RealMySQL8.0 - Chapter11

> 쿼리 작성 및 최적화

- 쿼리 작성과 연관신 시스템 변수
- 메뉴얼의 SQL 문법 표기를 읽는 방법
- MySQL 연산자와 내장 함수
- SELECT
- INSERT
- UPDATE와 DELETE
- 스키마 조작(DDL)
- 쿼리 성능 테스트

> ⚠ 참고
> 애플리케이션 코드를 튜닝해서 성능을 2배 개선한다는 것은 쉽지 않은 일입니다.
> 하지만 DBMS에서 몇십 배 에서 몇백 배의 성능 향상이 이뤄지는 것은 상당히 흔한 일입니다.
> SQL 처리에서 어떻게를 이해하고, 쿼리를 작성하는 것이 그만큼 중요하다는 것입니다.

> DDD, 객체지향코드, 스트림API (성능 이슈)

쿼리 작성과 연관된 시스템 변수

SQL 작성 규칙은 MySQL 서버의 시스템 설정에 따라 달라집니다.

SQL 모드

MySQL 서버의 sql_mode라는 시스템 설정에는 여러 개의 값이 동시에 설정될 수 있습니다.

```
mysql> show variables like 'sql_mode';
+---------------+-----------------------------------------------------------------------------------------------------------------------------------------+
| Variable_name | Value                                                                                                                                   |
+---------------+-----------------------------------------------------------------------------------------------------------------------------------------+
| sql_mode      | ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION |
+---------------+-----------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

- STRICT_ALL_TABLES & STRICT_TRANS_TABLES
  - INSERT, UPDATE 쿼리를 실행시 값의 타입이 다를 때 자동으로 타입 변경
- ANSI_QUOTES
  - 홑따옴표만 문자열 값 표기로 사용할 수 있고, 쌍따옴표는 컬럼명, 테이블명 같은 식별자를 표기하는 데만 사용
- ONLY_FULL_GROUP_BY
  - Group by 절에 포함되지 않은 컬럼이더라도 집합 함수의 사용 없이 그래도 select 절에 having을 사용할 수 있지만 이런 설정을 막는 옵션
- PIPE_AS_CONCAT
  - || 연산자를 문자열 연결 연산자로 사용
- PAD_CHAR_TO_FULL_LENGTH
  - 문자열 뒤의 공백 문자는 제거되어 반환
- NO_BACKSLASH_ESCAPE

    - 역슬래시 문자를 이스케이프 용도로 사용하지 못하도록 설정
- INGNORE_SPACE
    - 스토어드 프로시저나 함수명과 괄호 사이에 있는 공백까지도 스토어드 프로시저나 함수의 이름으로 간주하도록 설정
- REAL_AS_FLOAT
    - REAL이라는 타입이 float 타입의 동의어로 바뀜
- NO_ZERO_IN_DATE & NO_ZERO_DATE
    - 실제로 존재하지 않는 날짜는 저장하지 못하게 설정
- ANSI
    - MySQL 서버가 최대한 sql 표준에 맞게 동작하게 만들어준다.
- TRADITIONAL

영문 대소문자 구분

MySQL 서버는 설치된 운영체제에 따라 테이블명의 대소문자를 구분합니다.

윈도우에 설치된 MySQL에서는 대소문자를 구분하지 않지만 유닉스 계열의 운영체제에서는 대소문자를 구분합니다.

lower_case_table_names는 세 종류의 정수 값으로 해당 설정을 관리합니다.

- 0: 기본값이며 대소문자를 모두 구분합니다.
- 1: 모두 소문자로 저장되고 mysql 서버가 대소문자를 구분하지 않습니다.
- 2: 저장은 대소문자를 구분해서 하지만 MySQL의 쿼리에서는 대소문자를 구분하지 않게 해준다.

```
mysql> show variables like 'lower_case_table_names';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| lower_case_table_names | 2     |
+------------------------+-------+
1 row in set (0.00 sec)
```

문제 예상

대소문자 설정을 바꿀 수 있을까요?

```
mysql> set lower_case_table_names=0;
ERROR 1238 (HY000): Variable 'lower_case_table_names' is a read only variable
```

윈도우 서버로 구상한 데이터베이스 서버에서는 대소문자를 구분하지 않습니다.
만약 윈도우 서버에서 유닉스 계열 운영체제로 데이터를 이관시킬 때 어떻게 하시겠습니까?

메뉴얼의 SQL 문법 표기를 읽는 방법

```
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)] ...
    [ON DUPLICATE KEY UPDATE assignment_list]

value: {expr | DEFAULT}

value_list: value [, value] ...

assignment: col_name = value

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

그림 11.1 MySQL 매뉴얼의 SQL 문법 표기

위 표기법에서 대문자로 표현된 단어는 모두 키워드를 의미합니다.

여기서 대문자는 키워드, 이탤릭체로 표현한 단어는 사용자가 선택해서 작성하는 토큰, 테이블명 또는 표현식을 사용합니다.

sql키워드나 식별자가 아니라면 하다에 추가적인 설명이 주어집니다.

- `대괄호([])` 는 키워드나 표현식 자체가 선택 사항임을 의미함
- `파이프(|)` 는 앞과 뒤의 키워드나 표현식 중에서 단 하나만 선택해서 사용될 수 있음을 의미함
- `중괄호({})` 는 괄호 내의 아이템 중에서 반드시 하나를 사용해야 하는 경우를 의미함
- `...` 표기는 앞에 명시된 키워드나 표현식의 조합이 반복될 수 있음을 의미함

MySQL 연산자와 내장 함수

> ⚠ 알아가기
>
> MySQL에는 다양한 연산자가 있습니다. 여기에는 ANSI표쥰 형태의 연산자가 있지만 표준이 아닌 MySQL만의 고유한연산자가 존재합니다. ASNI 표준이 아닌 연산자는 다른 사용자에게 혼란을 야기할 수 있기 때문에 가능하다면 ANSI 표준형태의 연산자를 사용하길 권장합니다.

리터럴 표기법 문자열

문자열

SQL 표준에서 문자열은 항상 홑따옴표를 사용해서 표시합니다.
MySQL에서는 쌍따옴표를 사용해 문자열을 표기할 수도 있습니다.

```
SELECT * FROM departments WHERE dept_no='d001';
SELECT * FROM departments WHERE dept_no="d001";
```

홑따옴표를 두번 연속 입력하면 문자열에 홑따옴표를 포함할 수 있고 쌍따옴표도 마찬가지로 표현가능하다.

숫자

숫자 값과 문자열 값을 비교할 때는 한 가지 주의할 점이 있습니다.
서로 다른 타입(숫자, 문자열)을 where 조건 비교가 수행되는 경우 자동으로 타입 변환이 발생합니다.

```
mysql>
mysql> select * from employees where emp_no = '463218';
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  |
+--------+------------+------------+-----------+--------+------------+
| 463218 | 1963-11-07 | Oddvar     | Krohm     | M      | 1991-04-11 |
+--------+------------+------------+-----------+--------+------------+
1 row in set (0.00 sec)

mysql> desc employees;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| emp_no     | int           | NO   | PRI | NULL    |       |
| birth_date | date          | NO   |     | NULL    |       |
| first_name | varchar(14)   | NO   | MUL | NULL    |       |
| last_name  | varchar(16)   | NO   | MUL | NULL    |       |
| gender     | enum('M','F') | NO   | MUL | NULL    |       |
| hire_date  | date          | NO   | MUL | NULL    |       |
+------------+---------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

반대의 경우는 일어나지 않는다.

```
mysql> select * from employees where last_name = 'A';
+---------+------------+------------+-----------+--------+------------+
| emp_no  | birth_date | first_name | last_name | gender | hire_date  |
+---------+------------+------------+-----------+--------+------------+
| 1000000 | 1999-07-14 |            | A         | M      | 1999-07-14 |
+---------+------------+------------+-----------+--------+------------+
1 row in set (0.01 sec)

mysql> select * from employees where last_name = 65;
Empty set, 65535 warnings (0.26 sec)
```

이는 MySQL에서 숫자 타입과 문자열 타입 간의 비교에서 숫자 타입을 우선시하기 때문입니다.

날짜

다른 DBMS에서 날짜 타입을 비교하거나 INSERT하려면 문자열을 DATE 타입으로 변환하는 코드가 필요합니다.

하지만 MySQL에서는 정해진 형태의 날짜 포맷으로 표기하면 MySQL 서버가 자동으로 DATE, DATETIME값으로 변환하기 때문에 별도의 변환과정은 필요하지 않습니다.

```
mysql> select * from dept_emp where from_date ='2011-04-29';
Empty set (0.00 sec)

mysql> select * from dept_emp where from_date = STR_TO_DTAE('2011-04-29', '%Y-%m-%d');
```

> 💡 원래 변환과정을 거치는 것 때문에 성능에 있어서 크게 좋을게 없다고 생각했었는데 mysql이 알아서 변환 처리를 해주는구나..!
> oracle에서 database 컬럼은 varchar2(10) yyyy-MM-dd 형태로 저장했는데 안해도 되는구나!

불리언

Bool, Boolean 타입이 있지만 이는 TINYINT 타입에 대한 동의어입니다.
테이블의 칼럼을 생성해보면 tinyint라는 형태인 것을 알 수 있습니다.

(질문)boolean 타입에서 null 허용을 해야 하는 이유가 있을까요?

(질문)boolean 타입의 컬럼을 조회할 때 0, 1로 조회할 수 있을까요?

(질문) jpa 에서 boolean이 들어간 entity 를 생성하면 tinyint로 생성될까요?

MySQL 연산자

동등(equal) 비교(=, <=>)

동등 비교는 다른 DBMS에서와 마찬가지로 = 기호를 사용해 비교를 수행합니다.

mysql에서는 동등 비교를 위해 "<=>" 연산자도 제공하는데 = 연산자와 같으며 null 값에 대한 비교까지 수행합니다.
(NULL-Safe 비교 연산자라고 부르기도 함)

null safe 연산자는 양쪽 모두 null이거나 값이 같은 경우 True를 반환합니다. 반대로 한쪽이 null이라면 false를 반환합니다.

부정(Not-Equal) 비교 (<>, !=)

같지 않다 비교를 위한 연산자 입니다. (<> == !=)
하지만 != 연산자와 같이 사용하게 될 경우 가독성이 떨어지므로 통일시키는 것을 권장합니다.

NOT 연산자 (!)

TRUE 또는 FALSE 연산의 결과를 반대로 만드는 연산자로 NOT을 사용합니다.

AND OR 연산

dbms에서 불리언 표현식의 결과를 결합하기 위해 and, or을 사용합니다.
"&&" 는 and, || => or과 같습니다.

mysql에서 sql_mode 시스템 변숫값에 PIPE_AS_CONCAT을 설정하면 || 연산자를 문자열 결합에 사용할 수 있습니다

단 &&연산자는 불리언 표현식의 결과 결합을 위해 사용할 수 있지만 || 는 사용할 수 없습니다.

```
mysql> SELECT TRUE OR FALSE AND FALSE;
+-------------------------+
| TRUE OR FALSE AND FALSE |
+-------------------------+
|                       1 |
+-------------------------+

mysql> SELECT TRUE OR (FALSE AND FALSE);
+----------------------------+
| TRUE OR (FALSE AND FALSE) |
+----------------------------+
|                          1 |
+----------------------------+

mysql> SELECT (TRUE OR FALSE) AND FALSE;
+----------------------------+
| (TRUE OR FALSE) AND FALSE |
+----------------------------+
|                          0 |
+----------------------------+
```

https://dev.mysql.com/doc/refman/8.0/en/operator-precedence.html

```
INTERVAL
BINARY, COLLATE
!
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
= (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN, MEMBER OF
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
AND, &&
XOR
OR, ||
= (assignment), :=
```

나누기 (/, DIV) 와 나머지(%, MOD) 연산자

나누기 연산자는 일반적으로 알고 있는 / (DIV) 연산자를 사용하고 나머지 또한 %(mod) 연산자를 사용합니다.

(끝)

REGEXP 연산자

문자열 값이 어떤 패턴을 만족하는지 확인하는 연산자입니다.
문자열 값 검사를 위해 정규표현식을 사용해서 문제를 해결합니다. (정규 표현식은 별도로 학습할 것)

- ^: 문자열의 시작을 표시. 정규 표현식은 그 표현식에 일치하는 부분이 문자열의 시작이나 중간 또는 끝부분 어디에 나타나든 관계없지만 "^" 심벌을 표현식의 앞쪽에 넣어주면 일치하는 부분이 반드시 문자열의 제일 앞쪽에 있어야 함을 의미한다.
- $: 문자열의 끝을 표시. "^"와는 반대로 표현식의 끝부분에 "$"를 넣어주면 일치하는 부분이 반드시 문자열의 제일 끝에 있어야 함을 의미한다.
- []: 문자 그룹을 표시. [xyz] 또는 [x-z]라고 표현하면 'x', 'y', 'z' 문자 중 하나인지 확인하는 것이다. 대괄호는 문자열이 아니라 문자 하나와 일치하는지를 확인하는 것이다.
- (): 문자열 그룹을 표시. (xyz)라고 표현하면 세 문자 중 한 문자가 있는지 체크하는 것이 아니라 반드시 "xyz"가 모두 있는지 확인하는 것이다.
- |: "|"로 연결된 문자열 중 하나인지 확인한다. "abc|xyz"라고 표현하면 "abc"이거나 "xyz"인지 확인하는 것이다.
- .: 어떠한 문자든지 1개의 문자를 표시하며, 정규 표현식으로 "..."이라고 표현했다면 3개의 문자(실제 문자의 값과 관계없이)로 구성된 문자열을 찾는 것이다.
- *: 이 기호 앞에 표시된 정규 표현식이 0 또는 1번 이상 반복될 수 있다는 표시다.
- +: 이 기호 앞에 표시된 정규 표현식이 1번 이상 반복될 수 있다는 표시다.
- ?: 이 기호 앞에 표시된 정규 표현식이 0 또는 1번만 올 수 있다는 표시다.

LIKE 연산자

LIKE연산자는 REGEXP와 같이 문자열 패턴 비교가 유효한지 검사하는 연산자입니다.
DBMS에서 REGEXP연산자 보다 LIKE 연산자를 더 자주 사용하는데 이는 like 연산자는 인덱스를 이용할 수 있는 차이가 있습니다.

와일드 카드 연산자

- %: 0 또는 1개 이상의 모든 문자에 일치
- _: 정확히 1개의 문자에 일치

이스케이프 문자

- 백분율 밑줄과 같은 데이터를 구성하는 경우

```
mysql> select '62%' like '62_' escape '%';
+-----------------------------+
| '62%' like '62_' escape '%' |
+-----------------------------+
|                           1 |
+-----------------------------+
1 row in set (0.00 sec)
```

BETWEEN 연산자

between 연산자는 크거나 같다와 작거나 같다라는 두 개의 연산자를 하나로 합친 연산자입니다.

```
mysql> select * from employees where emp_no between 10000 and 10110;
+--------+------------+------------+-------------+--------+------------+
| emp_no | birth_date | first_name | last_name   | gender | hire_date  |
+--------+------------+------------+-------------+--------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello     | M      | 1986-06-26 |
|  10002 | 1964-06-02 | Bezalel    | Simmel      | F      | 1985-11-21 |
|  10003 | 1959-12-03 | Parto      | Bamford     | M      | 1986-08-28 |
|  10004 | 1954-05-01 | Chirstian  | Koblick     | M      | 1986-12-01 |
|  10005 | 1955-01-21 | Kyoichi    | Maliniak    | M      | 1989-09-12 |
|  10006 | 1953-04-20 | Anneke     | Preusig     | F      | 1989-06-02 |
|  10007 | 1957-05-23 | Tzvetan    | Zielinski   | F      | 1989-02-10 |
|  10008 | 1958-02-19 | Saniya     | Kalloufi    | M      | 1994-09-15 |
|  10009 | 1952-04-19 | Sumant     | Peac        | F      | 1985-02-18 |
|  10010 | 1963-06-01 | Duangkaew  | Piveteau    | F      | 1989-08-24 |
|  10011 | 1953-11-07 | Mary       | Sluis       | F      | 1990-01-22 |
|  10012 | 1960-10-04 | Patricio   | Bridgland   | M      | 1992-12-18 |
|  10013 | 1963-06-07 | Eberhardt  | Terkki      | M      | 1985-10-20 |
```

IN 연산자

IN은 여러 개의 값에 대해 동등 비교 연산을 수행하는 연산자입니다.
여러 개의 값이 비교되지만 범위로 검색하는 것이 아니라 여러 번의 동등 비교로 실행하기 때문에 일반적으로 빠르게 처리됩니다.

IN 연산자는 두 형태로 나눠서 생각해야 합니다

- 상수가 사용된 경우의 IN
  - 상수가 사용된 IN의 경우는 동등 비교와 동일하게 작동하기 때문에 매우 빠르게 쿼리가 처리됩니다.
- 서브쿼리가 사용된 경우의 IN
  - 세미조인과 같은 형태의 최적화를 고려해야 함

MySQL 내장 함수

DBMS 종료와 관계없이 기본적인 기능의 SQL 함수는 대부분 동일하게 제공합니다.
하지만 함수의 이름이나 사용법은 표준이 없기 때문에 DBMS별로 호환되지 않습니다.

여기서 MySQL에서는 기본으로 제공하는 **내장 함수**와 사용자가 직접 작성해서 추가하는 **사용자 정의 함수**로 구분됩니다.
c/c++ API를 이용해서 사용자 정의 함수를 추가할 수 있습니다.

단 사용자 정의 함수는 스토어드 프로그램으로 작성되는 프로시저나 스토어드 함수와는 다르므로 혼동하지 않도록 합니다.

NULL값 비교 및 대체 (IFNULL, ISNULL)

IFNULL 함수는 컬럼이나 표현식의 값이 null인지 비교하고 null이면 다른 값으로 대체하는 용도로 사용하는 함수입니다.

- ifnull(null, 1) -> 1 , ifnull(0, 1) -> 0
  ISNULL 함수는 이름 그대로 인자로 전달한 표현식이나 컬럼의 값이 null여부를 비교하는 함수입니다.
- isnull(0) -> 0, isnull(null) -> 1

현재 시각 조회 (now, sysdate)

now, sysdate 함수 모두 현재 시간을 반환하는 함수입니다.
둘의 작동방식에 차이가 있는데 먼저 now함수는 같은 값을 가지지만 sysdate는 호출되는 순간 결과값이 달라집니다.

```
mysql> select now(), sysdate(), sleep(2), now(), sysdate();
+---------------------+---------------------+----------+---------------------+---------------------+
| now()               | sysdate()           | sleep(2) | now()               | sysdate()           |
+---------------------+---------------------+----------+---------------------+---------------------+
| 2024-01-30 01:07:51 | 2024-01-30 01:07:51 |        0 | 2024-01-30 01:07:51 | 2024-01-30 01:07:53 |
+---------------------+---------------------+----------+---------------------+---------------------+
1 row in set (2.00 sec)
```

sysdate 함수는 이런 특징 때문에 두 가지 큰 문제점이 있습니다.

1. sysdate함수가 사용된 sql은 레플리카 서버에서 안정적으로 복제되지 못합니다.
2. sysdate함수와 비교되는 컬럼은 인덱스를 효율적으로 사용하지 못한다(왜?)

(질문) 왜 sysdate함수는 인덱스를 효율적으로 사용하지 못할까?

먼저 앞서 설명한대로 sysdate는 호출되는 순간 값이 결정됩니다.
즉 now와 달리 값을 상수화할 수 없습니다. 즉 서브쿼리로 사용되는 순간 매 쿼리마다 값이 달라지는 문제가 있기 때문입니다.

```
mysql> explain analyze  select * from employees where hire_date < sysdate();
+------------------------------------------------------------------------------------------------------------------------------------
-------+
| EXPLAIN
       |
+------------------------------------------------------------------------------------------------------------------------------------
-------+
| -> Filter: (cast(employees.hire_date as datetime) < sysdate())  (cost=10243 rows=100111) (actual time=0.0729..332 rows=300025 loops=1)
    -> Table scan on employees  (cost=10243 rows=300364) (actual time=0.0656..181 rows=300025 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------
-------+
1 row in set (0.39 sec)

mysql> explain analyze  select * from employees where hire_date < now();
+------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN
+------------------------------------------------------------------------------------------------------------------------------------
| -> Filter: (employees.hire_date < <cache>(now()))  (cost=30269 rows=150182) (actual time=0.0595..242 rows=300025 loops=1)
    -> Table scan on employees  (cost=30269 rows=300364) (actual time=0.0524..189 rows=300025 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.31 sec)
```

실제 tree형태의 결과를 보면 now는 캐싱된 것을 알 수 있지만 sysdate는 캐시가 적용되지 않았습니다.

이를 해결하기 위해서 MySQL 서버의 설정 파일에 sysdate-is-now시스템 변수를 넣어서 활성화하는 것으로 위 문제를 피할 수 있다.

결론은 mysql 서버의 설정 파일에 sysdate-is-now를 추가하는 것 보다 처음부터 now만 사용하는 것이 좋은 방법이라고 생각합니다.

날짜와 시간 포맷(Date_format, str_to_date)

DateTime 타입의 컬럼이나 값을 원하는 형태의 문자열로 반환해야 할 때는 DATE_FORMAT 함수를 이용하면 됩니다.

지정문자	내용
%Y	4자리 연도
%m	2자리 숫자 표시의 월 (01 ~ 12)
%d	2자리 숫자 표시의 일자 (01 ~ 31)
%H	2자리 숫자 표시의 시 (00 ~ 23)
%i	2자리 숫자 표시의 분 (00 ~ 59)
%s	2자리 숫자 표시의 초 (00 ~ 59)

참고로 sql에서 표준형태로 입력된 문자열은 필요한 경우 자동으로 datetime 타입으로 변환되어 처리합니다.
만약 날짜 포맷을 알 수 없는 경우 명시적으로 날짜 타입으로 변환해야 합니다.

날짜와 시간의 연산 (date_add,date_sub)

특정 날짜에 연도나 월일 또는 시간 등을 더하거나 뺄 때는 date_add 함수나 date_sub 함수를 사용합니다.

```
mysql> SELECT DATE_ADD(NOW(), INTERVAL 1 DAY) AS tomorrow;
+---------------------+
| tomorrow            |
+---------------------+
| 2020-08-24 15:11:07 |
+---------------------+

mysql> SELECT DATE_ADD(NOW(), INTERVAL -1 DAY) AS yesterday;
+---------------------+
| yesterday           |
+---------------------+
| 2020-08-22 15:11:07 |
+---------------------+
```

단위는 대표적인 것들만 명시했으며 더 상세한 내용은 매뉴얼을 참조하자.

단위	의미
YEAR	연도(중간의 숫자 값은 더하거나 뺄 연수를 의미함)
MONTH	월(중간의 숫자 값은 더하거나 뺄 개월 수를 의미함)
DAY	일(중간의 숫자 값은 더하거나 뺄 일자 수를 의미함)
HOUR	시(중간의 숫자 값은 더하거나 뺄 시를 의미함)
MINUTE	분(중간의 숫자 값은 더하거나 뺄 분 수를 의미함)
SECOND	초(중간의 숫자 값은 더하거나 뺄 초 수를 의미함)
MICROSECOND	마이크로 초(중간의 숫자 값은 더하거나 뺄 마이크로초 수를 의미함)
QUARTER	분기(중간의 숫자 값은 더하거나 뺄 분기의 수를 의미함)
WEEK	주(중간의 숫자 값은 더하거나 뺄 주 수를 의미함)

타임스탬프 연산(unix_timestamp, from_unixtime)

unix_timestamp 함수는 1970-01-01 00:00:00으로부터 경과된 초의 수를 반환하는 함수입니다.
다른 운영체제나 프로그래밍 언어에서도 같은 방식으로 타임스탬프를 산출하는 경우에는 호환가능한 특징이 있습니다.

사용법

```
mysql> SELECT UNIX_TIMESTAMP();
+------------------+
| UNIX_TIMESTAMP() |
+------------------+
|       1598163535 |
+------------------+

mysql> SELECT UNIX_TIMESTAMP('2020-08-23 15:06:45');
+---------------------------------------+
| UNIX_TIMESTAMP('2020-08-23 15:06:45') |
+---------------------------------------+
|                            1598162805 |
+---------------------------------------+

mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP('2020-08-23 15:06:45'));
+-----------------------------------------------------+
| FROM_UNIXTIME(UNIX_TIMESTAMP('2020-08-23 15:06:45')) |
+-----------------------------------------------------+
| 2020-08-23 15:06:45                                 |
+-----------------------------------------------------+
```

문자열 처리(PRAD, LPAD / RTRIM, LTRIM, TRIM)

- RPAD와 LPAD는 좌측 혹은 우측에 문자를 덧붙여서 지정된 길이의 문자열로 만드는 함수입니다.
- RTRIM, LTRIM는 좌측 혹은 우측에 연속된 공백 문자를 제거하는 함수입니다.

- TRIM은 LTRIM과 RTRIM을 동시에 수행하는 함수입니다.

```
mysql> SELECT RPAD('Cloee', 10, '_');
+------------------------+
| RPAD('Cloee', 10, '_') |
+------------------------+
| Cloee_____             |
+------------------------+

mysql> SELECT LPAD('123', 10, '0');
+----------------------+
| LPAD('123', 10, '0') |
+----------------------+
| 0000000123           |
+----------------------+

mysql> SELECT RTRIM('Cloee ') AS name;
+-------+
| name  |
+-------+
| Cloee |
+-------+

mysql> SELECT LTRIM('     Cloee') AS name;
+-------+
| name  |
+-------+
| Cloee |
+-------+

mysql> SELECT TRIM('     Cloee     ') AS name;
+-------+
| name  |
+-------+
| Cloee |
+-------+
```

문자열 결합 (CONCAT)

여러 개의 문자열을 연결해서 하나의 문자열로 반환하는 함수로, 인자의 개수는 제한이 없습니다.
숫자 값을 인자로 전달하면 문자열 타입으로 자동 변환한 후 연결합니다.

```
mysql> SELECT CONCAT('Georgi','Christian') AS name;
+----------------+
| name           |
+----------------+
| GeorgiChristian |
+----------------+

mysql> SELECT CONCAT('Georgi','Christian',2) AS name;
+-----------------+
| name            |
+-----------------+
| GeorgiChristian2 |
+-----------------+

mysql> SELECT CONCAT('Georgi','Christian',CAST(2 AS CHAR)) AS name;
+-----------------+
| name            |
+-----------------+
| GeorgiChristian2 |
+-----------------+
```

Group by 문자열 결합 (group_concat)

count, max, min, avg 등과 같은 그룹 함수 중 하나입니다. 주로 group by와 함께 사용하며, group by가 없는 sql에서 사용하면 단 하나의 결괏값만 만들어줍니다.

group concat 함수는 값들을 먼저 정렬한 후 연결하거나 각 값의 구분자 설정도 가능하며, 여러 값 중에서 중복을 제거하고 연결하는 것도 가능합니다.

```
mysql> SELECT GROUP_CONCAT(dept_no) FROM departments;
+----------------------------------------------+
¦ GROUP_CONCAT(dept_no)                        ¦
+----------------------------------------------+
¦ d009,d005,d002,d003,d001,d004,d006,d008,d007 ¦
+----------------------------------------------+

mysql> SELECT GROUP_CONCAT(dept_no SEPARATOR '¦') FROM departments;
+----------------------------------------------+
¦ GROUP_CONCAT(dept_no SEPARATOR '¦')          ¦
+----------------------------------------------+
¦ d009¦d005¦d002¦d003¦d001¦d004¦d006¦d008¦d007 ¦
+----------------------------------------------+

mysql> SELECT GROUP_CONCAT(dept_no ORDER BY emp_no DESC)
       FROM dept_emp
       WHERE emp_no BETWEEN 100001 and 100003;
+--------------------------------------------+
¦ GROUP_CONCAT(dept_no ORDER BY emp_no DESC) ¦
+--------------------------------------------+
¦ d005,d008,d008,d005                        ¦
+--------------------------------------------+

mysql> SELECT GROUP_CONCAT(DISTINCT dept_no ORDER BY emp_no DESC)
       FROM dept_emp
       WHERE emp_no BETWEEN 100001 and 100003;
+-----------------------------------------------------+
¦ GROUP_CONCAT(DISTINCT dept_no ORDER BY emp_no DESC) ¦
+-----------------------------------------------------+
¦ d008,d005                                           ¦
+-----------------------------------------------------+
```

(질문) group_concat 함수가 사용하는 메모리 버퍼의 크기가 제한된 범위를 넘어서면?

group_concat 함수는 지정한 컬럼의 값들을 연결하기 위해 제한적인 메모리 버퍼 공간을 사용합니다. 이때 지정된 크기를 초과하면 경고 메시지가 발생하고 해당 결과 중 일부만 보여지고 나머지는 삭제된다.

```
+----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
-------------------------------------------------------+
¦ 10001,10002,10003,10004,10005,10006,10007,10008,10009,10010,10011,10012,10013,10014,10015,10016,10017,10018,10019,10020,10021,10022,10023,10024,10025,10026,10027,10028,10029,10030,10031,10032,10033,10034,10035,10036,10037,10038,1003
9,10040,10041,10042,10043,10044,10045,10046,10047,10048,10049,10050,10051,10052,10053,10054,10055,10056,10057,10058,10059,10060,10061,10062,10063,10064,10065,10066,10067,10068,10069,10070,10071,10072,10073,10074,10075,10076,10077,1007
8,10079,10080,10081,10082,10083,10084,10085,10086,10087,10088,10089,10090,10091,10092,10093,10094,10095,10096,10097,10098,10099,10100,10101,10102,10103,10104,10105,10106,10107,10108,10109,10110,10111,10112,10113,10114,10115,10116,1011
7,10118,10119,10120,10121,10122,10123,10124,10125,10126,10127,10128,10129,10130,10131,10132,10133,10134,10135,10136,10137,10138,10139,10140,10141,10142,10143,10144,10145,10146,10147,10148,10149,10150,10151,10152,10153,10154,10155,1015
6,10157,10158,10159,10160,10161,10162,10163,10164,10165,10166,10167,10168,10169,10170,1017 ¦
+----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
-------------------------------------------------------+
1 row in set, 1 warning (0.36 sec)

mysql> show warnings;
+---------+------+---------------------------------+
¦ Level   ¦ Code ¦ Message                         ¦
+---------+------+---------------------------------+
¦ Warning ¦ 1260 ¦ Row 171 was cut by GROUP_CONCAT() ¦
+---------+------+---------------------------------+
1 row in set (0.00 sec)
```

값의 비교와 대체(case when...then...end)

case when은 함수가 아니라 sql 구문입니다.
switch 와 유사한 역할을 하며 when...then 구문을 필요한 만큼 반복해서 사용할 수 있습니다.

```
mysql> SELECT emp_no, first_name,
         CASE gender WHEN 'M' THEN 'Man'
                     WHEN 'F' THEN 'Woman'
                     ELSE 'Unknown' END AS gender
       FROM employees
       LIMIT 10;
```

```
mysql> SELECT emp_no, first_name,        CASE gender WHEN 'M' THEN 'Man'                WHEN 'F' THEN 'Woman'                ELSE 'Unknown' END AS gender       FROM employees       LIMIT 10;
+--------+------------+--------+
| emp_no | first_name | gender |
+--------+------------+--------+
|  10001 | Georgi     | Man    |
|  10002 | Bezalel    | Woman  |
|  10003 | Parto      | Man    |
|  10004 | Chirstian  | Man    |
|  10005 | Kyoichi    | Man    |
|  10006 | Anneke     | Woman  |
|  10007 | Tzvetan    | Woman  |
|  10008 | Saniya     | Man    |
|  10009 | Sumant     | Woman  |
|  10010 | Duangkaew  | Woman  |
+--------+------------+--------+
10 rows in set (0.00 sec)
```

타입의 변환 (CAST, CONVERT)

Prepared Statement를 제외하면 sql은 텍스트 기반으로 작동하기 때문에 sql에 포함된 모든 입력값은 문자열처럼 취급됩니다. 이럴 때 명시적으로 타입의 변환이 필요한데 cast함수를 사용하면 된다.

cast 함수를 통해 변환할 수 있는 데이터 타입은 다음과 같습니다.

- DATE
- TIME
- DATETIME
- BINARY
- CHAR
- DECIMAL
- SIGNED
- INTEGER
- UNSIGNED
- INTERGE

```
mysql> SELECT CAST('1234' AS SIGNED INTEGER) AS converted_integer;
mysql> SELECT CAST('2000-01-01' AS DATE) AS converted_date;
```

앞부분에 변환할 문자열 그 다음 타입을 선언하면 됩니다.

convert 함수는 cast함수 같이 타입을 변환하는 용도와 문자열의 문자 집합을 변환하는 용도로 사용됩니다.

```
mysql> SELECT CONVERT(1-2 , UNSIGNED);
+-------------------------+
| CONVERT(1-2 , UNSIGNED) |
+-------------------------+
|    18446744073709551615 |
+-------------------------+

mysql> SELECT CONVERT('ABC' USING 'utf8mb4');
+--------------------------------+
| CONVERT('ABC' USING 'utf8mb4') |
+--------------------------------+
| ABC                            |
+--------------------------------+
```

이진값과 16진수 문자열 변환

HEX 함수는 이진값을 사람이 읽을 수 있는 형태의 16진수의 문자열로 변환하는 함수이고, UNHEX함수는 16진수의 문자열을 읽어서 이진값으로 변환하는 함수입니다.

두가지 비대칭형 암호화 알고리즘을 지원합니다

- md5
- sha
- sha2

```
mysql> SELECT MD5('abc');
+----------------------------------+
| MD5('abc')                       |
+----------------------------------+
| 900150983cd24fb0d6963f7d28e17f72 |
+----------------------------------+

mysql> SELECT SHA('abc');
+------------------------------------------+
| SHA('abc')                               |
+------------------------------------------+
| a9993e364706816aba3e25717850c26c9cd0d89d |
+------------------------------------------+
```

```
mysql> SELECT SHA2('abc',256);
+------------------------------------------------------------------+
| SHA2('abc',256)                                                  |
+------------------------------------------------------------------+
| ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad |
+------------------------------------------------------------------+
```

처리대기 (sleep)

sleep 함수는 프로그래밍 언어나 쉘 스크립트 언어에서 제공하는 sleep 기능을 수행합니다.

- sql의 개발 혹은 디버깅 용도로 잠깐 대기하거나 혹은 쿼리의 실행을 오랜 시간 유지하고자 할 때 유용함 (데드락 체크용도?)

초 단위로 인자를 받으며 어떠한 처리를 하거나 반환값을 넘겨주지 않으며 지정한 시간만큼 대기할 뿐입니다.

```
mysql> SELECT SLEEP(1.5)
       FROM employees
       WHERE emp_no BETWEEN 10001 AND 10010;
```

> 1.5초 정도 대기

벤치마크(BENCHMARK)

benchmark함수는 sleep함수와 같이 디버깅이나 간단한 함수의 성능 테스트용으로 유용한 함수입니다.

2개의 인자를 필요로 하는데

1. 반복해서 수행할 횟수
2. 반복해서 실행할 표현식
   - 인자의 표현식은 반드시 스칼라값을 반환해야 함
   - 즉 select 쿼리를 benchmark에서 사용하지만 반드시 스칼라값을 반환하는 select쿼리만 사용해야 함

```
mysql> select benchmark(100000000, (select count(hire_date) from employees where emp_no between 1 and 499999));
+-----------------------------------------------------------------------------------------------------+
| benchmark(100000000, (select count(hire_date) from employees where emp_no between 1 and 499999)) |
+-----------------------------------------------------------------------------------------------------+
|                                                                                                   0 |
+-----------------------------------------------------------------------------------------------------+
1 row in set (4.38 sec)

mysql> select benchmark(100000000, (select count(hire_date) from employees where emp_no >= 1 and emp_no <= 499999)
);
+-----------------------------------------------------------------------------------------------------------+
| benchmark(100000000, (select count(hire_date) from employees where emp_no >= 1 and emp_no <= 499999)) |
+-----------------------------------------------------------------------------------------------------------+
|                                                                                                         0 |
+-----------------------------------------------------------------------------------------------------------+
1 row in set (4.54 sec)
```

JSON 포맷

mysql 클라이언트에서 json 데이터의 기본적인 표시 방법은 단순 텍스트 포맷인데, json 컬럽값에 비해 가독성이 떨어진다.
이때 json_pretty함수를 이용하면 읽기 쉬운 포맷으로 변환해준다.

```
mysql> select doc from employee_docs where emp_no= 10005;
+--------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
| doc
                                                              |
+--------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
| {"emp_no": 10005, "gender": "M", "salaries": [{"salary": 91453, "to_date": "2001-09-09", "from_date": "2000-09-09"}, {"salary": 94692, "to_date": "9999-01-01", "from_date": "2001-09-09"}], "hire_date": "1989-09-12", "last_name": "Ma
liniak", "birth_date": "1955-01-21", "first_name": "Kyoichi"} |
+--------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
1 row in set (0.01 sec)

mysql> select json_pretty(doc) from employee_docs where emp_no= 10005;
+--------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------+
| json_pretty(doc)
                                                                                                                              |
+--------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------+
| {
  "emp_no": 10005,
  "gender": "M",
  "salaries": [
    {
      "salary": 91453,
      "to_date": "2001-09-09",
      "from_date": "2000-09-09"
    },
    {
      "salary": 94692,
      "to_date": "9999-01-01",
      "from_date": "2001-09-09"
    }
  ],
  "hire_date": "1989-09-12",
  "last_name": "Maliniak",
  "birth_date": "1955-01-21",
  "first_name": "Kyoichi"
} |
+--------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------+
```

JSON 필드 크기

JSON 데이터는 텍스트 기반이지만 MySQL 서버는 디스크의 저장 공간을 절약하기 위해 JSON 데이터를 실제 디스크에 저장할 때 BSON 포맷을 사용합니다. 이때 BSON으로 변환됐을 때 저장 크기를 예측할 수 없기에
mysql 서버에서는 json_storage_size함수를 제공합니다.

> 값의 단위는 byte임

```
mysql> select emp_no, json_storage_size(doc) from employee_docs limit 2;
+--------+-------------------------+
| emp_no | json_storage_size(doc) |
+--------+-------------------------+
|  10001 |                     611 |
|  10002 |                     383 |
+--------+-------------------------+
2 rows in set (0.00 sec)
```

JSON 필드 추출

JSON 도큐먼트에서 특정 필드의 값을 가져오는 방법은 여러가지가 있습니다.

가장 일반적인 방법은 JSON_EXTRACT 함수를 사용하는 것인데 JSON_EXTRACT 함수는 2개의 인자를 필요로 하는데,

1. JSON 데이터가 저장된 컬럼 또는 JSON 도큐먼트 자체
2. 가져오고자 하는 JSON 경로

```
mysql> SELECT emp_no, JSON_EXTRACT(doc, "$.first_name") FROM employee_docs;
+--------+----------------------------------+
| emp_no | JSON_EXTRACT(doc, "$.first_name") |
+--------+----------------------------------+
|  10001 | "Georgi"                         |
|  10002 | "Bezalel"                        |
|  10003 | "Parto"                          |
|  10004 | "Christian"                      |
|  10005 | "Kyoichi"                        |
+--------+----------------------------------+
```

```
mysql> SELECT emp_no, JSON_UNQUOTE(JSON_EXTRACT(doc, "$.first_name")) FROM employee_docs;
+--------+--------------------------------------------------+
| emp_no | JSON_UNQUOTE(JSON_EXTRACT(doc, "$.first_name")) |
+--------+--------------------------------------------------+
|  10001 | Georgi                                           |
|  10002 | Bezalel                                          |
|  10003 | Parto                                            |
|  10004 | Christian                                        |
|  10005 | Kyoichi                                          |
+--------+--------------------------------------------------+
```

아래 방법을 사용해서 JSON_EXTRACT, JSON_UNQUOTE 함수 없이 가져올 수 있습니다.

```
mysql> SELECT emp_no, doc->"$.first_name" FROM employee_docs LIMIT 2;
+--------+---------------------+
| emp_no | doc->"$.first_name" |
+--------+---------------------+
|  10001 | "Georgi"            |
|  10002 | "Bezalel"           |
+--------+---------------------+
```

따옴표 제거는 ->> 연산자를 통해 제거가 가능합니다.

JSON 오브젝트 포함 여부 확인(JSON_CONTAINS)

JSON 도큐먼트 또는 지정된 JSON 경로에 JSON 필드를 가지고 있는지 확인하는 함수입니다.

```
mysql> SELECT emp_no FROM employee_docs
       WHERE JSON_CONTAINS(doc, '{"first_name":"Christian"}');
+--------+
| emp_no |
+--------+
|  10004 |
+--------+

mysql> SELECT emp_no FROM employee_docs
       WHERE JSON_CONTAINS(doc, '"Christian"', '$.first_name');
+--------+
| emp_no |
+--------+
|  10004 |
+--------+
```

JSON 오브젝트 생성(JSON_OBJECT)

RDBMS 컬럼의 값을 이용해 JSON 오브젝트를 생성하는 함수입니다.

```
mysql> SELECT
         JSON_OBJECT("empNo", emp_no,
                     "salary", salary,
                     "fromDate", from_date,
                     "toDate", to_date) AS as_json
       FROM salaries LIMIT 3;

+-----------------------------------------------------------------------------------------+
| as_json                                                                                 |
+-----------------------------------------------------------------------------------------+
| {"empNo": 10001, "salary": 60117, "toDate": "1987-06-26", "fromDate": "1986-06-26"} |
| {"empNo": 10001, "salary": 62102, "toDate": "1988-06-25", "fromDate": "1987-06-26"} |
| {"empNo": 10001, "salary": 66074, "toDate": "1989-06-25", "fromDate": "1988-06-25"} |
+-----------------------------------------------------------------------------------------+
```

JSON 컬럼으로 집계(JSON_OBJECTAGG & JSON_ARRAYGG)

json_objectagg, json_arrayagg 함수는 group by절과 함께 사용되는 집계 함수로 RDBMS 컬럼의 값들을 모아 JSON 배열 또는 도큐먼트를 생성하는 함수입니다.

```
mysql> SELECT dept_no, JSON_OBJECTAGG(emp_no, from_date) AS agg_manager
       FROM dept_manager
       WHERE dept_no IN ('d001','d002','d003')
       GROUP BY dept_no;
+---------+-----------------------------------------------------+
| dept_no | agg_manager                                         |
+---------+-----------------------------------------------------+
| d001    | {"110022": "1985-01-01", "110039": "1991-10-01"} |
| d002    | {"110085": "1985-01-01", "110114": "1989-12-17"} |
| d003    | {"110183": "1985-01-01", "110228": "1992-03-21"} |
+---------+-----------------------------------------------------+
```

JSON 데이터를 테이블로 변환 (JSON_TABLE)

JSON_TABLE() 함수는 JSON 데이터의 값들을 모아서 RDBMS 테이블을 만들어 반환합니다.

```
mysql> SELECT e2.emp_no, e2.first_name, e2.gender         FROM employee_docs e1,            JSON_TABLE(doc, "$" COLUMNS (emp_no INT PATH "$.emp_no",                                  gender CHAR(1) PATH "$.gender",
                        first_name VARCHAR(20) PATH "$.first_name")                    ) AS e2       WHERE e1.emp_no IN (10001, 10002);
+--------+------------+--------+
| emp_no | first_name | gender |
+--------+------------+--------+
|  10001 | Georgi     | M      |
|  10002 | Bezalel    | F      |
+--------+------------+--------+
2 rows in set (0.00 sec)

mysql>
mysql> explain SELECT e2.emp_no, e2.first_name, e2.gender         FROM employee_docs e1,            JSON_TABLE(doc, "$" COLUMNS (emp_no INT PATH "$.emp_no",                                  gender CHAR(1) PATH "$.gender",
                        first_name VARCHAR(20) PATH "$.first_name")                    ) AS e2       WHERE e1.emp_no IN (10001, 10002);
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-----------------------------------------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref  | rows | filtered | Extra                                         |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-----------------------------------------------+
|  1 | SIMPLE      | e1    | NULL       | range | PRIMARY       | PRIMARY | 4       | NULL |    2 |   100.00 | Using where                                   |
|  1 | SIMPLE      | e2    | NULL       | ALL   | NULL          | NULL    | NULL    | NULL |    2 |   100.00 | Table function: json_table; Using temporary |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-----------------------------------------------+
2 rows in set, 1 warning (0.00 sec)
```

단 JSON_TABLE 함수는 내부 임시 테이블을 이용하기 때문에 임시 테이블에 레코드가 많이 저장되지 않게 주의해야 합니다.