INSERT Optimization

목표: 이 강의에서는 INSERT 문을 최적화 하는 방법을 배웁니다.

포인트: INSERT 문을 최적화하는 방법과 그 중 LOAD DATA 문에 대해서 이해하는 것이 중요합니다.

▼ Understanding the Cost of INSERT Operations

- **연결 비용 (Connecting):** MySQL 서버와의 연결 비용 (3)
- **쿼리 전송 (Sending Query to Server):** 서버로 쿼리를 보내는 비용 (2)
- **쿼리 파싱 (Parsing Query):** 서버에서 쿼리를 분석하는 비용 (2)
- **행 삽입 (Inserting Row):** 데이터 행을 삽입하는 비용 (1 × size of row)
- **인덱스 삽입 (Inserting Indexes):** 각 인덱스에 대한 삽입 비용 (1 × number of indexes)
- **연결 종료 (Closing):** 연결을 종료하는 비용 (1)

▼ Optimizing INSERT Statements

대량 삽입 (Bulk Insert): 여러 번의 네트워크 통신을 줄이고 한 번에 많은 데이터를 전송.

- **INSERT 문 개선:** 단일 INSERT 문에 여러 값(MULTIPLE VALUES) 삽입으로 처리 속도 개선.
- **LOAD DATA 활용:** 파일에서 직접 데이터를 불러오는 기능으로 처리 속도가 훨씬 빠름.

일관성 검사 지연 (Consistency check delay): 일관성 검사를 미루어 처리 속도 향상.

기본 값을 넣을 땐 INSERT 시 생략하기: INSERT 문 파싱 속도를 더 빠르게 하기 위한 방법.

▼ Considerations for Using LOAD DATA

`LOAD DATA` **Statement 란?**

- 스토로지 엔진에서 지원해주는 Bulk Insert 기능.
- INSERT 문보다 처리속도가 20배 더 빠르다고 함.
- INSERT 문과는 달리 데이터가 들어있는 File 을 줘야한다.

`LOAD DATA` **문을 사용할 때 주의사항:**

- **auto commit 하지 않도록 변경:**
  - 이를 하지 않으면 INSERT 할 때마다 Redo Log 플러쉬가 될 것
- **병렬 처리 고려:**
  - LOAD DATA 문은 단일 스레드 + 단일 트랜잭션으로 처리한다.
  - 트랜잭션이 처리되는 동안에는 Undo Log 지울 수 없는 문제가 발생함.
- **데이터 파일을 전송하는 경우 보안 설정 필요:**
  - `LOAD DATA` 문은 MySQL 서버가 파일로 가지고 있어야 실행됨.
  - 클라이언트 쪽에서 파일을 가지고 있는 경우에는 `LOAD DATA LOCAL` 을 실행해야함.
  - 파일을 전송하기 위해서는 서버/클라이언트 모두 `local_infile=ON` 설정을 활성화 해야함.

▼ Optimizing LOAD DATA

- 파일안의 데이터들을 PK 순으로 정렬:
- UNIQUE KEY Constraint 를 비활성화 (가능하다면):
- FOREIGN KEY Constraint 를 비활성화 (가능하다면):
- `auto_increment_lock_mode = 2` 로 설정하기 (다음 강의에서 자세하게)

▼ Practice

Task 1: Try Using the LOAD DATA SQL Statement

목적: LOAD DATA 문과 일반 INSERT 문과 성능 차이를 비교해보기

시나리오: 애플리케이션에서 사용자의 활동을 추적하고 분석하기 위해, 매일 수백만 건의 사용자 활동 로그를 데이터베이스에 저장해야 하는 상황

테이블: `user_activity_logs`

- `id` (기본 키)
- `user_id` (사용자 식별자)
- `activity_type` (활동 유형)
- `activity_timestamp` (활동 시각)
- `additional_info` (추가 정보)

1. **`user_activity_logs` 테이블 작성**

```
CREATE TABLE user_activity_logs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    activity_type VARCHAR(50),
    activity_timestamp DATETIME,
    additional_info TEXT
);
```

2. **MySQL 컨테이너 `local_infile=ON` 설정하기:**

- **2.1 MySQL 컨테이너에 접속:**

```
docker exec -it mysql-container bash
```

2.2 MySQL 설정 파일인 `my.cnf` 찾기

```
$ find / -name "my.cnf"
```

2.3 my.cnf 파일에 `local_infile=ON` 설정 추가

```
echo "[mysqld]" >> my.cnf

echo "local_infile=ON" >> my.cnf
```

2.4 MySQL 컨테이너 재가동

```
$ docker restart mysql-container
```

2.5 MySQL 에 설정 제대로 되었는지 확인

```
SHOW GLOBAL VARIABLES LIKE 'local_infile';
```

3. Python 코드를 이용해 `LOAD DATA` 문을 통해서 10만건의 데이터 삽입

```
import pymysql
import csv
import os
import random
import time
from datetime import datetime, timedelta

# 데이터베이스 설정
host = 'localhost'
port = 3306
user = 'root'
password = '1234'
database = 'test'

# 데이터베이스 연결 (local_infile 옵션 활성화)
connection = pymysql.connect(host=host, port=port, user=user, password=password, db=database, local_infile=True)

def create_csv_file(file_name, num_records):
    # CSV 파일 생성
    with open(file_name, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        for _ in range(num_records):
            user_id = random.randint(1, 10000)
            activity_type = random.choice(['login', 'logout', 'click', 'view'])
            activity_timestamp = datetime.now() - timedelta(days=random.randint(0, 30))
            additional_info = 'Sample info'
            writer.writerow([user_id, activity_type, activity_timestamp, additional_info])

def load_data_from_csv(file_name):
    with connection.cursor() as cursor:
        query = f"""
        LOAD DATA LOCAL INFILE '{file_name}'
        INTO TABLE user_activity_logs
        FIELDS TERMINATED BY ','
        ENCLOSED BY '"'
        LINES TERMINATED BY '\\n';
        """
        cursor.execute(query)
    connection.commit()

# CSV 파일 생성 및 데이터 로드
csv_file_name = 'user_activity_logs.csv'
num_records = 100000  # 예: 10만 건의

create_csv_file(csv_file_name, num_records)

start_time = time.perf_counter()
load_data_from_csv(csv_file_name)
end_time = time.perf_counter()
elapsed_time = end_time - start_time
print(f"Execution time: {elapsed_time} seconds")

# 연결 닫기
connection.close()
```

4. 테이블의 데이터를 모두 비워준 후 이번에는 `INSERT` 문을 사용해서 10만건의 데이터를 삽입:

```
TRUNCATE TABLE user_activity_logs;
```

```
import pymysql
import random
import time

from datetime import datetime, timedelta

# 데이터베이스 설정
host = 'localhost'
port = 3306
user = 'root'
password = '1234'
database = 'test'

# 데이터베이스 연결
connection = pymysql.connect(host=host, port=port, user=user, password=password, db=database)

def insert_multiple_values(data):
    with connection.cursor() as cursor:
        query = "INSERT INTO user_activity_logs (user_id, activity_type, activity_timestamp, additional_info) VALUES (%s, %s, %s, %
        cursor.executemany(query, data)
    connection.commit()

def generate_data(num_records):
    data = []
    for _ in range(num_records):
        user_id = random.randint(1, 10000)
        activity_type = random.choice(['login', 'logout', 'click', 'view'])
        activity_timestamp = datetime.now() - timedelta(days=random.randint(0, 30))
        additional_info = 'Sample info'
        data.append((user_id, activity_type, activity_timestamp, additional_info))
    return data

# 데이터 생성 및 INSERT 실행
num_records = 100000  # 예: 10만 건의 레코드
data_to_insert = generate_data(num_records)

start_time = time.perf_counter()
insert_multiple_values(data_to_insert)
end_time = time.perf_counter()
elapsed_time = end_time - start_time
print(f"Execution time: {elapsed_time} seconds")

# 연결 닫기
connection.close()
```