이션 I/O 스레드도 압축된 상태 그대로 릴레이 로그에 기록합니다.
        - 디스크 저장 공간 및 네트워크 대역폭 사용량도 줄어듭니다.
    - 다만, 8.0.20이상의 버전을 사용해야 합니다.

- 시스템 변수를 통한 설정

- `binlog_transaction_compression` 시스템 변수를 통해 압축 기능을 활성화
  - (ON(1), OFF(0) 기본값은 OFF)
- `binlog_transaction_compression_level_zstd` 시스템 변수를 통해 압축시 사용되는 zstd 알고리즘 레벨을 설정합니다.
  - 압축 레벨은 1~22까지 값을 지정할 수 있고, 기본값은 3입니다.
  - 압축 수준이 높아지면 CPU, 메모리 사용량이 늘어나고 처리 시간 증가가 야기될 수 있습니다.
    (반드시 압축률이 좋아지진 않음)
- 세션별로도 설정 가능합니다. (압축, 비압축 데이터의 공존이 문제되지 않음)
- 특정 이벤트들은 항상 바이너리 로그에 압축되지 않습니다.

  - GTID 설정 관련 이벤트
  - 그룹 복제에서 발생하는 View Change 이벤트 또는 소스 서버에서 레플리카 서버에 살아있음을 알리는 Heartbeat 이벤트와 같은 제어 이벤트(Heartbeat 이벤트는 바이너리 로그에 실제로 기록되지는 않는다.)
  - 복제 실패 및 소스 서버와 레플리카 서버 간 데이터 불일치를 발생시킬 수 있는 Incident 타입의 이벤트
  - 트랜잭션을 지원하지 않는 스토리지 엔진에 대한 이벤트 및 그러한 이벤트를 포함하고 있는 트랜잭션 이벤트
  - Statement 포맷으로 기록되는 트랜잭션 이벤트(바이너리 로그 포맷이 `MIXED`로 설정돼 있는 경우에 해당한다고 볼 수 있다. 바이너리 로그 트랜잭션 압축 기능은 Row 포맷으로 기록되는 이벤트들에만 적용된다.)

- 압축된 트랜잭션 데이터는 개별 이벤트들의 내용이 어떤 것인지 실제로 확인이 필요할 때 압축이 해제됩니다.

  - 레플리카 서버에서 레플리케이션 SQL 스레드에 의해 복제된 트랜잭션이 적용될 때
  - `mysqlbinlog`를 사용해 트랜잭션을 재실행할 때
  - `SHOW BINLOG EVENTS` 혹은 `SHOW RELAYLOG EVENTS` 구문이 사용될 때

- `mysqlbinlog`를 통해 압축된 크기, 압축되지 않은 크기 및 알고리즘을 확인할 수 있습니다.(477~478page)
- `Performance` 스키마를 통해 압축된 트랜잭션들의 통계 정보와 압축 성능을 확인할 수 있습니다.

  (레플리카 서버에서 바이너리 로그 및 log_slave_updates 설정이 활성화 된다면 `릴레이 로그 + 바이너리 로그`에 대한 통계 정보도 함께 표시됩니다.)

(`479 ~ 480page`)

- 압축 성능 관련해서도 Performance 스키마를 통해 확인할 수 있는데 UPDATE문을 통해 스키마 설정을 변경해야 합니다. 그러면 설정 이후부터 데이터가 수집됩니다.

  (`481~482page`)

  (다만 수집 정보가 늘어나며 부하가 발생할 수 있으므로 테스트를 해봐야 합니다.)

- `테스트 결과`

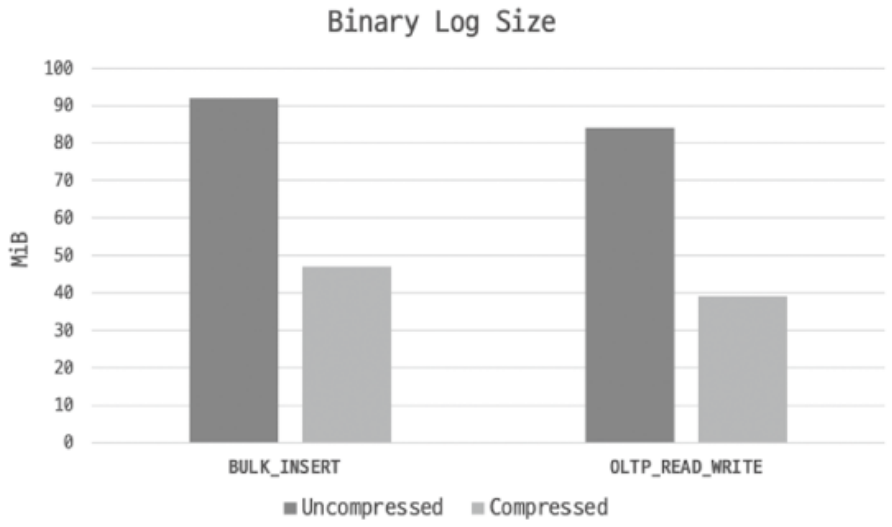

그림 16.8 압축 기능 사용 여부에 따른 바이너리 로그 크기 비교

  - 압축 레벨은 기본으로 설정되는 값(3)임

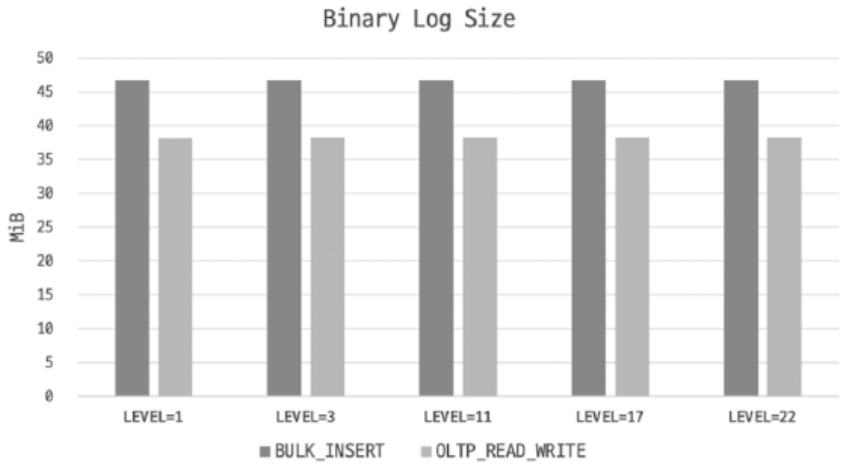

그림 16.9 압축 레벨별 바이너리 로그 크기 비교

- 레벨별 압축률 차이 → 크게 나지 않음

표 16.4 압축 기능 사용 여부에 따른 소요 시간 및 평균 CPU 사용률

압축 사용 여부	총 소요 시간(초)	평균 CPU 사용률(%)
미사용	39.63	12.33
사용	47.71	12.44

- 압축을 사용하면 소요시간에서 꽤 차이가 발생합니다.

- `결론`
  - 압축 기능을 사용하고 할 때는 현재 `mysql 서버의 리소스 사용률 현황`, 서비스 요건을 충족 시키는 쿼리 응답 속도 등을 파악하고 별도로 구축한 `테스트 환경에서 성능을 확인`해 압축 기능의 사용 여부를 결정해야 합니다.

✅ 16.5 복제 동기화 방식

복제 동기화 방식으로는 `비동기 복제(Asynchronous replication)`와 `반동기 복제(Semi-synchronous replication)`가 존재합니다.

16.5.1 비동기 복제(Asynchronouse replication)

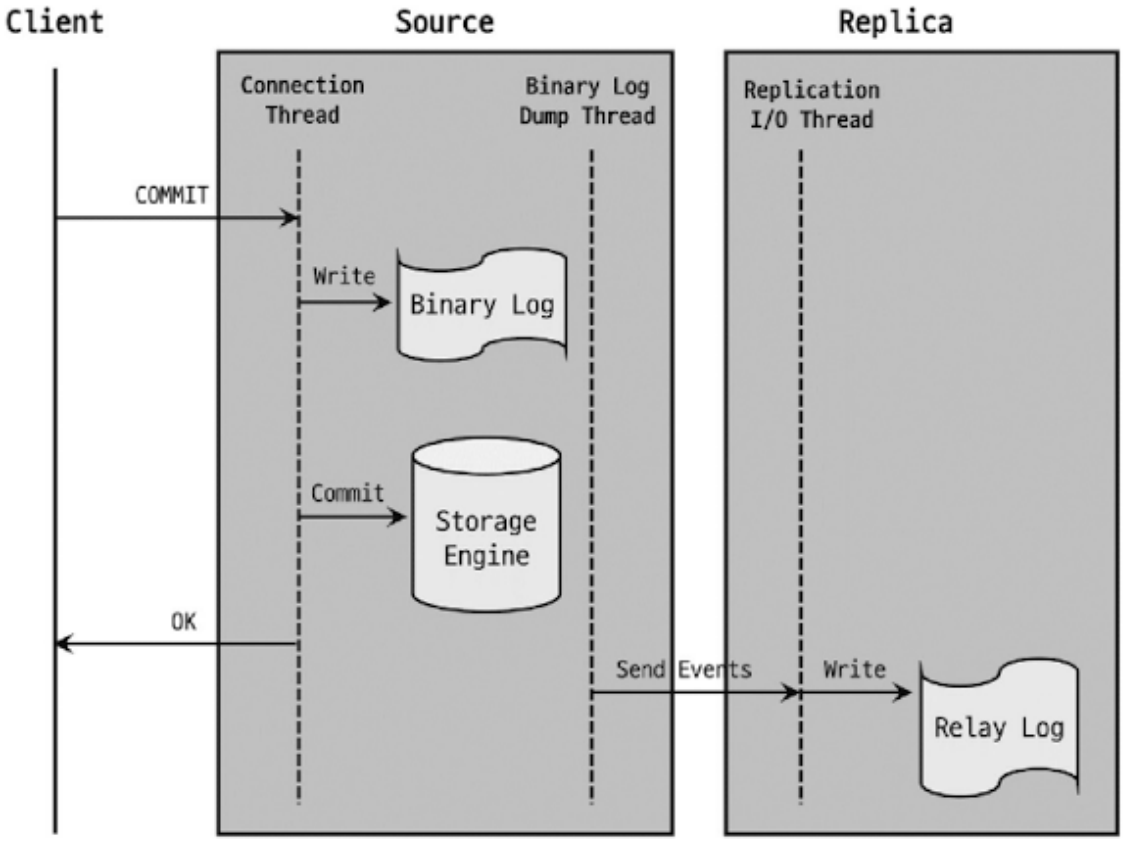

그림 16.10 비동기 복제 동작 방식

- 소스 서버가 레플리카 서버에서 변경 이벤트가 정상적으로 전달됐고, 적용됐는지 확인하지 않는 비동기 방식으로 동작합니다.
- 소스 서버에서 커밋된 트랜잭션은 바이너리 로그에 기록되고, 레플리카 서버는 주기적으로 신규 트랜잭션에 대한 바이너리 로그를 소스 서버에 요청합니다.
- 다만 잘 적용됐는지 소스 서버는 알 수 없고, 만약 소스서버에서의 장애가 발생하면 트랜잭션이 전달되지 않안 트랜잭션이 누락이 발생할 수 있습니다.
    - 이는 레플리카 서버를 소스 서버로 승격시킬때 문제가될 수 있음

장단점

- 장점
    - 레플리카 서버로 전달 및 적용을 신경쓰지 않으므로 트랜잭션 처리가 좀 더 빠릅니다.

    - 레플리카 서버의 문제가 소스 서버로 전파되지 않습니다.
    - 너무 많은 레플리카 서버가 아니라면 소스 서버에 여러대의 레플리카 서버를 연결해도 큰 성능저하가 없습니다.
- `단점`
    - 소스 서버가 레플리카 서버의 동기화 여부를 보장하지 않으므로 트랜잭션 누락과 같은 현상이 발생할 수있습니다.

> **참고** 비동기 복제에서는 사용자가 소스 서버에서 데이터를 변경한 후 바로 레플리카 서버에서 해당 데이터를 확인했을 때 변경 전 데이터가 보여질 수도 있다. 그러나 일반적으로 특별한 문제가 없다면 소스 서버에서 실행된 쿼리는 2~300밀리초(MySQL 서버의 처리 성능이나 네트워크 속도에 따라서 가변적이긴 하지만) 이내의 짧은 시간 내에 레플리카 서버에도 적용된다. 따라서 서비스에서 직접적으로 사용되는 읽기 쿼리를 레플리카 서버에서 실행한다 하더라도 큰 문제는 없다고 볼 수 있다. 다만 즉각적으로 반영된 데이터를 조회해야 하는 민감한 경우에는 레플리카 서버보다는 소스 서버에서 직접 데이터를 읽어 가도록 구현하는 것이 좋다.

16.5.2 반동기 복제(Semi-synchronous replication)

- 비동기 복제보다 좀 더 향상된 데이터 무결성을 제공하는 복제 동기화 방식
- 레플리카 서버가 전달받은 변경 이벤트를 릴레이 로그에 기록하고 소스서버에게 ACK 응답을 보냅니다.
    - 이때 소스 서버는 설정에 따라 ACK를 받고 커밋할지, 커밋 이후 ACK를 받을지를 결정합니다.
      (
      `rpl_semi_sync_master_wait_point`의 `AFTER_SYNC`, `AFTER_COMMIT)`
- 따라서 소스 서버에서 커밋되고 정상적으로 결과가 반환된 모든 트랜잭션들은 적어도 하나의 레플리카 서버에 해당 트랜잭션이 전송됨을 보장합니다. (적용은 보장 안됨, 단순히 전송 및 릴레이 로그 저장만)

AFTER_SYNC, AFTER_COMMIT

소스 서버가 트랜잭션 처리 중 어떤 지점에서 레플리카 서버의 ACK를 기다리는지에 따른 차이

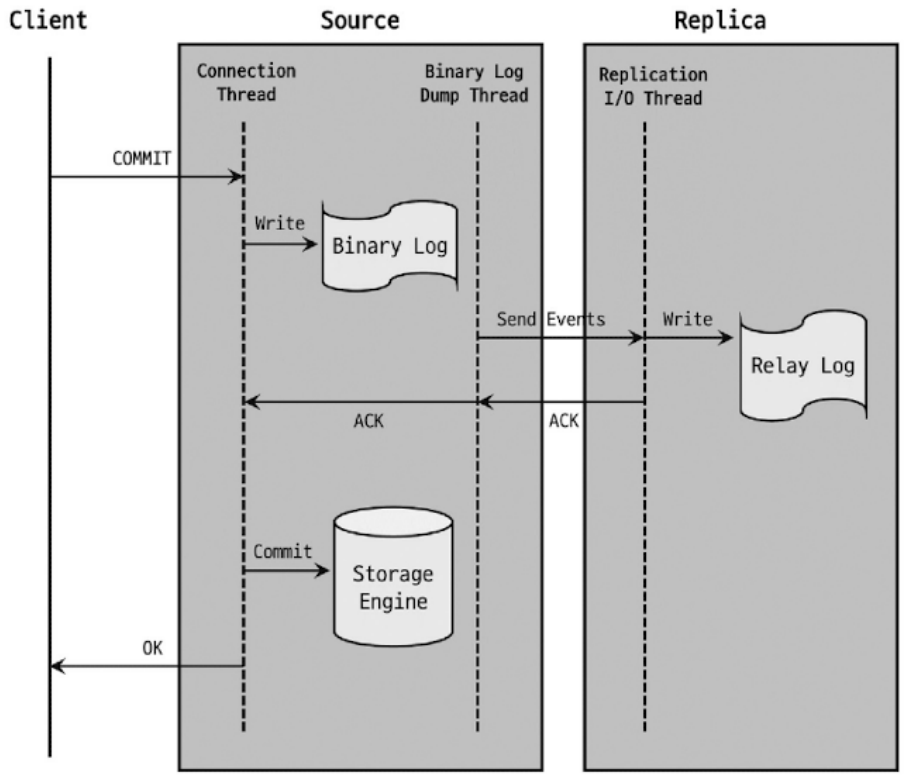

그림 16.11 AFTER_SYNC 반동기 복제 방식

- 각 트랜잭션을 바이너리 로그에 기록하고 레플리카 서버의 응답을 기다린 후 스토리지 엔진에 커밋함
- 5.7 버전부터 도입됐고, 8.0부터는 기본 방식입니다.

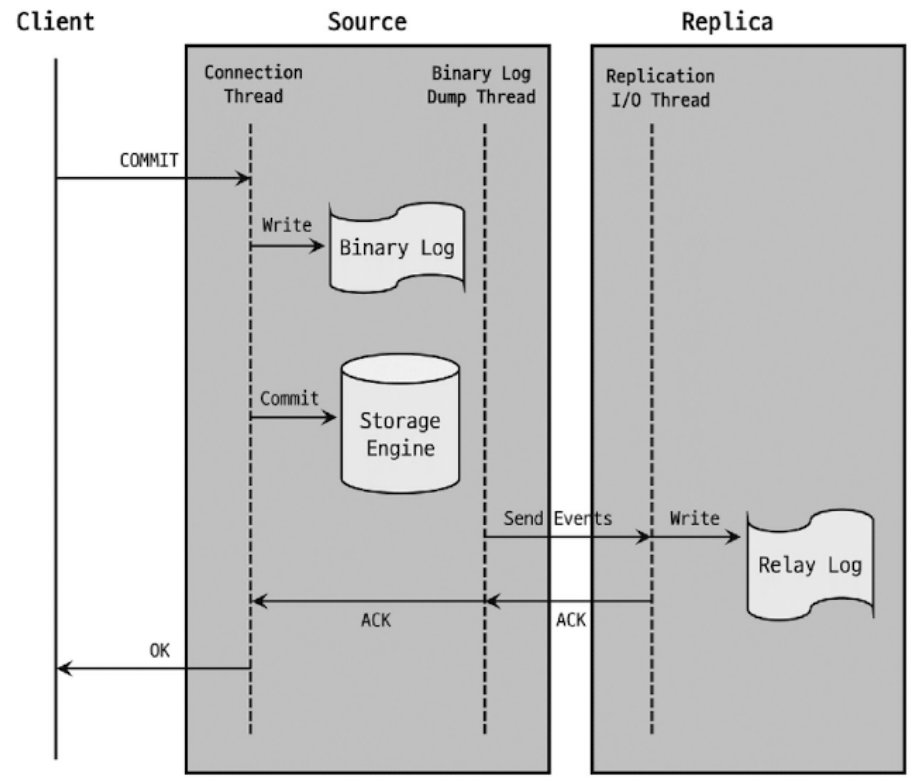

그림 16.12 AFTER_COMMIT 반동기 복제 방식

- 각 트랜잭션을 바이너리 로그에 기록하고, 스토리지 엔진에 커밋 이후 레플리카 서버의 응답을 기다리고 클라이언트에게 결과를 반환합니다.
- 5.7이전까지 사용되던 방식
- `Phantom Read` 문제 발생 여지
  - 소스 서버에선 커밋됐으나, 레플리카 서버에 복제가 되지 않은 경우 두 상태가 존재하게 되므로 팬텀리드가 발생할 수 있습니다.
  - 따라서 팬텀 리드 현상이 발생할 여지가 없는 `AFTER_SYNC` 방식이 기본값이 됐습니다.

반동기 복제 특징

- 트랜잭션 처리 중 레플리카 서버의 응답을 기다리므로 그만큼 비동기 방식에 비해 트랜잭션의 처리 속도가 더 느릴 수 있습니다. (응답이 늦어지면 더 느려짐)
- 따라서 반동기 복제는 물리적으로 가깝게 위치한 레플리카 서버와의 복제에 더 적합하다고 할 수 있습니다.

- 또한 지연과 같은 현상을 해결하기 위해 `타임아웃 시간`을 두어 시간이 지나면 자동으로 비동기 복제로 전환하던가, `사용자가 응답을 받아야 하는 레플리카 서버 수 등과 같이 다양한 설정`이 존재합니다.

반동기 복제 설정 방법(자세한 설정 방법은 `490 ~ 494page`를 참고)

- 여기서는 위에서 설명한 지연 현상을 완화하기 위해 설정할 수 있는 시스템 변수 몇개만 설명
  - 타임아웃 시간은 `rpl_semi_sync_master_timeout`을 통해 설정할 수 있습니다. 지정된 시간까지 레플리카의 응답이 없으면 비동기로 전환됩니다.
  - 소스 서버가 반드시 응답받아야 하는 레플리카 수는 `rpl_semi_sync_master_wait_for_salve_count`를 통해 설정할 수 있습니다.

또한 반동기 복제는 5.7.2 버전에 AFTER_SYNC 방식이 도입됐으므로 5.7.2 미만 버전과 5.7.2 이상 버전 이상의 서버간 반동기 복제는 제대로 동작하지 않을 수 있음에 주의해야 합니다.

✅ 16.6 복제 토폴로지

이번 절은 일반적으로 사용되는 복제 구성 형태들을 살펴보며 각 구성 형태별 적합한 용도와 주의점을 알아봅니다.

16.6.1 싱글 레플리카 복제 구성

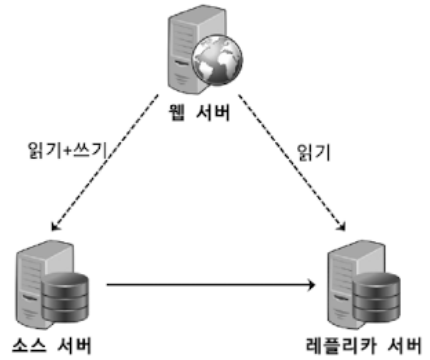

그림 16.13 싱글 레플리카 복제 구성

- 하나의 소스 서버에 하나의 레플리카 서버만 연결돼 있는 복제 형태를 말함(가장 기본적인 형태)
- 소스 서버 장애시 대응할 수 있는 예비 서버, 데이터 백업 서버 용도로 많이 사용됩니다.
- 만약 애플리케이션 서버가 레플리카 서버에서도 서비스용 읽기 쿼리를 실행할때 레플리카 서버에 문제가 발생하면 서비스 장애 상황이 발생할 수 있습니다.
    - 따라서 일대일로 구성된 형태에선 레플리카 서버를 예비용 서버로만 사용하는게 적합합니다.
    - 서비스와 연관 없는 쿼리(배치, 어드민 툴)들은 레플리카 서버에서 실행되어도 무방합니다.
      (문제가 생겨도 서비스 동작에 영향을 주지 않으므로)

16.6.2 멀티 레플리카 복제 구성

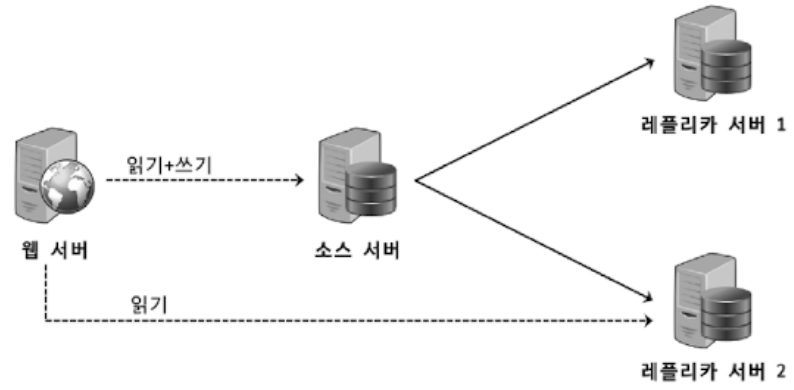

그림 16.14 멀티 레플리카 복제 구성

- 소스 서버에 2개 이상의 레플리카 서버를 연결한 복제 형태입니다.
- DB 서버로 유입되는 쿼리 요청이 많아지면 일반적으로 읽기 요청이 많아집니다. 따라서 하나의 레플리카 서버는 읽기용으로 사용할 수 있습니다.
    - 레플리카 서버가 문제가 발생하거나, 소스 서버에 문제가 발생하게 되는 경우가 있습니다.
    - 이때 레플리카 서버1을 대체 레플리카 혹은 소스 서버의 대체 서버로 사용해야 합니다.
    - 이미 사용중인 상태라면 부하가 높아져 문제가 발생할 수 있기에, 2개의 레플리카 서버중에서 하나는 예비용으로 두는것이 좋습니다.

16.6.4 체인 복제 구성

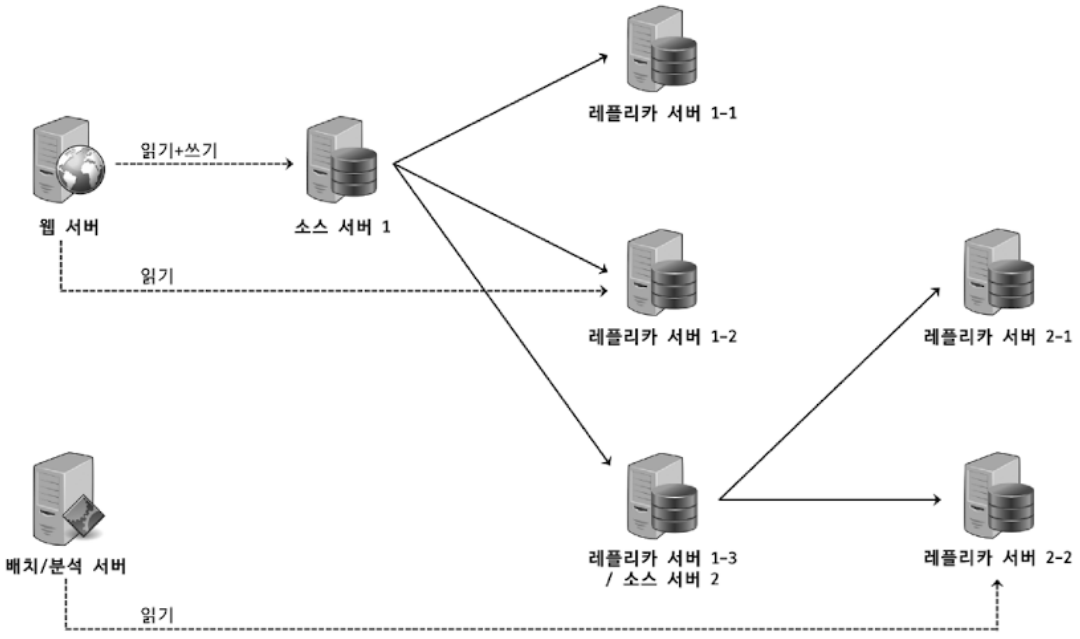

그림 16.15 체인 복제 구성

- 멀티 레플리카 복제 구성에서 레플리카 서버가 너무 많아 소스 서버의 악영향이 예상될 때 사용할 수 있습니다.
- 위와 같이 구성하여 기존의 소스 서버가 해야 할 바이너리 로그 배포 역할을 새로운 mysql 서버로 넘깁니다.
- 소스 서버 1과 연결된 서버들은 1차 복제그룹, 소스서버2는 2차 복제그룹일때 1차 복제그룹들은 주로 직접 연결돼 있으므로 서버의 변경이 빠르기에 OLTP용도로 2차 복제 그룹은 통계, 배치, 백업 용도로 구분해서 사용할 수 있습니다.

---

mysql 서버를 업그레이드 혹은 장비를 일괄 교체할때 체인 복제 구성 활용하기

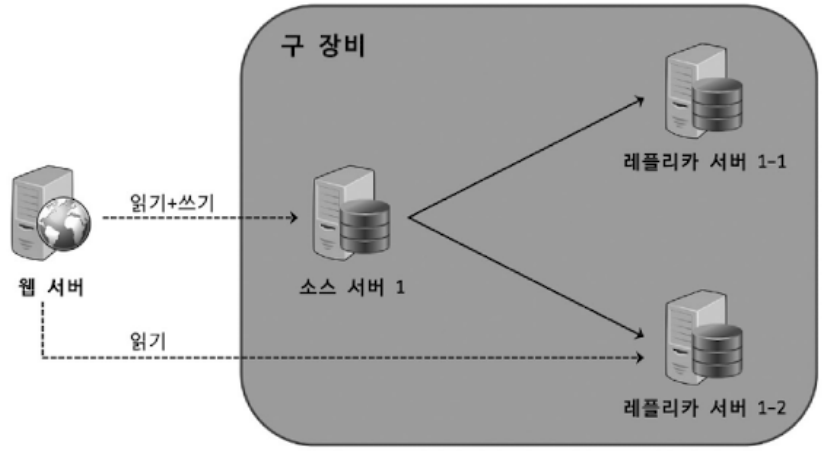

그림 16.16 장비 교체 1단계(초기 상태)

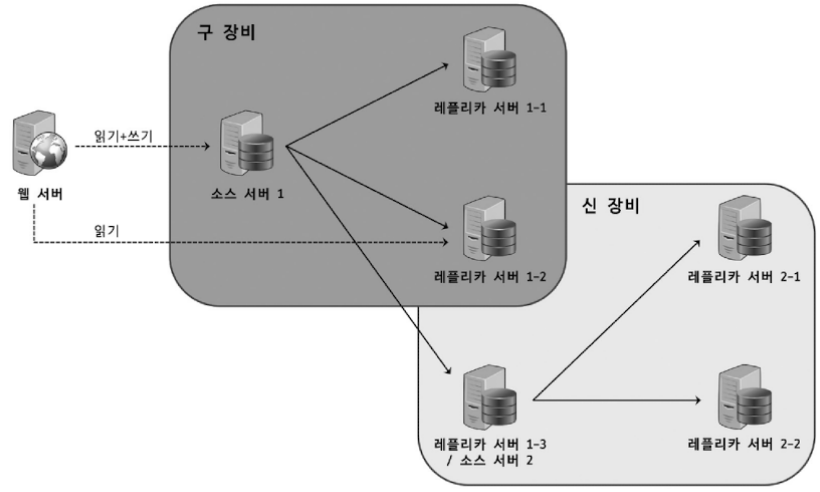

그림 16.17 장비 교체 2단계(새로운 MySQL 서버를 복제에 투입)

- 구 장비를 바라보던 도메인 네임, IP 주소를 새로운 mysql 서버를 바라보게 하고 웹 서버나 애플리케이션 서버를 한 대씩 돌아가며 재시작 합니다.(Rolling Restart)
- 재시작된 서버는 모두 신 장비로 이루어진 새로운 mysql 서버를 바라봅니다.

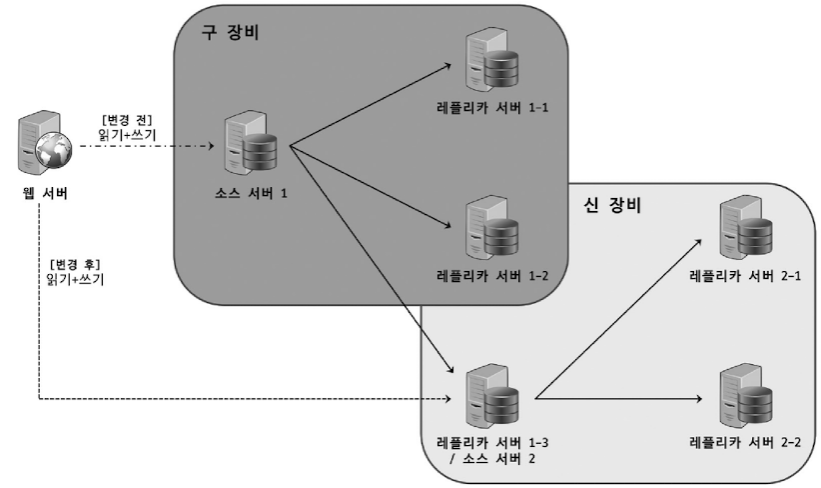

그림 16.18 장비 교체 3단계(웹 서버나 애플리케이션 서버를 새로운 MySQL 서버로 접속 유도)

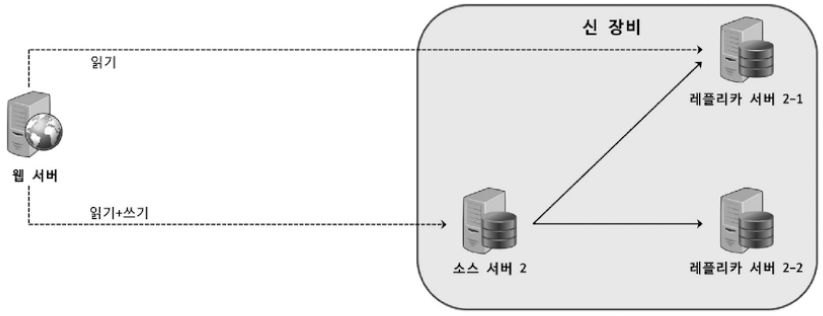

그림 16.19 장비 교체 4단계(구 MySQL 장비들을 복제에서 제거 및 이전 작업 완료)

- 완료됐다면 기존의 mysql 서버 3대는 모두 복제 그룹에서 제외시키면 됩니다.

- 위와 같이 구성하려면 레플리카 서버이자 소스 서버가 되는 `레플리카 서버1-3`은 바이너리 로그와 `log_slave_updates` 시스템 변수가 반드시 활성화 돼 있어야 합니다.
  (복제 내용도 바이너리 로그에 기록할 수 있어야 레플리카 서버 2-1, 2-2에 전달됨)
- 체인 복제 구성을 사용할 때는 중간 계층의 서버에서 장애가 발생하면 하위 계층의 레플리카 서버들도 복제가 중단됩니다. → 장애 처리시 복잡도가 좀 더 높음

16.6.4 듀얼 소스 복제 구성

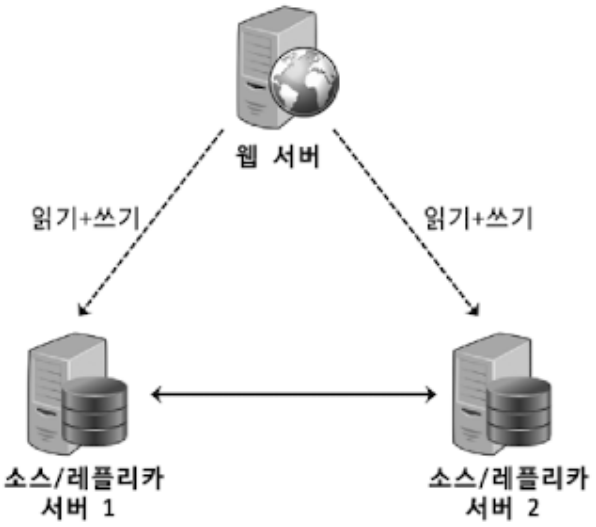

그림 16.20 듀얼 소스 복제 구성

- mysql 서버가 서로 소스 서버이자 레플리카 서버로 구성돼 있는 형태
- 두 mysql 서버 모두 쓰기가 가능하며, 구성 목적에 따라 두 가지 방식으로 나뉩니다.

1. `ACTIVE-PASSIVE`
    - 하나의 mysql 서버에서만 쓰기 작업이 수행되는 형태를 말함
    - 기존 서버가 문제가 발생하면 별도의 설정 없이 예비 서버인 다른 mysql 서버가 바로 쓰기 작업이 가능한 상태하다는 점에서 싱글 레플리카 복제와 차이를 보입니다.
2. `ACTIVE-ACTIVE`
    - 두 서버 모두에 쓰기 작업을 수행하는 형태
    - 다만 서로의 트랜잭션이 전달 완료되어 적용되기 전까지 두 mysql 서버는 서로 일관되지 않은 데이터를 가질 수 있고 다음과 같은 부분에서 문제가 발생합니다.
        - 동일한 데이터를 각 서버에서 변경 → 나중에 변경한 값으로 덮어씌워 질 수 있음
        - 테이블에서 Auto-Increment 키 사용 → 동시에 추가되는 경우 동일한 AI값을 가질 수 있음

모든 소스 서버들은 다른 소스 서버의 변경 내용들을 복제를 통해 자신에게도 똑같이 실행해야 하기에 쓰기 확장 효과는 크지 않고, 트랜잭션 충돌로 인한 롤백, 복제 멈춤과 같은 역효과가 많은 편입니다.

오히려 책에서는 Sharding을 권장합니다.

16.6.5 멀티 소스 복제 구성

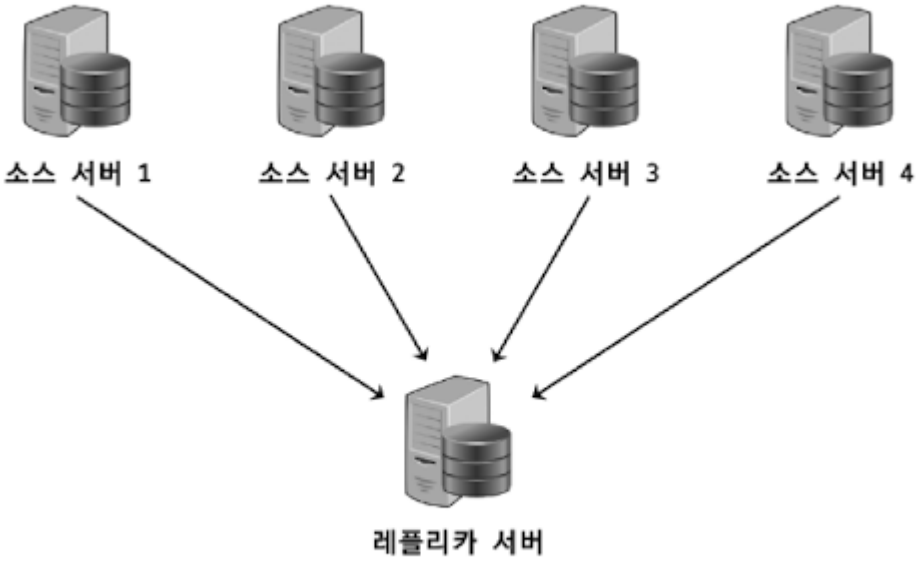

그림 16.21 멀티 소스 복제 구성

- 하나의 레플리카 서버가 둘 이상의 소스 서버를 갖는 형태입니다. (`5.7.6` 버전에서 처음 도입됐습니다.)
- 목적은 다음과 같습니다.
  - 여러 mysql 서버에 존재하는 각기 다른 데이터를 하나의 mysql 서버로 통합
  - 여러 mysql 서버에 샤딩돼 있는 테이블 데이터를 하나의 테이블로 통합
  - 여러 mysql 서버의 데이터들을 모아 하나의 mysql 서버에서 백업을 수행
- 멀티 소스 복제는 한곳으로 데이터를 모으거나, 샤딩된 데이터를 통합하고자 할때 손쉽게 구현할 수 있습니다.

1. 멀티 소스 복제 동작

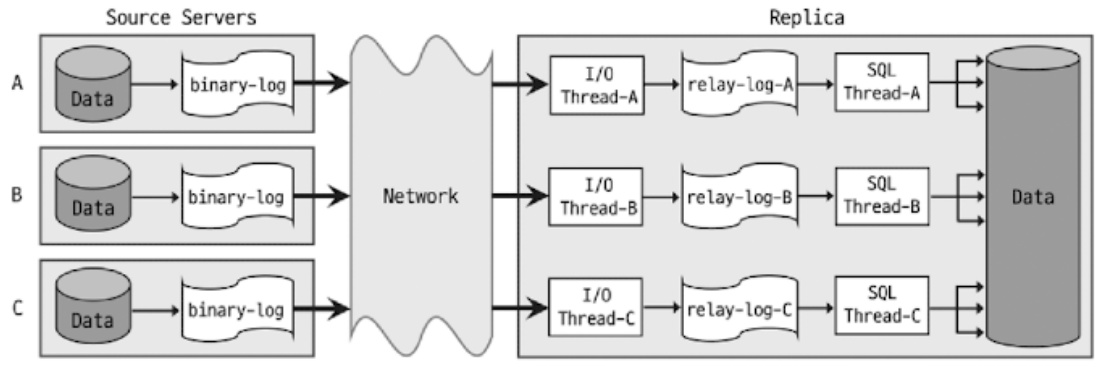

그림 16.22 멀티 소스 복제 동작 방식

- 멀티 소스 복제에서 레플리카 서버는 자신과 연결된 소스 서버들의 변경 이벤트들을 동시점에, 병렬로 동기화 합니다.
  - 각 소스 서버들에 대한 복제가 독립적으로 처리됨
  - 독립된 복제 처리를 채널(Channel)이라고 합니다.
  - 복제 채널은 개별적인 `레플리케이션 I/O 스레드`, `릴레이 로그`, `레플리케이션 SQL 스레드`를 가집니다.
  - 채널의 이름은 소스 서버와의 복제 연결인지를 구별할 수 있는 식별자 역할을 합니다. (최대 256개)

- `CHANGE REPLICATION SOURCE TO`(or `CHANGE MASTER TO`) 명령에서 `FOR CHANNEL` 구문을 사용해 복제 채널명을 지정할 수 있다.
- 복제 관련 명령에서도 `FOR CHANNEL` 구문과 함께 채널 이름을 지정하면 됩니다.

```
▪ CHNAGE [REPLICATION SOURCE | MASTER] TO ...      FOR CHANNEL ["channel_name"]
▪ START [REPLICA | SLAVE] [IO_THREAD | SQL_THREAD] FOR CHANNEL ["channel_name"]
▪ STOP [REPLICA | SLAVE] [IO_THREAD | SQL_THREAD]  FOR CHANNEL ["channel_name"]
▪ RESET [REPLICA | SLAVE]                          FOR CHANNEL ["channel_name"]
▪ SHOW [REPLICA | SLAVE] STATUS                    FOR CHANNEL ["channel_name"]
▪ FLUSH RELAY LOGS                                 FOR CHANNEL ["channel_name"]
▪ SHOW RELAY LOG EVENTS                            FOR CHANNEL ["channel_name"]
```

- 명령의 경우 FOR CHANNEL 절을 명시하지 않으면 전체 복제 채널에 대한 명령이 됩니다.

- START [REPLICA ¦ SLAVE] [IO_THREAD ¦ SQL_THREAD]
- STOP [REPLICA ¦ SLAVE] [IO_THREAD ¦ SQL_THREAD]
- SHOW [REPLICA ¦ SLAVE] STATUS
- FLUSH RELAY LOGS

- 멀티 소스 복제에서도 GTID 설정, 반동기 복제 방식 설정 등 모두 가능합니다.
- 각 복제 채널별로 멀티 스레드로 복제를 처리하거나, 소스 서버의 변경 이벤트를 필터링 할 수 있음

---

2. 멀티 소스 복제 구축

- 데이터가 없다면 그냥 구축만 하면 되지만, 데이터가 있다면 시스템 테이블 스페이스의 충돌 및 병합을 고려해야 합니다.
  - `mysqldump같은 논리 수준의 백업 도구 이용`

    InnoDB 시스템 테이블 스페이스를 물리적 백업하지 않으므로 병합과 관련된 문제가 발생하지 않음 백업된 데이터가 매우 크다면 mysqldump로는 데이터 백업, 적재 작업이 상당히 시간이 소요됩니다.
  - `XtraBackup과 같은 물리 수준의 백업 도구 이용`

    대용량의 데이터베이스를 빠르게 레플리카 서버로 가져올 수 있습니다. 다만 시스템 테이블 스페이스를 포함해 mysql 서버의 모든 데이터 파일들을 그대로 복사하고 복구하기에 충돌 및 병합에 대해서 처리해줘야 합니다.
- 결국 혼합해서 사용하는게 손쉬운 방법입니다.

- 시나리오 별 A, B소스 서버에서 C 레플리카 서버에 멀티 소스 복제를 연결 가정
  - `A 서버와 B 서버 모두 데이터가 크지 않은 경우`

    A, B서버의 mysqldump 결과를 차례대로 레플리카 서버 C에 적재하고 스토어드 프로시저나 함수, 유저 정보 및 권한만 따로 확인해서 조정하면 됩니다.
  - `A 서버의 데이터는 크고 B 서버의 데이터가 상대적으로 훨씬 작은 경우`

    데이터가 작은 B서버는 mysqldump를, 큰 A서버는 XtraBackup으로 백업해 레플리카 서버 C에 복구합니다.

    물론 스토어드 프로시나 함수, 유저 정보 및 권한은 체크를 해줘야 합니다.

    - `A 서버와 B 서버 모두 데이터 큰 경우`

      둘 다 XtraBackup을 이용해 물리 수준의 백업을 수행합니다.

      이때 테이블의 개수가 많은 서버를 먼저 레플리카 서버 C에 복구합니다. 남은 서버의 백업은 InnoDB 시스템 테이블 스페이스 충돌로 인해 ibd 파일을 InnoDB Export 명령을 명령을 사용하고 Import 하는 형태로 진행합니다.

      다만, 테이블 스페이스를 export, import는 테이블에 대해 수동으로 하나씩 진행해야 하므로 테이블이 적은 쪽 서버에 진행하는게 좋습니다.

- 멀티 소스 복제 구축 실습(자세한 설명은 `507 ~ 514page` 확인)
    - 멀티 소스 복제 구축 이전에 `master_info_repository`, `relay_log_info_repository` 시스템 변수들의 값이 반드시 `TABLE`로 설정돼 있어야 합니다.