4주차

5. INSERT

`INSERT` 문장 자체보다는 테이블 구조가 성능에 더 큰 영향을 미친다. 하지만 `INSERT`와 `SELECT`의 성능을 동시에 빠르게 만들 수 있는 테이블 구조는 거의 없다. 따라서 성능을 타협하며 테이블 구조를 설계해야 한다.

5.1 고급 옵션

1. INSERT IGNORE

저장하는 레코드의 PK나 유니크 인덱스 컬럼의 값이 이미 테이블에 존재하는 레코드와 중복되는 경우나 저장하는 레코드의 컬럼이 테이블의 컬럼과 호환되지 않는 경우 모두 무시하고 다음 레코드를 처리

`IGNORE` 키워드가 있으면 `NOT NULL` 컬럼에 `NULL`을 삽입하는 경우 각 타입별 기본 값을 저장

2. INSERT ... ON DUPLICATE KEY UPDATE

PK나 유니크 인덱스 컬럼의 값 중복이 발생하면 `UPDATE` 문장의 역할을 수행하게 한다.

> `REPLACE` 문장도 비슷하지만 `DELETE`와 `INSERT`의 조합으로 동작함

`ON DUPLICATE KEY UPDATE` 절에는 `GROUP BY` 결과인 `COUNT(*)`를 참조할 수 없기 때문에 `VALUES()`를 사용하면 된다. `VALUES()`는 인자로 받은 컬럼에 `INSERT` 하려고 했던 값을 반환한다.

> 8.0.20 부터는 `VALUES()` 함수를 사용하면 경고 메시지 출력
>
> ```
> Warning (Code 1287): 'VALUES function' is deprecated and will be removed in a future release. Please use an alias (INSERT INTO ... VALUES (...) AS alias) and replace VALUES(col) in the ON DUPLICATE KEY UPDATE clause with alias.col instead
> ```

5.2 LOAD DATA 명령 주의 사항

내부적으로 MySQL 엔진과 스토리지 엔진의 호출 횟수를 최소화하고 스토리지 엔진이 직접 데이터를 적재하도록 하는 명령어(매우 빠름)

단점

- 단일 스레드로 실행
- 단일 트랜잭션으로 실행

데이터의 양이 매우 많다면 단일 스레드로 처리되기 때문에 시간이 매우 길어질 수 있다. 또한 단일 트랜잭션으로 실행되므로 LOAD DATA 문장이 시작된 시점부터 Undo Log가 삭제되지 못하고 유지되어야 한다. 이는 로그 기록 부하도 있지만 로그가 많이 쌓이면 레코드를 읽는 쿼리들이 필요한 레코드를 찾는 데 더 많은 오버헤드를 발생시킨다.

> 여러 개의 파일로 준비하여 LOAD DATA 문장을 동시에 여러 트랜잭션으로 나누어 실행하는 것이 좋다.

5.3 성능을 위한 테이블 구조

1. 대량 INSERT 성능

수백 건, 수천 건의 레코드를 하나의 INSERT 문장으로 처리할 때 레코드들을 PK 값 기준으로 미리 정렬하여 문장을 구성하는 것이 성능에 도움이 될 수 있다.

LOAD DATA 문장이 레코드를 INSERT 할 때마다 InnoDB 스토리지 엔진은 PK를 검색하여 레코드가 저장될 위치를 찾아야 하기 때문이다.

세컨더리 인덱스 여부도 성능에 영향을 준다. 세컨더리 인덱스를 너무 남용하지 말자.

2. PK 선정

INSERT가 매우 많이 실행되는 테이블이라면 테이블의 PK를 단조 증가•감소 패턴의 값을 선택하고 세컨더리 인덱스를 최소화

SELECT가 매우 많이 실행되는 테이블(대부분이 해당)이라면 빈번히 실행되는 SELECT 쿼리의 조건을 기준으로 PK로 선택(세컨더리 인덱스는 자연적으로 많아질 것이다)

> 테이블에 저장될 레코드가 수 백만 건 이하라면 최적화에 너무 많은 시간을 소모하지는 않아도 괜찮다!

3. Auto-Increment 컬럼

INSERT에 최적화된 테이블

- 단조 증가•감소 PK
- 세컨더리 인덱스 최소화

InnoDB 스토리지 엔진 테이블은 자동으로 PK로 클러스터링 되지만 `AUTO_INCREMENT` 컬럼을 이용하면 클러스터링되지 않은 테이블의 효과를 얻을 수 있다. ⇒ MySQL 서버에서 가장 빠른 INSERT 보장

자동 증가 값 채번을 위해서는 AUTO-INC라는 잠금이 필요 (`innodb_autoinc_lock_mode` 시스템 변수로 잠금 사용 방식을 변경 가능)

- `innodb_autoinc_lock_mode = 0`
  - 항상 AUTO-INC를 걸고 한 번에 1씩만 증가된 값을 가져옴
  - 서비스용 MySQL 서버에서는 이 방식을 사용할 필요 없음
- `innodb_autoinc_lock_mode = 1`
  - Consecutive mode
  - 단순히 레코드 한 건씩 INSERT하는 쿼리에서는 AUTO-INC 잠금을 사용하지 않고 뮤텍스를 이용해 더 가볍고 빠르게 처리
  - 여러 레코드를 INSERT하는 쿼리에서는 AUTO-INC 잠금을 걸고 필요한 만큼의 자동 증가 값을 한번에 가져와 사용
  - INSERT 순서대로 증가
- `innodb_autoinc_lock_mode = 2` (기본값)

- Interleaved mode
- LOAD DATA나 벌크 INSERT를 포함한 INSERT 계열의 문장 실행 시 AUTO-INC 잠금을 사용하지 않음
- 자동 증가 값을 적당히 미리 할당받아 처리 (가장 빠른 방식)
- INSERT 순서와 번호 연속 보장 X
- SBR을 사용하는 MySQL에서는 소스 서버와 레플리카 서버의 자동 증가 값이 동기화되지 못할 수 있음

> `LAST_INSERT_ID()`를 사용해 현재 커넥션에서 가장 마지막에 저장된 `AUTO_INCREMENT` 값을 조회할 수 있다.

6. UPDATE와 DELETE

`UPDATE`와 `DELETE`의 작성 방법이나 `WHERE` 절의 인덱스 사용법은 모두 동일하다.

6.1 UPDATE ... ORDER BY ... LIMIT n

`ORDER BY` 절과 `LIMIT`을 사용해 특정 컬럼으로 정렬하여 상위 몇 건만 변경 및 삭제하는 방식으로, 한 번에 많은 레코드를 변경 및 삭제하는 작업을 피할 수 있다.

주의할 점은 바이너리 로그 포맷이 STATEMENT인 경우 복제 소스 서버에서 경고 메시지가 발생한다. ⇒ `ORDER BY`에 의해 정렬되더라도 중복된 값의 순서가 복제 소스 서버와 레플리카 서버에서 달라질 수 있기 때문

6.2 JOIN UPDATE

두 개 이상의 테이블을 조인하여 조인된 결과 레코드를 변경 및 삭제하는 쿼리

조인되는 모든 테이블에 대해 읽기 참조만 되는 테이블은 읽기 잠금이, 컬럼 값이 변경되는 테이블은 쓰기 잠금이 걸리기 때문에 웹 서비스와 같은 OLTP(Online Transaction Processing) 환경에서 데드락 유발 가능성이 높으므로 주의

```
UPDATE departments d, (SELECT de.dept_no, COUNT(*) AS emp_count
                       FROM dept_emp de
                       GROUP BY de.dept_no) dc
SET d.emp_count = dc.emp_count
WHERE dc.dept_no = d.dept_no;
```

JOIN UPDATE 에서는 `GROUP BY`나 `ORDER BY` 절을 사용할 수 없기 때문에 위의 쿼리처럼 서브쿼리를 이용한다.

일반 테이블이 조인될 때는 임시 테이블이 드라이빙 테이블이 되는 것이 일반적으로 빠른 성능을 보여준다. 옵티마이저가 최적의 조인 방향을 알아서 선택하겠지만, 조인 방향을 강제하고 싶다면 `STRAIGHT_JOIN`(또는 8.0에 추가된 `JOIN_ORDER` 옵티마이저 힌트)을 사용하면 된다.

> MySQL :: MySQL 8.0 Reference Manual :: 10.9.3 Optimizer Hints
> One means of control over optimizer strategies is to set the
> optimizer_switch system
> variable (see Section 10.9.2, “Switchable Optimizations”).
> https://dev.mysql.com/doc/refman/8.0/en/optimizer-hints.html#optimizer-hints-join-order

> `STRAIGHT_JOIN` is similar to `JOIN`, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer processes the tables in a suboptimal order.
>
> *MySQL 8.0 Reference Manual*

6.3 여러 레코드 UPDATE

하나의 UPDATE 문장으로 여러 개의 레코드를 업데이트 하는 경우 모든 레코드를 동일한 값으로만 업데이트할 수 있었다. 8.0 부터는 레코드 생성(Row Constructor) 문법을 이용해 레코드별로 서로 다른 값을 업데이트할 수 있다.

```
UPDATE user_level ul
    INNER JOIN (VALUES ROW (1, 1), ROW (2, 4)) new_user_level (user_id, user_lv)
    ON new_user_level.user_id = ul.user_id
SET ul.user_lv=ul.user_lv + new_user_level.user_lv;
```

`VALUES ROW(…), ROW(…), …` 문법을 사용하면 SQL 문장 내에서 임시 테이블을 생성하고 조인하여 업데이트를 수행하는 JOIN UPDATE 문장의 효과를 낼 수 있다.

6.4 JOIN DELETE

단일 테이블의 DELETE 문장과는 조금 다른 문법으로 쿼리를 작성해야 한다.

하나의 테이블에서만 제거

`DELETE`와 `FROM` 절 사이에 삭제할 테이블을 명시해야 한다.

```
DELETE e
FROM employees e,
     dept_emp de,
     departments d
WHERE e.emp_no = de.emp_no
  AND de.dept_no = d.dept_no
  AND d.dept_no = 'd001';
```

여러 테이블에서 제거

`DELETE`와 `FROM` 절 사이에 테이블을 여러 가지 명시하면 된다.

```
DELETE e, de, d
FROM employees e,
     dept_emp de,
     departments d
WHERE e.emp_no = de.emp_no
  AND de.dept_no = d.dept_no
  AND d.dept_no = 'd001';
```

> `STRAIGHT_JOIN` 키워드나 `JOIN_ORDER` 옵티마이저 힌트를 사용할 수 있다.