[10주차] 16 ~ 16.3

태그	Done
날짜	@2024년 4월 2일 → 2024년 4월 9일
제목	복제

📌 16 복제

DB운영에서 가장 중요한 두 가지 요소는 확장성(`Scalability`)과 가용성(`Availability`) 입니다.

- 대용량 트래픽을 안정적으로 처리학 위해선 DB 서버의 확장이 필수적
- 사용자가 언제든 안정적인 서비스를 이용할 수 있으려면 DBMS를 포함한 하위 시스템들의 가용성이 뒷받침돼야 함

위 두 요소를 위해 가장 일반적으로 사용하는 기술은 복제(`Replication`)입니다.

✅ 16.1 개요

복제

한 서버에서 다른 서버로 데이터가 동기화 됨을 말하며 원본 데이터를 가진 서버를 Source Server, 복제된 데이터를 가지는 서버를 Replica라고 합니다.

- `소스 서버`
  - 데이터 및 스키마에 대한 변경이 최초로 발생
- `레플리카 서버`

    - 소스 서버의 변경 내역을 소스 서버로부터 전달받아 자신이 가지고 있는 데이터에 반영하여 소스 서버의 데이터와 동기화 시킵니다.

대부분의 DBMS가 제공하며 일반적으로 서비스에서 사용될 DB 서버 구축시에는 메인으로 사용하는 소스 서버와 복제를 통해 레플리카 서버를 한 대 이 상 함께 구축합니다.

가장 큰 목적은 소스 서버에 문제가 생겼을 때를 대비하는 이유가 크지만, 외에도 레플리카 서버를 구축하는데는 아래와 같은 이유가 있습니다.

1. `스케일 아웃(Scale-out)`

   사용자가 늘어나면 트래픽 증가 → DB 부하 증가로 직결됨 만약 DB 서버가 한 대 일때는 어떻게 처리할까?

   - 서버의 사양을 업그레이드 (`Scale-up`)

     애플리케이션 단의 큰 변화 없이 늘어난 트래픽을 처리할 수 있음 하지만, 한 대에서 처리할 수 있는 양에는 한계가 있습니다.

   - 동일 데이터를 가진 DB 서버를 한 대 이상 더 사용하는 스케일 아웃을 통해 애플리케이션으로부터 실행되는 쿼리들을 분산시킬 수 있습니다.
     - 이는 스케일 업 방식보다 갑자기 늘어나는 트래픽을 대응하는 데 훨씬 더 유연한 구조입니다.
     - 복제를 사용해 DB 서버를 스케일 아웃 할 수 있습니다.

2. `데이터 백업`

   서버의 데이터가 삭제되면 서비스 운영에 치명적일 수 있습니다. 따라서 DB에 저장된 데이터들을 주기적으로 백업하는게 필수적입니다. 일반적으론 DB 서버에서 백업 프로그램이 백업을 진행합니다.

   DBMS가 서버의 자원을 공유해서 사용하기에 백업으로 인해 DBMS에서 실행 중인 쿼리들이 영향을 받을 수 있습니다. 따라서 레플리카 서버를 구축하고 `백업은 레플리카 서버`에서 실행합니다.

   이때의 레플리카 서버는 소스 서버의 문제 발생시 대체할 수 있는 `대체 서버`의 역할을 합니다.

3. `데이터 분석`

차세대 모델을 위해 기존 데이터들을 분석할때의 분석용 쿼리는 대량의 데이터 조회 및 집계 연산 때문에 무겁고 복잡한 경우가 많습니다.

이럴때 복제를 통한 `레플리카 서버를 구축해 분석용 쿼리만 전용으로 실행될 수 있는 환경`을 만드는것이 좋습니다.

4. `데이터의 지리적 분산`

   애플리케이션 서버와 DB 서버가 물리적으로 떨어져 있는 경우 통신 시간은 거리만큼 비례합니다.

   빠른 응답 속도를 제공하기 위해 복제를 사용해 애플리케이션 서버가 위치한 곳에 기존 DB 서버에 대한 레플리카 서버를 새로 구축해 사용하여 응답속도를 개선할 수 있습니다.

✅ 16.2 복제 아키텍처

mysql 서버에서 발생하는 모든 변경 사항은 `바이너리 로그` 파일에 순서대로 기록됩니다. 이곳에는 `데이터 변경 내역`, `DB 및 테이블의 구조 변경`, `계정이나 권한의 변경 정보`까지 모두 저장됩니다.

바이너리 로그에 기록된 각 변경 정보들을 이벤트 라고도 합니다.

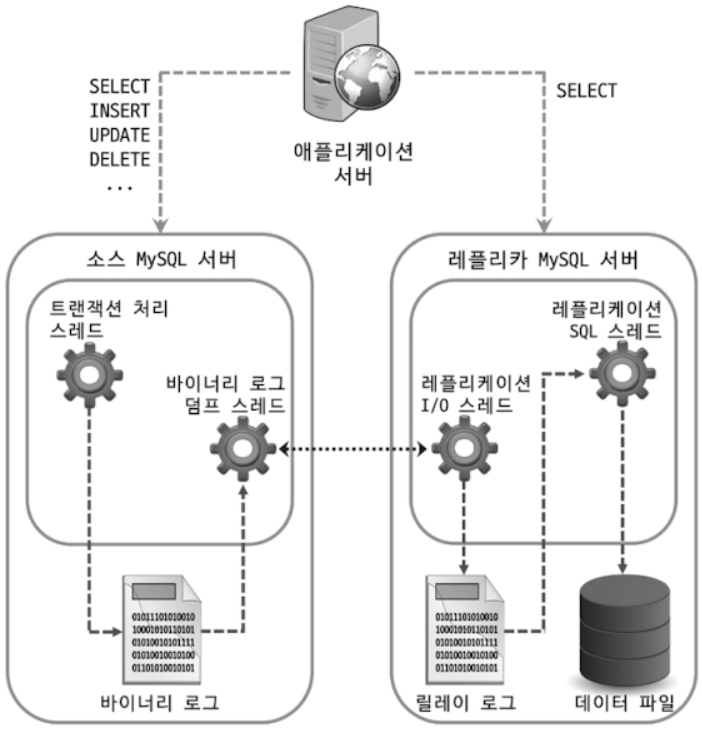

그림 16.1 MySQL 복제 동기화 과정

MySQL의 복제 또한 바이너리 로그를 기반으로 구현됐습니다.

- 소스 서버에서 생성된 바이너리 로그가 레플리카 서버로 전송
- 레플리카 서버는 해당 내용을 로컬 디스크에 저장하고 자신이 가진 데이터에 반영함

이를 통해 소스 서버 ↔ 레플리카 서버간 데이터 동기화가 이루어짐

레플리카 서버에서 소스 서버의 바이너리 로그를 읽어 따로 로컬 디스크에 저장해둔 파일을 `릴레이 로그(Relay Log)` 라고 합니다.

MySQL의 복제는 3개의 스레드에 의해 작동한다.

하나는 소스 서버, 두 개는 레플리카 서버에 존재함

- `바이너리 로그 덤프 스레드(Binary Log Dump Thread)`
  1. 레플리카 서버는 데이터 동기화를 위해 소스 서버에 접속해 바이너리 로그 정보를 요청
  2. 소스 서버는 레플리카 서버가 연결될 때 내부적으로 바이너리 로그 덤프 스레드를 생성하여 레플리카 서버로 전송함

3. 바이너리 로그 덤프 스레드는 레플리카 서버로 보낼 각 이벤트(변경 정보)를 읽을때 일시적으로 바이너리 로그에 잠금을 수행함
4. 이벤트를 읽고 난 후 바로 잠금을 해제함

이 스레드는 소스 서버에서 `SHOW PROCESSLIST` 명령을 통해 확인할 수 있음

- `레플리케이션 I/O 스레드(Replication I/O Thread)`
  1. 복제가 시작(START REPLICA or START SLAVE 명령)되면 레플리카 서버는 I/O 스레드를 생성하고 복제가 멈추면(STOP REPLICA or STOP SLAVE 명령) I/O 스레드는 종료됩니다.
  2. I/O 스레드는 소스 서버의 바이너리 로그 덤프 스레드로 부터 바이너리 로그 이벤트를 가져와 로컬 서버의 파일(릴레이 로그)로 저장하는 역할을 담당함
     - 소스 서버의 바이너리 로그를 읽어서 파일로 쓰는 역할만 하므로 I/O 스레드라고 함

  이 스레드는 MySQL 복제 현황을 보여주는 `SHOW REPLICA STATUS` or `SHOW SLAVE STATUS` 명령의 결과에서 `Replica_IO_Running` 또는 `Slave_IO_running` 칼럼에 표시된 값을 통해 확인할 수 있음

- `레플리케이션 SQL 스레드(Replication SQL Thread)`
  1. I/O 스레드에 의해 작성된 릴레이 로그 파일의 이벤트들을 읽고 실행합니다.
  2. SQL 스레드도 마찬가지로 `SHOW REPLICA STATUS` or `SHOW SLAVE STATUS` 명령을 통해 스레드의 상태를 확인할 수 있습니다. (`Replica_SQL_Running` or `Slave_SQL_running`) 칼럼에 현재 상태가 표시됨

In Replica Server

- 레플리케이션 I/O 스레드와 SQL 스레드는 독립적으로 동작함
  - SQL 스레드에서 이벤트를 적용하는게 느려도 I/O 스레드는 무관하게 소스 서버로부터 빠르게 이벤트를읽어올 수 있습니다.
  - 레플리카 서버가 소스 서버의 이벤트를 적용하는것도 소스 서버의 동작과는 별개로 진행되므로 레플리카 서버의 문제가 소스 서버까지 전파되진 않습니다.
  - 하지만 소스 서버의 문제로 레플리케이션 I/O 스레드가 정상 동작을 못하면 복제는 에러를 발생시키고 바로 중단됩니다. (복제 기능만 중단 됐으므로 레플리카 서버가

쿼리를 처리하는데는 문제 없음, 하지만 동기화는 실패)

복제가 시작되면

- 릴레이 로그를 비롯해 기본적으로 총 3개 유형의 복제 관련 데이터를 생성 및 관리합니다.

- `릴레이 로그(Relay Log)`

  레플리케이션 I/O 스레드에 의해 작성되는 파일, 소스 서버의 바이너리 로그에서 읽어온 이벤트(트랜잭션) 정보가 저장됨

  릴레이 로그는 바이너리 로그와 마찬가지로 현재 존재하는 `릴레이 로그 파일들의 목록이 담긴 인덱스 파일`과 `실제 이벤트 정보가 저장돼 있는 로그 파일`들로 구성됨

  릴레이 로그에 저장된 트랜잭션 이벤트들은 레플리케이션 SQL 스레드에 의해 레플리카 서버에 적용됨

- `커넥션 메타데이터(Connection Metatdata)`

  레플리케이션 I/O 스레드에서 소스 서버에 연결할 때 사용하는 DB 계정 정보, 현재 읽고 있는 소스 서버의 바이너리 파일명과 파일 내 위치 값 등이 담겨 있습니다.

  이런 정보는 `mysql.slave_master_info` 테이블에 저장됩니다.

- `어플라이어 메타데이터(Applier Metadata)`

  `Applier`는 레플리케이션 SQL 스레드가 릴레이 로그에 저장된 소스 서버의 이벤트를 레플리카 서버에 적용(Relay)하는 `컴포넌트`를 말합니다.

  `Applier Metadata`는 최근 적용된 이벤트에 대해 해당 이벤트가 저장돼 있는 릴레이 로그 파일명, 파일 내 위치 정보 들을 담고 있습니다.

  레플리케이션 SQL 스레드는 해당 정보들을 통해 레플리카 서버에 나머지 이벤트들을 적용합니다.

  이런 정보는 `mysql.slave_master_info` 테이블에 저장됩니다.

커넥션 및 어플라이어 메타 데이터는 mysql의 시스템 변수인 `master_info_repository`, `relay_log_info_repository`를 통해 어떤 형태로 데이터를 관리할지 FILE, TABLE 두 가지 중 하나로 설정합니다. (8.0.2 default `TABLE`, `FILE`은 Deprecated 될 예정)

- FILE로 설정하면

  두 메타데이터는 각각 디렉터리에서 `master.info`, `relay-log.info`라는 파일로 관리됩니다. 파일의 경로는 `--master-info-file` 옵션과 `relay_log_info_file` 시스템 변수를 이용해 경로를 지정할 수 있습니다.

  FILE의 경우 `레플리케이션 I/O 스레드`와 `SQL 스레드`가 동작할 때 이 두 파일의 내용이 동기화되지 않은 경우가 빈번하게 발생했습니다.

- TABLE로 설정하면

  mysql DB내 `slave_master_info`, `slave_relay_log_info` 테이블에 각각 데이터가 저장됨

  TABLE은 InnoDB 스토리지 엔진 기반의 테이블로 관리되므로 레플리케이션 SQL 스레드가 트랜잭션을 적용할 때 `slave_relay_log_info` 테이블의 데이터도 같은 시점에 Atomic하게 업데이트 됩니다.

  mysql이 얘기치 않게 종료됐다 하더라도 다시 구동했을때 문제없이 복제가 진행됩니다. (`Crash-safe` `replication`)

✅ 16.3 복제 타입

복제는 바이너리 로그에 기록된 변경 내역(바이너리 로그 이벤트)들을 식별하는 방식에 따라 2가지로 나뉩니다.

1. 바이너리 로그 파일 위치 기반 복제(Binary Log File Position Based Replication)
2. 글로벌 트랜잭션 ID 기반 복제(Global Transaction Identifiers Based Replication)

16.3.1 바이너리 로그 파일 위치 기반 복제

- 복제 기능이 처음 도입됐을 때부터 제공된 방식
- 레플리카 서버에서 소스 서버의 바이너리 로그 파일명과 파일 내에서의 위치(Offset or Position)로 개별 바이너리 로그 이벤트를 식별해 복제가 진행되는 형태

- **복제를 처음 구축한다면**
  - 레플리카 서버에 소스 서버의 어떤 이벤트부터 동기화를 할지에 대한 정보를 설정해야 합니다.

    - 복제가 설정된 레플리카 서버는 어떤 이벤트까지 로컬 디스크로 가져왔고, 적용했는지에 대한 정보를 관리하며 소스 서버에 해당 정보를 전달해 적용한것 이후의 바이너리 로그 이벤트를 가져옵니다.

`이를 위해서는 반드시 소스 서버에서 발생한 각 이벤트에 대한 식별이 필요하다`

- **바이너리 로그 파일 위치 기반 복제의 이벤트 식별 방법**
    - 이벤트 하나하나를 소스 서버의 바이너리 로그 파일명과 파일 내에서의 위치 값(File Offset)의 조합으로 식별함
    - 레플리카 서버는 이렇게 이벤트를 식별하고, 적용 내역을 추적하며 복제를 일시적으로 중단하거나, 재개할때 마지막으로 적용했던 이벤트 이후의 것들부터 읽어오기가 가능함

- **MySQL 서버들은 모두 고유한 server_id 값을 가지고 있어야 한다.**
    - 바이너리 로그의 각 이벤트 별로 이벤트가 최초 발생한 mysql 서버를 식별하기 위한 부가정보로 server_id도 함께 저장됩니다.
    - mysql 서버의 시스템 변수 중 하나로, 사용자가 서버마다 원하는 값으로 설정 가능합니다.(기본은 1)

- **바이너리 로그 파일에 기록된 이벤트가 레플리카 서버에 설정된 server_id와 동일한 server_id라면**
    - 레플리카 서버는 자신의 서버에서 발생한 이벤트로 간주해서 이벤트를 적용하지 않고 무시합니다.
    - 따라서 사용자는 바이너리 로그 파일 위치 기반으로 복제를 구축할때 복제의 구성원이 되는 모든 mysql 서버의 server_id를 고유하게 설정해야 합니다.

16.3.1.1 바이너리 로그 파일 위치 기반의 복제 구축

복제 설정 과정 및 구축 방법은 각 서버 데이터의 존재 여부, 복제의 활용 방식에따라 달라집니다.

아래 예시는 한 대로 구성해 사용하던 MySQL 서버에 새로운 레플리카 서버를 바이너리 로그 파일 위치 기반 복제로 연결하는 과정을 보여줌

1. `설정 준비`
    - 소스 서버는 반드시 바이너리 로그 활성화가 되어 있어야 함 + 각 서버는 모두 고유한 server_id값을 가져야함
    - 8.0은 기본적으로 활성화돼 있으므로 데이터 디렉터리 밑에 binlog라는 이름으로 바이너리 로그가 자동 생성됩니다.
    - 소스 서버는 server_id값만 적절하게 설정해도 복제가 가능합니다.
    - 바이너리 로그 파일 위치, 파일명 설정을 원하면 `log_bin` 시스템 변수를 통해 변경 가능 외에도 바이너리 캐시 메모리 크기, 바이너리 로그 파일 크기, 보관 주기 등도 지정 가능

```
## 소스 서버 설정
[mysqld]
server_id=1
log_bin=/binary-log-dir-path/binary-log-name
sync_binlog=1
binlog_cache_size=5M
max_binlog_size=512M
binlog_expire_logs_seconds=1209600
...
```

    (my.cnf파일)

    - 소스 서버에서 바이너리 로그 정상 기록 확인

```
mysql> show master status;
+---------------+----------+--------------+-------------
| File          | Position | Binlog_Do_DB | Binlog_Ignor
+---------------+----------+--------------+-------------
| binlog.000079 |    17530 |              |
+---------------+----------+--------------+-------------
```

바이너리 로그 파일 이름과과 현재까지 기록된 바이너리 로그의 위치(바이트 수)를 확인할 수 있음, 만약 트랜잭션 처리중이라면 값은 계속 증가할 것 입니다.

- 레플리카 서버도 중복되지 않는 고유 server_id만 설정해도 됩니다.
  (릴레이 로그 파일도 복제 설정시 데이터 디렉터리 밑에 자동 생성됨, 변경은 relay_log 시스템 변수)
  - 생성되는 릴레이 로그가 적용되면 자동으로 레플리카 서버가 삭제하는데 `relay_log_purge` 시스템 변수를 통해 삭제하지 않게할 수 있습니다. 다만 디스크의 여유공간을 살펴야 합니다.
- 레플리카 서버는 일반적으로 읽기 전용으로 사용되므로 `read_only` 설정도 함께 사용하는 편이 좋습니다.
- 소스 서버 장애시 레플리카 서버가 승격될 수 있음을 고려한다면 `log_slave_updates` 시스템 변수도 명시하는게 좋습니다.
  - 레플리카 서버는 복제에 의한 데이터 변경은 자신의 바이너리 로그에 기록하지 않지만 위 시스템 변수를 설정하면 기록하게 됩니다.

```
## 레플리카 서버 설정 my.cnf
[mysqld]
server_id=2
relay_log=/relay-log-dir-path/relay-log-nam
erelay_log_purge=ON
read_only
log_slave_updates
...
```

2. `복제 계정 준비`
   - 레플리카 서버가 소스 서버로부터 바이너리 로그를 가져올때 접속하기 위한 DB 계정이 필요합니다.
     (복제용 계정)

     복제에 사용되는 계정의 비밀번호는 `레플리카 서버의 커넥션 메타데이터에 평문 저장`되므로 보안 측면을 고려해 별도의 권한을 가진 계정을 생성해 사용하는게 좋습니다.
   - 복제용 계정은 복제 시작전 소스 서버에 미리 준비해야 하고 REPLICATION SLAVE 권한을 가지고 있어야 합니다.

```
CREATE USER 'repl_user'@'%' IDENTIFIED BY '!1Repl_user_
GRANT REPLICATION SLAVE ON *.* TO 'repl_user'@'%';

// 실제로는 %대신 필요한 IP 대역에만 복제 연결이 가능하도록 적절하
```

3. `데이터 복사`
   - 소스 서버 데이터를 레플리카 서버로 적재해야 하는데 mysqldump등과 같은 툴을 이용하면 됨
   - mysqldump를 통해 소스 서버의 데이터를 덤프할때는 `--single-transaction`과 `--master-data` 라는 두 옵션을 반드시 사용해야 합니다.
     - `--single-transaction`

       데이터 덤프시 하나의 트랜잭션을 사용하여 mysqldump가 테이블, 레코드에 잠금을 걸지 않고 InnoDB 테이블들에 대해 일관된 데이터를 덤프받게 함
     - `--master-data`

       덤프 시작 시점의 소스 서버의 바이너리 로그 파일명과 위치 정보를 포함하는 복제 설정 구문(`CHANGE REPLICATION SOURCE TO` or `CHANGE MASTER TO`)이 덤프 파일 헤더에 기록될 수 있게 함

       mysqldump는 위 옵션을 적용하면 `FLUSH TABLES WITH READ LOCK` 명령을 실행해 글로벌 락(모든 테이블에 대한 읽기 잠금)을 거는데, 바이너리 로그의 위치를 순간적으로 고정시키기 위함입니다.

       > **주의** 만약 mysqldump에 지정된 --master-data 옵션으로 소스 서버에 "FLUSH TABLES WITH READ LOCK" 명령이 실행되기 전에 MySQL 서버에 이미 장시간 동안 실행 중인 쿼리가 있다면 글로벌 락 명령어가 실행 중인 쿼리에서 참조하고 있는 테이블들에 대한 잠금을 획득할 수 없어 완료되지 못하고 대기하게 된다. 이처럼 글로벌 락 명령어가 대기하는 상황이 발생하면 그 뒤로 유입되는 다른 쿼리들도 연달아 대기해서 쿼리가 실행되지 못하고 적체될 수 있으며, 이 경우 서비스에 문제가 될 수 있으므로 mysqldump를 실행하기 전에 장시간 실행 중인 쿼리가 있는지 미리 확인하는 것이 좋다. 그리고 mysqldump를 실행한 후에도 앞서 설명한 것과 같은 대기 현상이 발생하고 있지는 않은지 한번 더 확인하는 것이 좋다. 글로벌 락에 대한 자세한 내용은 5.2.1절 '글로벌 락'에서 확인할 수 있다.

       옵션은 `1` 혹은 `2`로 설정가능함, 1은 덤프 파일 내의 복제 설정 구문이 실제 실행 가능한 형태로 기록, 2는 해당 구문이 주석을 처리되어 참조만 할 수 있는 형태로 기록됨
   - 로컬에서 mysqldump 실행

```
linux> mysqldump -uroot -p --single-transaction --maste
--opt --routines --triggers --hex-blob --all-databases
```

데이터 덤프가 완료되면 source_data.sql 파일을 레플리카 서버로 옮겨 데이터 적재를 진행합니다.

```
# source_data.sql파일이 레플리카 서버의 /tmp 디렉터리에 준비돼

# MySQL 서버에 직접 접속해 데이터 적재 명령을 실행
mysql> SOURCE /tmp/master_data.sql

## MySQL 서버에 로그인하지 않고 데이터 적재 명령을 실행
## 다음 두 명령어 중 하나를 사용
linux> mysql -uroot -p < /tmp/source_data.sql
linux> cat /tmp/source_data.sql | mysql -uroot -p
```

4. `복제 시작`

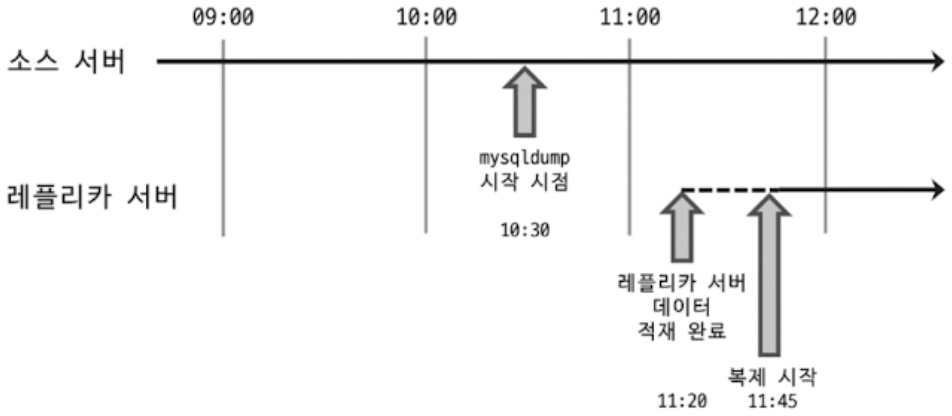

그림 16.2 소스 서버와 레플리카 서버의 데이터 상태

- 사전 준비를 마친 소스, 레플리카 서버는 위 그림과 같습니다. 10:30분에 mysqldump를하고 11:20 쯤에 레플리카 서버에 모두 적재했다는 의미는 `소스 서버의 데이터보다 50분이 지연된 상태입니다.`
- 소스 서버와 레플리카 서버 간의 복제 설정 과정의 자세한 내용은 책을 참조

    - mysqldump로 백업 받은 파일의 헤더에서 CAHNGE MASTER로 시작하는 줄을 복사
    - 편집기에 복사해둔 내용에 mysql 서버의 내용을 삽입

```
-- // MySQL 8.0.23 이상 버전
CHANGE REPLICATION SOURCE TO
  SOURCE_HOST='source_server_host',
  SOURCE_PORT=3306,
  SOURCE_USER='repl_user',
  SOURCE_PASSWORD='repl_user_password',
  SOURCE_LOG_FILE='binary-log.000002',
  SOURCE_LOG_POS=2708,
  GET_SOURCE_PUBLIC_KEY=1;

-- // MySQL 8.0.23 미만 버전
CHANGE MASTER TO
  MASTER_HOST='source_server_host',
  MASTER_PORT=3306,
  MASTER_USER='repl_user',
  MASTER_PASSWORD='repl_user_password',
  MASTER_LOG_FILE='binary-log.000002',
  MASTER_LOG_POS=2708,
  GET_MASTER_PUBLIC_KEY=1;
```

        - `GET_SOURCE_PUBLIC_KEY=1` : RSA 키 기반 비밀번호 교환 방식 통신을 위한 공개키를 소스 서버에 요청할지 여부를 나타냄
        - 계정 인증 플러그인이 caching_sha2_password로 설정되었다면 mysql 접속시 반드시 보안 연결 혹은 RSA키를 사용해 패스워드를 교환하는 방식의 비암호화된 연결을 사용해야 합니다.
- 복제 관련 정보만 등록된것이므로 동기화를 시작하려면 START REPLICA(START SLAVE)로 명령을 실행 합니다. (위 그림에서 11:45분에시작한것)
    - `SHOW REPLICA STATUS` 결과에서 `Replica_IO_Running`과 `Replica_SQL_Running` 값을 통해 시작여부를 확인 가능함
- 그렇게 되면 레플리카 서버는 가능한 빨리 `10:30 ~ 11:45`까지의 데이터 변경사항들을 소스 서버로부터 가져와 적용하게 됩니다.

- 완료되면 `SHOW REPLICA STATUS` 명령 결과에서 `Seconds_Behind_Source`(Seconds_Behind_Master)의 값이 0이되어 완전히 동기화 됨을 의미합니다.
- 시작을 해도 `Replica_IO_Running`과 `Replica_SQL_Running` 값이 NO라면 복제용 접속 계정, 비밀번호 확인 및 네트워크 상의 문제가 아닌지 체크가 필요합니다.

5. `바이너리 로그 파일 위치 기반의 복제에서 트랜잭션 건너뛰기`
   - 레플리카 서버에서 소스 서버로 넘어온 트랜잭션이 제대로 실행되지 못하고 에러가 발생해 복제가 멈출때도 있는데, 대부분은 사용의 실수가 많습니다.
   - 수동으로 복구가 불가능할 정도라면 처음부터 구성하는 경우도 있고 트랜잭션을 무시하고 넘어가도록 처리할 수 있습니다.
     - `sql_slave_skip_counter` 시스템 변수를 이용하면 문제의 트랜잭션을 건너 뛸 수 있습니다.
     - 위 값을 특정 숫자로 지정하면 해당 숫자만큼의 이벤트 그룹에 포함된 DML쿼리 문장을 무시합니다.
     - 만약 트랜잭션을 지원하지 않는다면 DML 문장 하나하나가 이벤트 그룹이 됩니다.

16.3.2 글로벌 트랜잭션 아이디(GTID) 기반 복제

- 5.5 버전까지는 복제 설정시 바이너리 로그 파일 위치 기반 복제 방식만 가능했습니다.
- 복제에서 각각 이벤트들은 바이너리 로그 파일명과 파일 내 위치 값의 조합으로 식별되는데, `문제는 식별이 바이너리 로그 파일이 저장돼 있는 소스 서버에서만 유효합니다.`
- `데이터베이스 서버들은 동일한 이벤트에 대해 서로 다른 식별 값을 갖게` 되기에 바이너리 로그 파일명, 파일내 위치값은 다른 레플리카 서버에선 사용할 수 없습니다.
- 따라서 GTID를 사용하게 됩니다. GTID는 글로벌 트랜잭션 아이디로 복제에 참여한 전체 MySQL 서버들에서 고유하도록 각 이벤트에 식별 값을 부여하여 이를 기반으로 복제가 진행됩니다.

---

1. `GTID의 필요성`

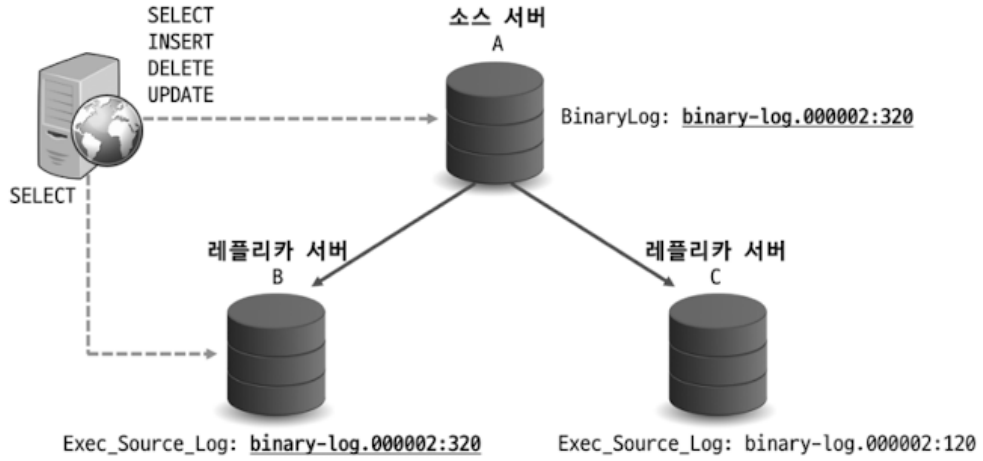

그림 16.3 장애가 발생하기 전 복제 토폴로지

- B는 SELECT 쿼리 분산용, C는 배치, 통계용입니다.
- 소스서버 A가 존재하고 레플리카 서버 B는 A와 100% 동기화 됐습니다.
  C서버는 지연이 발생해서 이전 위치까지만 복제가 동기화 됐습니다.

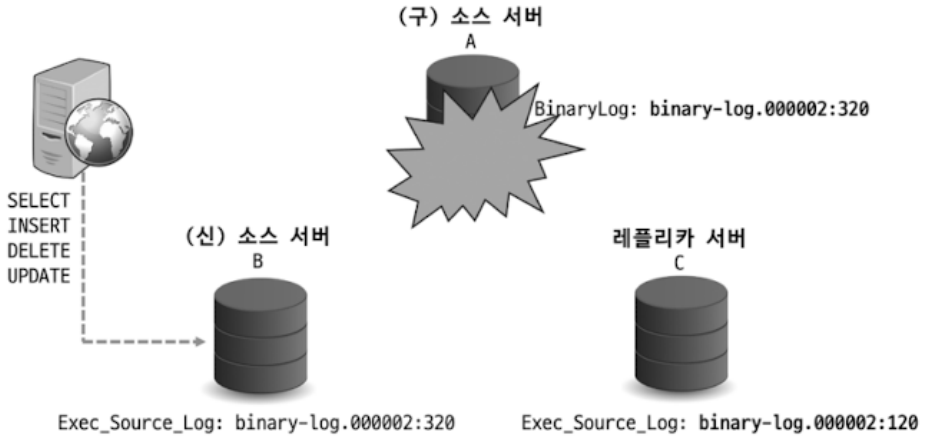

그림 16.4 장애 발생 후 B 서버를 소스 서버로 승격

- 만약 A서버가 장애가 발생했고, B 서버를 소스 서버로 승격(`Promotion`)하고, C에 대해선 기존 B가 받던 SELECT 쿼리를 받게 하려 합니다.
- 하지만 C서버는 동기화가 완벽하지 않은 상태에서 A서버가 종료됐으므로 복제를 최종 시점까지 동기화할 방법이 없습니다.
  - B서버의 릴레이 로그가 남아있으면 해당 로그는 소스 서버의 바이너리 로그 위치가 함께 있으므로 복구가 가능하지만, 자동 삭제되는 릴레이 로그 특성상 제

한적입니다.

- GTID를 구성하면 A, B, C서버 모두 동일한 GTID를 가지고 있으므로 위와 같은 상황에서 C서버가 B서버를 통해 동기화를 진행할 수 있습니다.

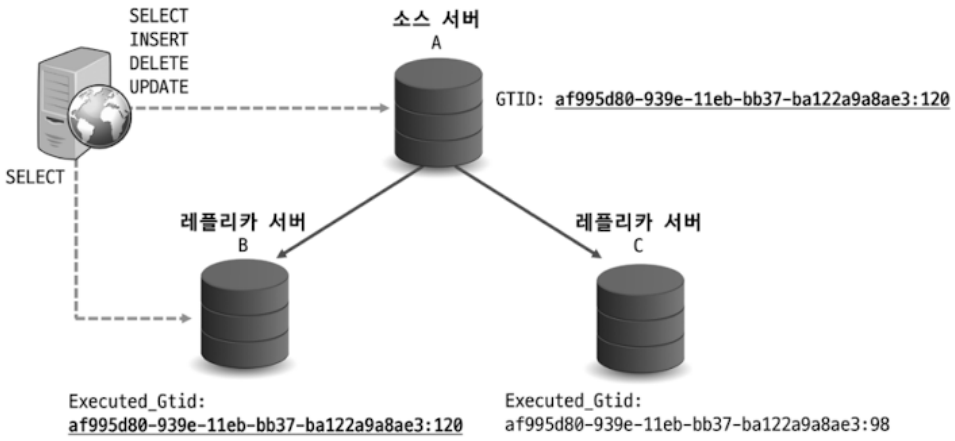

그림 16.5 GTID 사용 시 장애가 발생하기 전 복제 토폴로지

  - A 서버에서의 GTID는 B서버에서도 동일한 GTID를 가지고 C도 마찬가지 입니다.

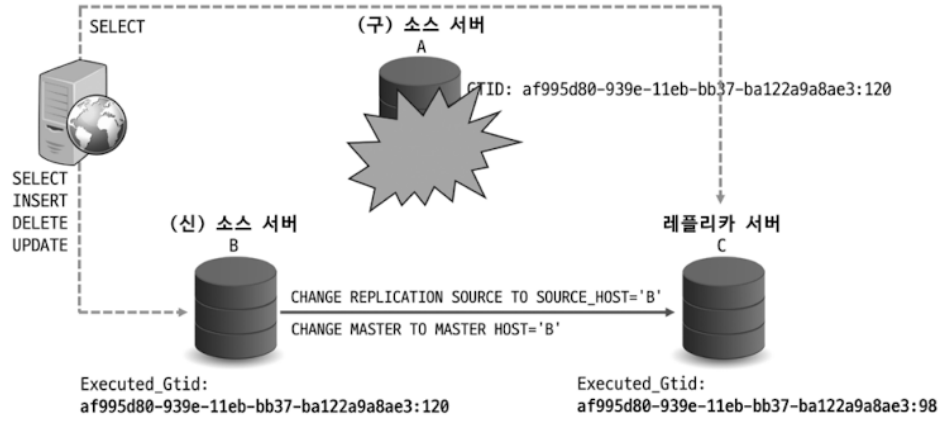

그림 16.6 GTID 사용 시 장애가 발생한 후 B 서버를 소스 서버로 승격

  - GTID를 통해 토폴리지 변경 시 동기화 문제가 간단히 해결됩니다.

---

2. 글로벌 트랜잭션 아이디

- 바이너리 로그 파일에 기록된 이벤트의 파일명, 파일 내 위치를 통해 식별하는건 물리적인 방식입니다.
- GTID는
  - 논리적인 의미로 파일 이름, 위치와는 무관하게 생성되며 서버에서 커밋된 각 트랜잭션과 연결된 고유 식별자 입니다.
  - 트랜잭션 발생 서버 뿐만 아니라 서버가 속한 복제 토폴리지 내 모든 서버에서 고유한 식별자 입니다.
  - 커밋되어 바이너리 로그에 기록된 트랜잭션에 한해서만 할당됩니다.
- SELECT 쿼리 sql_log_bin 설정이 비활성화돼 있는 상태에서 발생한 트랜잭션은 바이너리에 기록되지 않으므로 GTID 할당이 되지 않습니다.
- GTID 생성
  - 소스 아이디 + 트랜잭션 아이디 값의 조합

    `GTID = [source_id]:[transaction_id]`
  - 소스 아이디는 트랜잭션이 발생된 소스 서버를 식별하기 위한 값으로 mysql 서버의 `server_uuid` `시스템 변수` 값을 사용합니다.
  - 트랜잭션 아이디는 서버에서 커밋된 트랜잭션 순서대로 부여되는 값으로 1부터 1씩 단조 증가합니다.
  - `server_uuid`는 데이터 디렉터리에 `auto.cnf` 파일이 생성되고 그 안에 `server_uuid`값이 저장돼 있습니다.
    - auto.cnf는 삭제돼도 자동으로 생성되지만 UUID값은 소스 서버, 레플리카 서버의 GTID 값에 사용되고 있으므로 삭제되지 않도록 주의해야 합니다.
- GTID 조회
  - mysql.gtid_executed 테이블 or gtid_executed 시스템 변수, SHOW MASTER STATUS에서 GTID확인 가능
  - GTID는 범위로 조회되는 경우가 있는데 이를 GTID sets라 합니다.

    `ed22da00-e052-11ea-ae88-ee4baf89a396:1-5:18:99-103`
  - 서로다른 UUID를 갖는 GTID는 UUID가 변경됐거나 여러 서버에서 데이터를 복제해오는 경우가 해당됩니다.

    `8b20949a-da22-11ea-b0f8-44d14afe1f9d:1-2,ed22da00-e052-11ea-ae88-ee4baf89a396:1-5`
- `mysql.gtid_executed` 테이블은

    - GTID 값 저장 외에도 레플리카 서버에서 바이너리 로그가 비활성화돼 있어도 GTID 기반의 복제를 사용할 수 있게하고, 바이너리 로그가 손실됐을때 GTID 값이 보존될 수 있게 합니다.
    - 데이터가 많아지면 interval_start ~ interval_end를 연속된 것들끼리 모아 1건의 레코드로 압축합니다.
    - 바이너리 로그가 활성화 되면 로테이션시 자동 압축을 합니다.
        - 활성화 안된다면 `gtid_executed_compression_period` 값을 기준으로 압축을 수행하고, 보통때는 슬립 모드를 유지합니다.
        - 위 시스템 변수 값이 0이라면 스레드는 계속 슬립 모드 유지 및 압축을 수행하지 않습니다. (압축은 필요에 따라 자동으로 실행)

---

3. `GTID 기반의 복제 구축`
    - GTID 활성화는 GTID 복제를 위한 하나의 조건임
    - mysql 서버 재시작 없이 GTID 복제 적용 가능

    1. `설정 준비`
        - 복제 참여 mysql 서버들의 GTID가 활성화돼 있어야하고 server_id, server_uuid가 복제 그룹 내에서 고유해야 합니다.

```
## 소스 서버 설정
[mysqld]
gtid_mode=ON
enforce_gtid_consistency=ON
server_id=1111
log_bin=/binary-log-dir-path/binary-log-name

## 레플리카 서버 설정
[mysqld]
gtid_mode=ON
enforce_gtid_consistency=ON
server_id=2222
relay_log=/relay-log-dir-path/relay-log-name
relay_log_purge=ON
read_only
log_slave_updates
```

- `gtid_mode=ON`, `enforce_gtid_consistence=ON`을 반드시 함께 명시해야 함 (안그러면 에러 발생함)
- DBA 계정은 모든 권한이 있을 확률이 높으므로 DDL, DML read_only여도 스키마 변경이 가능합니다. 따라서 super_read_only 옵션도 함께 설정하는게 좋습니다.

2. `복제 계정 준비`

```
CREATE USER 'repl_user'@'%' IDENTIFIED BY 'repl_user_password';
GRANT REPLICATION SLAVE ON *.* TO 'repl_user'@'%';
```

- 호스트는 꼭 필요한 IP 대역을 설정하는게 좋음

3. `데이터 복사`

- mysqldump를 이용함

```
linux> mysqldump -uroot -p --single-transaction --master-data=2 --set-gtid-purged=ON \
       --opt --routines --triggers --hex-blob --all-databases > source_data.sql
```

- 덤프가 시작된 시점의 소스 서버 GTID 값을 레플리카 서버의 관련 시스템 변수 2개에 설정해야 복제 시작 가능

  - `gtid_executed`: MySQL 서버에서 실행되어 바이너리 로그 파일에 기록된 모든 트랜잭션들의 GTID 셋을 나타낸다.
  - `gtid_purged`: 현재 MySQL 서버의 바이너리 로그 파일에 존재하지 않는 모든 트랜잭션들의 GTID 셋을 나타낸다.

  - `gtid_purged`에 값을 설정하면 읽기 전용인 `gtid_executed`에도 값이 설정되므로 소스 서버의 데이터 덤프 시점의 GTID값을 gtid_purged에 설정해야 합니다.
  - 두 시스템 변수를 동일 값으로 설정하려면 두 변수가 비어있어야 하는데 만약 비어있지 않다면 RESET MASTER를 통해 두 변수 값을 초기화하고 설정해야 합니다.
    (RESET MASTER는 바이너리 로그를 모두 삭제하므로 고려해서 사용해야 함)

- mysqldump의 `--set-gtid-purged`

AUTO	덤프를 받는 서버에서 GTID가 활성화돼 있으면 덤프를 시작하는 시점의 GTID 값 및 sql_log_bin 비활성화 구문을 덤프 파일에 기록하며, 만약 GTID가 비활성상태인 서버의 경우 해당 내용들을 기록하지 않는다.
OFF	덤프 시작 시점의 GTID 값 및 sql_log_bin 비활성화 구문을 덤프 파일에 기록하지 않는다.
ON	덤프 시작 시점의 GTID 값 및 sql_log_bin 비활성화 구문을 덤프 파일에 기록한다. 만약 GTID가 활성화돼 있지 않은 서버에서 이 옵션값을 사용하는 경우 에러가 발생한다.
COMMENTED	MySQL 8.0.17 이상 버전부터 사용할 수 있는 값으로, 이 값이 설정되면 ON 값으로 설정됐을 때와 동일하게 동작하되, 덤프 시작 시점의 GTID 값이 주석으로 처리되어 기록된다. sql_log_bin 비활성화 구문은 주석으로 처리되지 않고 다른 경우와 동일하게 바로 적용 가능한 형태로 기록된다.

  - 레플리카 서버에서 새로운 GTID 발급을 막습니다. 원리는 덤프 파일 적재하는 작업이 바이너리 로그에 기록되지 않으므로 GTID가 생성되지 않습니다. (별도 설정 없으면 AUTO임)
  - 다만 A → B 로의 데이터 마이그레이션 작업이라면 B의 바이너리 로그에 데이터 덤프 기록이 남지 않으므로 `B와 관련된 레플리카 서버들에는 데이`

`터가 복제되지 않을 수 있기에 주의` 해야 합니다.

```
linux> less /tmp/source_data.sql
...
SET @@SESSION.SQL_LOG_BIN= 0;

--
-- GTID state at the beginning of the backup
--

SET @@GLOBAL.GTID_PURGED=/*!80000 '+'*/ 'ed22da00-e052-11ea-ae88-ee4baf89a396:1-30';
...
```

- ‘+’는 현재 gtid_purges 시스템 변수에 설정돼 있는 값에 새로운 값을 덧붙임을 의미합니다.

```
mysql_Replica> SHOW GLOBAL VARIABLES LIKE 'gtid_executed';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| gtid_executed |       |
+---------------+-------+

mysql_Replica> SHOW GLOBAL VARIABLES LIKE 'gtid_purged';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| gtid_purged   |       |
+---------------+-------+

mysql_Replica> SOURCE /tmp/source_data.sql;

mysql_Replica> SHOW GLOBAL VARIABLES LIKE 'gtid_executed';
+---------------+-----------------------------------------+
| Variable_name | Value                                   |
+---------------+-----------------------------------------+
| gtid_executed | ed22da00-e052-11ea-ae88-ee4baf89a396:1-30 |
+---------------+-----------------------------------------+

mysql_Replica> SHOW GLOBAL VARIABLES LIKE 'gtid_purged';
+---------------+-----------------------------------------+
| Variable_name | Value                                   |
+---------------+-----------------------------------------+
| gtid_purged   | ed22da00-e052-11ea-ae88-ee4baf89a396:1-30 |
+---------------+-----------------------------------------+
```

- 레플리카 서버의 두 시스템 변수에 값이 자동으로 설정됩니다.
- XtraBackup 툴을 이용한 레플리카 서버에 복구하는 경우는 책을 참조(457page)

4. 복제 시작

```
-- // MySQL 8.0.23 이상 버전
CHANGE REPLICATION SOURCE TO
  SOURCE_HOST='source_server_host',
  SOURCE_PORT=3306,
  SOURCE_USER='repl_user',
  SOURCE_PASSWORD='repl_user_password',
  SOURCE_AUTO_POSITION=1,
  GET_SOURCE_PUBLIC_KEY=1;

-- // MySQL 8.0.23 미만 버전
CHANGE MASTER TO
  MASTER_HOST='source_server_host',
  MASTER_PORT=3306,
  MASTER_USER='repl_user',
  MASTER_PASSWORD='repl_user_password',
  MASTER_AUTO_POSITION=1,
  GET_MASTER_PUBLIC_KEY=1;
```

- 레플리카 서버는 소스 서버에서 `백업 시점 ~ 지금`까지 변경된 데이터와 이후 변경될 데이터를 실시간으로 가져와 적용합니다.
- `SOURCE_AUTO_POSITION` 덕분에 레플리카 서버는 자신의 `gtid_executed` 값을 참조해 해당 시점부터 소스 서버와 복제를 연결해 데이터를 동기화 합니다.

---

4. `GTID 기반 복제에서 트랜잭션 건너뛰기`
   - 레플리카 서버는 소스 서버의 GTID와 내 GTID를 비교해 변경점을 가져오기에 바이너리 로그 위치 기반 복제에서 사용했던 건너뛰기 방식을 사용할 수 없습니다.
   - 따라서 레플리카 서버에서 수동으로 빈 트랜잭션을 생성해 GTID 값을 만들어야 합니다.(Empty Transaction, Dummy Transaction)
   - 자세한 방법은 책을 참조(460 ~ 462p)

---

5. `Non-GTID 기반 복제에서 GTID 기반 복제로 온라인 변경`
    - 5.7.6버전 부터 온라인으로 GTID 모드를 전환할 수 있는 기능을 제공함(Non ↔ GTID 양방향 가능)
    - 두 시스템 변수 `enforce_gtid_consistency`, `gtid_mode`를 서버 재시작 없이 동적으로 값 변경이 가능합니다.
    - `enforce_gtid_consistency`

OFF	GTID 일관성을 해칠 수 있는 쿼리들을 허용
ON	GTID 일관성을 해칠 수 있는 쿼리들을 허용하지 않음
WARN	GTID 일관성을 해칠 수 있는 쿼리들을 허용하지만 그러한 쿼리들이 실행될 때 경고 메시지가 발생함

        - 데이터의 일관성을 해칠 수 있는 쿼리들이 mysql 서버에서 실행 되는걸 허용할지를 제어합니다.
            - 트랜잭션을 지원하는 테이블과 지원하지 않는 테이블을 함께 변경하는 쿼리 혹은 트랜잭션
            - CREATE TABLE ... SELECT ... 구문
            - 트랜잭션 내에서 CREATE TEMPORARY TABLE, DROP TEMPORARY TABLE 구문 사용
        - 위와 같은 쿼리들은 단일 트랜잭션으로 처리된 소스 서버에서 `소스 → 레플리카 복제 시 단일 트랜잭션에서 처리되지 않을 수 있습니다.`
        - 따라서 트랜잭션 단위로 올바르게 할당돼야 복제가 정상동작하는 GTID 기반 복제에선 문제가 됩니다.
        - GTID가 활성화 된 경우에는 반드시 ON이어야 함
    - `gtid_mode`

	신규 트랜잭션	복제된 트랜잭션
OFF	익명 트랜잭션으로 기록됨	익명 트랜잭션만 처리 가능
OFF_PERMISSIVE	익명 트랜잭션으로 기록됨	익명 트랜잭션 및 GTID 트랜잭션 모두 처리 가능
ON_PERMISSIVE	GTID 트랜잭션으로 기록됨	익명 트랜잭션 및 GTID 트랜잭션 모두 처리 가능
ON	GTID 트랜잭션으로 기록됨	GTID 트랜잭션만 처리 가능

        - 바이너리 로그에 트랜잭션들이 GTID 기반으로 로깅될 수 있는 여부 및 트랜잭션 유형별로 mysql 서버에서의 처리 가능 여부를 제어함

- 익명 트랜잭션이 바이너리 로그 파일명 및 위치로 식별됨
- 위 단계는 한번에 한 단계씩 변경 가능합니다.
- 초기엔 소스 서버와 레플리카 서버간 gtid_mode가 동일해도 변경 작업을 진행하면 기존 값 + 새로운 설정 값으로 동작하는 서버가 존재하게 되므로 호환성이 잘 이루어지는지 미리 파악해야 합니다.

표 16.1 gtid_mode별 소스 서버와 레플리카 서버 간 복제 가능 여부 및 자동 포지션 옵션(SOURCE_AUTO_POSITION) 사용 가능 여부

- O: 복제 가능
- X: 복제 불가능
- A: 복제 설정 시 자동 포지션 옵션 사용 가능

	소스 서버 OFF	소스 서버 OFF_PERMISSIVE	소스 서버 ON_PERMISSIVE	소스 서버 ON
레플리카 서버 OFF	O	O	X	X
레플리카 서버 OFF_PERMISSIVE	O	O	O	O + A
레플리카 서버 ON_PERMISSIVE	O	O	O	O + A
레플리카 서버 ON	X	X	O	O + A

- 복제 그룹 내 mysql 서버들이 GTID 모드를 변경하는 과정
  1. 각 서버에 `enforce_gtid_consistencty=WARN`

     모니터링 통해 경고성 로그를 확인해 애플리케이션을 수정하여 메시지 출력이 없도록함
  2. 각 서버에 `enforce_gtid_consistency=ON`

     GTID 사용했을때 안전하게 처리될 수 있는 쿼리만 실행하게 함
  3. 각 서버에 `gtid_mode=OFF_PERMISSIVE`

     신규 트랜잭션은 익명으로 기록되지만, 읽을때 익명, GTID 모두 처리 가능 (다음 단계를 위해 복제 토폴로지에 속한 모든 서버들에게 설정해야 함)
  4. 각 서버에 `gtid_mode=ON_PERMISSIVE`

     기록은 GTID, 여전히 익명, GTID를 모두 처리 가능
  5. 잔여 익명 트랜잭션 확인

     ```
     SHOW GLOBAL STATUS LIKE 'Ongoing_anonymous_transaction_count';
     ```

     Value가 0이라면 다음 단계로 넘어감

6. 각 서버에 `gtid_mode=ON`
7. `my.cnf` 파일에 `gtid_mode`, `enforce_gtid_consistency` 설정을 적용해 재시작해도 설정이 유지되도록 함
8. GTID 기반 복제를 사용하도록 복제 설정을 변경

```
mysql> STOP REPLICA;
mysql> CHANGE REPLICATION SOURCE TO SOURCE_AUTO_POSITION=1;
mysql> START REPLICA;
```

- 비활성화 작업은 위 작업을 역순으로 진행하면 됩니다.

---

6. `GTID 기반 복제 제약 및 변경 사항`

- `제약 사항`
  - GTID가 활성화된 MySQL 서버에는 `enforce_gtid_consistency=ON` 때문에 GTID 일관성을 해칠 수 있는 일부 쿼리들은 실행 불가
- `변경 사항`
  - GTID 기반 복제가 설정된 레플리카 서버에선 `sql_slave_skip_counter` 시스템 변수로 복제된 트랜잭션을 건너뛸 수 없음
    → 레플리카 서버에서 수동으로 빈 트랜잭션을 생성해 GTID 값을 만들어야 함
  - GTID 기반 복제에서 `CHANGE REPLICATION SOURCE TO(or CHANGE MASTER TO)` 구문의 `IGNORE_SERVER_IDS` 옵션은 더이상 사용되지 않음

    해당 옵션은 순환 복제 구조에서 한 서버가 장애로 복제 토폴로지 제외시 장애 서버에서 발생한 이벤트가 중복 적용되지 않을때 유용하게 사용되지만, GTID 사용시 레플리카 서버는 이미 적용된 트랜잭션을 식별할 수 있고 자동으로 무시하므로 필요치 않다고 한다~