11. 쿼리 작성 및 최적화

▼ 목차

- [ ] EXPLAIN 수행 시 결과 컬럼 종류 알아보기
- [ ] 인덱스 공부하기
- [ ] Prepared Statement
- [ ] 내부 임시 테이블?

1. 쿼리 작성 연관 시스템 변수

1.1 SQL 모드

📌 `sql_mode`

> **WARNING**
> 1. MySQL 서버에 사용자 테이블을 생성하고 저장했다면 가능한 한 `sql_mode` 시스템 변수의 값을 변경하지 말자.
> 2. 하나의 복제 그룹에 속한 모든 MySQL 서버는 동일한 `sql_mode` 시스템 변수를 유지하는 것이 좋다.
> 3. `sql_mode` 시스템 변수를 변경해야 하는 경우 MySQL 서버가 자동으로 실행하는 데이터 타입 변환이나 기본값 제어에 영향을 미치는지 확인한 후 적용하는 것이 좋다.

(`SELECT @@GLOBAL.sql_mode;` 쿼리를 사용하여 조회 가능)

- `STRICT_ALL_TABLES` & `STRICT_TRANS_TABLES` (기본): MySQL 서버에서 `INSERT`나 `UPDATE` 문장으로 데이터를 변경하는 경우 칼럼의 타입과 저장되는 값의 타입이 다른 경우 자동으로 타입 변경을 수행한다. 이때 타입이 적절히 변환되기 어렵거나 컬럼에 저장될 값이 없거나 값의 길이가 컬럼의 최대 길이보다 큰 경우 쿼리를 계속 실행할지, 에러를 발생할지를 결정한다.
  - `STRICT_TRANS_TABLES`: InnoDB 처럼 트랜잭션을 지원하는 스토리지 엔진에만 Strict Mode 적용
  - `STRICT_ALL_TABLES`: 트랜잭션 지원 여부와 무관하게 모든 스토리지 엔진에 대해 Strict Mode 적용

- `ANSI_QUOTES`: 홑따옴표(')만 문자열 값 표기로 사용할 수 있도록 설정(쌍따옴표는 컬럼명, 테이블명 등의 식별자 표기)
- `ONLY_FULL_GROUP_BY`: `GROUP BY` 절이 사용된 문장의 `SELECT` 절에는 `GROUP BY` 절에 명시된 컬럼과 집계 함수만 사용 가능
- `PIPE_AS_CONCAT`: `||`를 `OR` 연산이 아닌 문자열 연결 연산자(`CONCAT`)로 사용
- `PAD_CHAR_TO_FULL_LENGTH`: `CHAR` 타입의 컬럼값을 가져올 때 뒤쪽의 공백 문자가 제거되지 않음
- `NO_BACKSLASH_ESCAPE`: 역슬래시를 문자의 이스케이프 문자로 사용하지 않음
- `IGNORE_SPACE`: 프로시저나 함수명, 괄호 사이의 공백 무시
- `REAL_AS_FLOAT`: `REAL` 타입을 `DOUBLE` 대신 `FLOAT`로 설정
- `NO_ZERO_IN_DATE` & `NO_ZERO_DATE`: 날짜에 0 저장 불가능
- `ANSI`: 여러 옵션들을 조합하여 MySQL 서버가 최대한 SQL 표준에 맞게 동작하도록 설정 (`REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, ONLY_FULL_GROUP_BY`)
- `TRADITIONAL`: `STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_ DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_ENGINE_SUBSTITUTION`

1.2 대소문자 구분

Windows에서는 대소문자 구분 X 유닉스 계열 운영체제에서는 대소문자 구분 O

MySQL 서버 설정 파일에 `lower_case_table_names` 시스템 변수를 1로 설정하면 모두 소문자로만 저장되고 대소문자를 구분하지 않는다.

> Windows와 macOS에서 2로 설정하면 저장은 대소문자로 저장하지만 쿼리에서 대소문자를 구분하지 않는다.

그냥 직접 통일해서 저장하자

1.3 MySQL 예약어

백틱(`)을 사용하면 예약어와 같은 키워드를 사용해 테이블명, 컬럼명을 지정할 수 있지만 **되도록 피하자**

2. 메뉴얼의 SQL 문법 표기

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    { {VALUES | VALUE} (value_list) [, (value_list)] ... }
    [AS row_alias[(col_alias [, col_alias] ...)]]
    [ON DUPLICATE KEY UPDATE assignment_list]

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    SET assignment_list
    [AS row_alias[(col_alias [, col_alias] ...)]]
    [ON DUPLICATE KEY UPDATE assignment_list]

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    { SELECT ...
      | TABLE table_name
      | VALUES row_constructor_list
    }
    [ON DUPLICATE KEY UPDATE assignment_list]

value:
    {expr | DEFAULT}

value_list:
    value [, value] ...

row_constructor_list:
    ROW(value_list)[, ROW(value_list)][, ...]

assignment:
    col_name =
          value
        | [row_alias.]col_name
        | [tbl_name.]col_name
        | [row_alias.]col_alias

assignment_list:
    assignment [, assignment] ...
```

- 대문자: **키워드**
- 이탤릭체: **사용자가 선택**하여 작성하는 토큰 (테이블명, 컬럼명, 표현식 등. 아니라면 하단에 알려줌)
- 대괄호([]): 해당 키워드나 표현식은 **선택 사항**
- 파이프(|): 앞, 뒤 키워드나 표현식 중 **하나만 선택**
- 중괄호({}): 괄호 내 아이템 중 **반드시 하나 사용**
- …: 앞에 명시된 키워드나 표현식의 조합이 **반복될 수 있음을 의미**

3. MySQL 연산자와 내장 함수

MySQL은 ANSI 표준 형태가 아닌 연산자들을 제공하지만 되도록 표준 형태 연산자를 쓰자

3.1 리터럴 표기법 문자열

문자열

문자열 표기가 쌍따옴표(")도 가능

문자열 내에 홑따옴표(')를 포함하고 싶으면 홑따옴표X2 하면 됨

오라클이나 PostgreSQL에서 식별자가 키워드와 충돌 시 쌍따옴표 또는 대괄호로 감싸 피하지만 MySQL은 백틱(`ANSI_QUOTES` 설정하면 앞에 친구들처럼 됨)

숫자

```
SELECT * FROM tab_test WHERE string_column = '10000';
```

위 쿼리는 주어진 리터럴이 숫자이므로 컬럼값을 숫자로 변환하여 비교
⇒
`string_column`의 모든 문자열을 숫자로 변환하기 때문에 인덱스가 있어도 활용 불가능
⇒ 알파벳과 같은 숫자로 변환이 불가능한 값인 경우 실패

그래서? 숫자는 숫자 타입 컬럼에만 저장

날짜

MySQL에서는 정해진 형태의 날짜 포맷으로 표기하면 자동으로 `DATE`, `DATETIME`으로 변환함 (복잡하게 `STR_TO_DATE()` 같은 거 안 써도 됨)

```
SELECT * FROM dept_emp WHERE from_date = '2024-01-24';
```

그래서 이 쿼리는 from_date 컬럼값을 문자열로 변환하여 비교하지 않기 때문에 인덱스 이용에 문제 없음!

불리언

`BOOL` == `BOOLEAN` == `TINYINT`

`FALSE` == 0

`TRUE` == 1(1만. 1 이외 포함 아님)

> **무슨 말이지?**
> 모든 숫자 값이 TRUE 나 FALSE 라는 두 개의 불리언 값으로 매핑되지 않는다는 것은 혼란스럽고 애플리케 이션의 버그로 연결됐을 가능성이 크다. 불리언 타입을 꼭 사용하고 싶다면 ENUM 타입으로 관리하는 것이 조금 더 명확하고 실수할 가능성도 줄일 수 있다.

3.2 연산자

동등(Equal) 비교

`=` 또는 `<=>`

`<=>`는 `NULL` 비교까지 수행(NULL-Safe 비교 연산자)

```
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
+-------+-------------+----------+
| 1 = 1 | NULL = NULL | 1 = NULL |
+-------+-------------+----------+
|     1 |        NULL |     NULL |
+-------+-------------+----------+
```

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
+---------+--------------+-----------+
| 1 <=> 1 | NULL <=> NULL | 1 <=> NULL |
+---------+--------------+-----------+
|       1 |            1 |         0 |
+---------+--------------+-----------+
```

부정(Not-Equal) 비교

`<>` 또는 `!=`

하나로 통일하자

NOT 연산자

`!` 또는 `NOT`

AND와 OR 연산자

`AND` 또는 `&&`

`OR` 또는 `||`

`&&, ||`는 다른 용도로 사용 가능할 수 있으니 자제하면 좋음

> `AND` VS `OR`
> 누가 우선순위가 높을까?
> `AND` 승

나누기와 나머지 연산자

`/` 또는 `DIV`

`%` 또는 `MOD`

REGEXP 연산자

문자열 패턴 확인 연산자

`RLIKE` == `REGEXP`

```
${비교 문자열} REGEXP ${정규 표현식}
```

정규 표현식은 POSIX 정규 표현식

POSIX 연산자 - Amazon Redshift

페타바이트 규모의 엔터프라이즈급 완전관리형 데이터 웨어하우징 서비스인 Amazon Redshift로 데이터 웨어하우스를 생성하고 관리합니다.

https://docs.aws.amazon.com/ko_kr/redshift/latest/dg/pattern-matching-conditions-posix.html

> **WARNING**
> REGEXP 조건의 비교는 인덱스 레인지 스캔을 사용할 수 없다! 따라서 WHERE 조건절에 REGEXP 연산자를 단독으로 사용하는 것은 성능상 좋지 않음

LIKE 연산자

이 친구는 인덱스를 이용해 처리가 가능. 대신 단순한 문자열 패턴 비교만 가능

특정 상수 문자열이 있나 없나만 판단

`%`와 `_` 와일드카드 사용 가능

- `%`: 0 또는 1개 이상의 모든 문자
- `_`: 1개의 문자

%나 _ 문자를 비교해야 하면 ESCAPE 절을 LIKE 조건 뒤에 추가하여 설정 가능

```
SELECT 'a%' LIKE 'a/%' ESCAPE '/';
```

> **WARNING**
> 와일드카드 문자가 검색어의 뒤쪽에 있다면 인덱스 레인지 스캔으로 사용 가능하지만 앞쪽에 있으면 사용 불가
> Left-most 특성으로 레인지 스캔을 사용하지 못하고 인덱스를 풀 스캔

BETWEEN 연산자 *좀 어려운 것 같아요 ㅠ

크거나 같다 + 작거나 같다

다른 비교 조건과 결합해 하나의 인덱스를 이용할 때 주의해야 함. ⇒ `BETWEEN`은 범위 연산자이기 때문에 많은 레코드를 읽음

동등 비교 연산을 여러 번 수행하는 IN을 사용하는 게 나을 수 있음 (`BETWEEN 'd003' AND 'd005' AND …` ⇒ `IN ('d003', 'd004', 'd005') AND …`)

IN 연산자

여러 개의 값에 대해 동등 비교 연산을 수행

- 상수가 사용된 경우: `IN (?, ?, ?)`
  - 동등 비교와 동일 ⇒ 빠름
- 서브쿼리가 사용된 경우: `IN (SELECT .. FROM ..)`
  - MySQL8.0 버전 이전에는 IN 절에 튜플을 사용하면 항상 풀 테이블 스캔

NOT IN 연산자가 Primary 키와 비교될 때 가끔 실행 계획에 인덱스 레인지 스캔으로 표시되지만 이는 InnoDB 테이블에서 프라이머리 키가 클러스터링 키이기 때문임 (IN 처럼 효율적이라는 말은 아니다)

3.3 MySQL 내장 함수

NULL 값 비교 및 대체

`IFNULL(expr1, expr2)`: expr1이 NULL이면 expr2 반환. 아닌 경우 expr1 반환

`ISNULL(expr)`: NULL 값 여부

현재 시각 조회

`NOW()` & `SYSDATE()`

모두 현재 시간을 반환하지만 NOW()는 쿼리가 실행되는 시점에 값을 할당 받아 SQL 문장 모든 부분에서 사용하고, SYSDATE()는 함수가 호출될 때마다 다른 값을 반환 ⇒ SYSDATE()와 비교되는 컬럼은 인덱스를 효율적으로 사용할 수 없음

SYSDATE()대신 NOW() 사용

날짜와 시간 포맷

`DATE_FORMAT(date, format)`: DATETIME → 문자열

`STR_TO_DATE(str, format)`: 문자열 → DATETIME

날짜와 시간 연산

`DATE_ADD(date, INTERVAL n [YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, …])`: 날짜 더하기

`DATE_SUB(date, INTERVAL n [YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, …])`: 날짜 빼기

어차피 DATE_ADD()로 빼기까지 가능

타임스탬프 연산

`UNIX_TIMESTAMP([date])`: '1970-01-01 00:00:00'으로부터 경과된 초의 수 반환

`FROM_UNIXTIME(unix_timestamp[, format])`: 인자로 전달된 타임스탬프 값을 DATETIME으로 변환

> MySQL `TIMESTAMP`는 4바이트 숫자 타입으로 '1970-01-01 00:00:01' ~ '2038-01-09 03:14:07' 범위 표현 가능

문자열 처리

`RPAD(str, len, padstr)` & `LPAD(str, len, padstr)`: 문자열 우측 또는 좌측에 문자를 덧붙여 지정된 길이의 문자열로 만드는 함수

`RTRIM(str)` & `LTRIM(str)` & `TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)`: 문자열의 우측 또는 좌측 또는 좌•우측에 연속된 공백 문자(Space, NewLine, Tab) 제거

문자열 결합

`CONCAT(str1, str2, …)`: 여러 문자열을 연결해 하나의 문자열로 반환

`CONCAT_WS(separator, str1, str2, …)`: 구분자를 사용해 여러 문자열을 연결

GROUP BY 문자열 결합

`GROUP_CONCAT(expr)`

COUNT(), MAX, MIN() 등과 같은 Aggregate 함수로, 주로 GROUP BY와 함께 사용됨

컬럼의 값들을 연결하기 위해 제한적인 메모리 버퍼 공간을 사용함. 지정 크기를 초과하면 쿼리에서 경고 메시지가 발생하지만 JDBC로 실행될 때는 에러로 취급되어 실패함 (`group_concat_max_len` 시스템 변수로 조정 가능. 기본은 1KB)

값 비교 및 대체

`CASE WHEN .. THEN .. END`

Java의 switch와 비슷함

```
CASE case_value
    WHEN when_value THEN statemen
    [WHEN when_value THEN stateme
    [ELSE statement_list]
END CASE
```

```
CASE
    WHEN search_condition THEN st
    [WHEN search_condition THEN s
    [ELSE statement_list]
END CASE
```

타입 변환

`CAST(expr AS type [ARRAY])`: 타입 변환

ex)

`CAST(1 - 2 AS UNSIGNED)`

`CONVERT(expr USING transcoding_name)`: 타입 변환 + 문자 집합 변환

ex)

`CONVERT('ABC' USING 'utf8mb4')`

2진수 ↔ 16진수 문자열

`HEX()` : 2진값(BINARY) → 16진수 문자열

`UNHEX()` : 16진수 문자열 → 2진값

암호화 및 해시 함수

MD5, SHA 모두 비대칭 암호화 알고리즘. 문자열을 각각 지정된 비트 수의 해시 값을 생성. 16진수 문자열로 표현

`MD5(str)` : Message Digest 알고리즘 (CHAR(32) 또는 BINARY(16) 필요)

`SHA(str)` : SHA-1 암호화 알고리즘 사용 ⇒ 20byte 해시 값 반환 (CHAR(40) 또는 BINARY(20) 필요)

`SHA2(str, hash_length)` : SHA보다 강력한 28byte ~ 64byte 암호화 알고리즘 사용

이 함수들의 반환값은 중복 가능성이 매우 낮음 ⇒ 길이가 긴 데이터를 줄이고 인덱싱하는 용도로 사용

처리 대기

`SLEEP(duration)` : 멈추고 대기 ㅋㅋ

SQL 개발이나 디버깅 용

레코드 건수만큼 sleep함 (초단위)

벤치마크

`BENCHMARK(count, expr)` :

이것도 디버깅이나 함수 테스트용

`expr` 에 `SELECT` 쿼리를 사용하는 경우 반드시 스칼라값(하나의 컬럼을 가진 하나의 레코드)을 반환해야 사용 가능

> WARNING
>
> SELECT BENCHMARK(10, expr)과 SELECT expr을 직접 10번 실행하는 것은 당연히 차이가 있음. 쿼리 파싱, 최적화, 테이블 락, 네트워크 등 고려해야 함

IP 주소 변환

(IPv4의 경우) 4바이트 unsigned integer로 표현. but 대부분 DBMS는 VARCHAR(15)로 .까지 저장 ⇒ 공간 낭비

`INET_ATON(expr)` : 문자열 IPv4 → unsigned integer

`INET_NTOA(expr)` : 문자열 unsigned integer → IPv4

IPv6용도 있음

JSON Utility 함수

MySQL은 json을 BSON(Binary JSON) 포맷을 사용해 저장함.

`JSON_PRETTY(json_val)`: JSON 이쁘게 포맷해서 반환

`JSON_STORAGE_SIZE(json_val)`: BSON일 때 크기 반환 (byte 단위)

`JSON_EXTRACT(json_doc, path[, path] …)`: JSON 데이터 컬럼 또는 도큐먼트 자체에서 JSON의 특정 필드 값 반환

- `JSON_UNQUOTE()`와 사용하면 큰 따옴표 제거
- `json_doc -> path`도 가능
- `json_doc ->> path`는 `JSON_EXTRACT()`와 `JSON_UNQUOTE()` 조합

`JSON_CONTAINS(target, candidate[, path])`: JSON 필드 포함 여부 확인

`JSON_OBJECT(key, val[, key, val] …)`: JSON 오브젝트 생성

`JSON_OBJECTAGG(key, value)` & `JSON_ARRAYAGG(col_or_expr)`: GROUP BY 절과 함께 사용하여 RDBMS 컬럼의 값들을 모아 JSON 배열 또는 도큐먼트를 생성

`JSON_TABLE(expr, path COLUMNS (column_list))`: JSON 데이터 값들을 모아 RDBMS 테이블을 만들어 반환