Oracle And Mysql SQL

Comparison Guide

Author	최명기
Creation Date	
Last Updated	
Version	1.0

Version	변경일자	변경자(작성자)	주요내용
1			

Contents

1. 개요

DBMS 에는 여러 종류가 있고 아키텍처는 서로 상이합니다. 따라서 관리방법과 접근 방법이 다릅니다. 이번 기술노트에서는 DBMS 의 여러 관리 방법 중 Oracle 과 MySQL 의 SQL 비교에 대해서 설명 하겠습니다.

2. Oracle vs Mysql syntax

2.1. 대소문자

MySQL 의 Database 는 데이터 디렉토리에 있는 디렉토리와 매칭되고 table 은 파일과 매칭됩니다. SQL 을 통해 값을 얻고자 할 때, 운영체제의 디렉토리와 파일을 찾아가게 됩니다. 따라서 MySQL 의 Identifier 은 기본 운영체제 환경의 대소문자를 따라갑니다. Window 에서는 대소문자를 구분하지 않고 Linux, Unix 에서는 대소문자를 구분하게 됩니다. Unix 기반이지만 대소문자를 구분하지 않는 기본 파일 시스템 유형 (HFS +)을 사용하는 macOS 는 예외입니다. 그러나 macOS 역시 UFS 를 지원하며, UFS 는 대소문자를 구분합니다. lower_case_table_names 시스템 변수 역시 서버가 Identifier 의 대소문자를 처리하는 동작에 영향을 줍니다.

데이터 베이스 및 테이블 이름이 어떤 플랫폼에서는 대소 문자를 구분하지 않는다 하더라도, 동일한 명령문 내에서는 데이터 베이스 및 테이블 이름을 대소 문자를 혼용해서 지정하지 않도록 해야 합니다. 아래의 명령문은 하나의 동일 테이블을 my_table 및 MY_TABLE 로 참조하기 때문에 동작을 하지 않게 되는 예시입니다.

```
mysql>    SELECT * FROM my_table WHERE MY_TABLE.col=1;
ERROR 1054 (42S22): Unknown column ' MY_TABLE.col ' in 'where clause'
```

Column, index, stored routine, and event names 은 모든 플랫폼에서 대소 문자를 구분하지 않으며, column aliases 도 마찬가지입니다.

유닉스에서는 테이블 별칭의 대소 문자를 default 로 구분하지만, 윈도우 또는 macOS 에서는 그렇지 않습니다. 아래의 명령문은 유닉스에서는 동작을 하지 않는데, 그 이유는 이것이 a 및 A 형태로 별칭을 참조하기 때문입니다

```
mysql> SELECT col_name FROM tbl_name AS a
        WHERE a.col_name = 1 OR A.col_name = 2;
```

그러나 Windows 에서는 동일한 내용이 허용됩니다. 이러한 차이로 인한 문제점을 피하려면 항상 소문자 이름을 사용하여 데이터베이스 및 테이블을 작성하고 참조하는 일관된 규칙을 채택하는 것이 가장 좋습니다. 이 방식이 이식성(portability)을 최대한 보장해 주고 사용을 용이하게 해주기

때문에 권장하고 있습니다.

MySQL 에서 테이블 및 데이터베이스 이름이 디스크에 저장되고 사용되는 방식은 mysqld 를 시작할 때 lower_case_table_names 를 설정할 수 있는 시스템 변수의 영향을 받습니다. Unix 의 기본값은 0 이며, Windows 는 1, macOS 는 2 의 기본값을 가집니다

Value	Meaning
0	CREATE TABLE 이나 CREATE DATABASE 실행시 디스크에 저장되는 테이블과 데이타베이스의 이름을 대소문자를 구분해서 생성합니다. SELECT 나 Insert 사용시에도 대소문자를 구분해서 사용해야 합니다. 대소문자를 구별하는 OS 에서만 의미가 있고 Windows/Mac OS X 에는 적용되지 않습니다 대소문자를 구분하지 않는 파일시스템 MyISAM 에서 이 변수를 강제로 사용하고 대소문자 구분없이 테이블에 엑세스 하면 인덱스가 손상 될 수 있습니다.
1	테이블과 DB 이름을 소문자로 생성하며 참조시에는 소문자로 변경하여 처리합니다. 기존에 대문자가 포함되어 생성한 테이블과 DB 는 문제가 될 수 있습니다.
2	CREATE TABLE 이나 CREATE DATABASE 실행시 디스크에 저장되는 테이블과 데이타베이스의 이름을 대소문자를 구분해서 생성합니다. 참조시에는 소문자로 변경합니다. 대소문자를 구분하지 않는 파일 시스템을 가진 OS(Mac OS X) 에서만 동작합니다.

하나의 플랫폼 상에서만 MySQL 을 사용한다면, lower_case_table_names 변수를 변경 시킬 필요는 없습니다. 하지만, 파일 시스템이 대소 문자를 구분하는 플랫폼 간에 테이블을 전달하고자 하는 경우에는 어려움을 직면할 수도 있게 됩니다. 예를 들면, 유닉스 상에서는 이름이 my_table 과 MY_TABLE 인 두 개의 테이블을 가질 수는 있으나, 윈도우에서는 이 두 개의 테이블이 동일한 것으로 간주됩니다. 문자 크기에 민감한 테이블과 데이터 베이스 이름 전송 문제를 해결하기 위해서는, 아래의 두 가지 옵션을 사용해야 합니다.

- 모든 시스템에서 lower_case_table_names=1 로 설정을 합니다. 이렇게 할 경우의 가장 큰 단점은, SHOW TABLES 또는 SHOW DATABASES 를 사용할 경우에, 이것들의 원래 문자 크기로 된 이름을 볼 수 없다는 점입니다.
- 유닉스에서는 lower_case_table_names=0 로 설정하고, 윈도우에서는 lower_case_table_names=2 로 설정합니다. 이렇게 하면, 데이터 베이스 및 테이블 이름의 원래 문자 크기가 보존됩니다. 이렇게 할 경우의 단점으로는, 여러분이 사용하는 명령문이 항상 윈도우에서 정확한 문자 크기를 사용해서 데이터 베이스 및 테이블 이름을 참조하는 것을확인을 해야 한다는 것입이다. 만일 명령문을 문자 크기에 민감한 유닉스로 전송할 경우에는,

문자 크기가 정확하지 않는 명령문은 동작을 하지 않게 됩니다.

Exception: 만일 InnoDB 테이블을 사용한다면, 모든 플랫폼 상에서 lower_case_table_names 를 1 로 설정해서 이름을 강제로 소문자로 변환하도록 해야 합니다.

만일 유닉스에서 lower_case_table_names 시스템 변수를 1 로 설정하고자 한다면, 우선 구형 데이터 베이스 및 테이블 이름을 소문자로 변경한 후에 mysqld 을 새로운 변수 값으로 재 시작해야 합니다.

2.2. 공백문자

MySQL 은 CHAR 및 VARCHAR 타입은 서로 비슷하지만, 저장 및 추출하는 방식에서 서로 차이가 있습니다. CHAR 및 VARCHAR 타입은 여러분이 저장하고자 하는 문자의 최대 숫자를 지정하는 길이와 함께 선언됩니다. 예를 들면, CHAR(30)은 30 개의 문자를 저장할 수가 있습니다.

CHAR 컬럼의 길이는 여러분이 테이블을 생성할 때 선언한 길이에 고정이 됩니다. 그 길이는 0 에서 255 사이의 값을 가집니다. CHAR 값을 저장할 때에는, 지정된 길이를 맞추기 위해서 오른쪽에 스페이스를 집어 넣게 됩니다. CHAR 값을 추출할 때 PAD_CHAR_TO_FULL_LENGTHSQL 모드를 사용 하지 않으면 추가된 스페이스가 제거됩니다.

VARCHAR 열의 값 은 가변 길이 문자열입니다. 길이는 0 에서 65,535 사이의 값으로 지정할 수 있습니다. CHAR 와는 반대로, VARCHAR 값은 필요한 문자 수 만큼만을 사용해서 저장되며, 여기에 길이를 기록하는 1 바이트가 추가 됩니다. (255 보다 긴 길이로 선언된 컬럼에 대해서는 2 바이트가 추가 됩니다.).

아래의 테이블은 CHAR 및 VARCHAR 간의 차이점에 대해서 예시를 하고 있습니다.

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Oracle vs MySQL (CHAR, VARCHAR)

```
##Oracle
SQL> create table test
  2  (c1 char(5), c2 char(5), v1 varchar(5), v2 varchar(5),
  3  PRIMARY KEY(v2));
Table created.

SQL> insert into test values ('a', 'a ', 'a', 'a ');
SQL> insert into test values ('a', 'a ', 'a', 'a    ');

SQL> select '('||c1||')','('||c2||')','('||v1||')','('||v2||')' from test;
'('||C1 '('||C2 '('||V1 '('||V2
------- ------- ------- -------
(a    ) (a    ) (a)    (a )
(a    ) (a    ) (a)    (a    )

SQL> select length(c1), length(c2), length(v1), length(v2) from test;
LENGTH(C1) LENGTH(C2) LENGTH(V1) LENGTH(V2)
---------- ---------- ---------- ----------
         5          5          1          2
         5          5          1          5

SQL> select length(c1), length(c2), length(v1), length(v2) from test where c1='a';
LENGTH(C1) LENGTH(C2) LENGTH(V1) LENGTH(V2)
---------- ---------- ---------- ----------
         5          5          1          2
         5          5          1          5

SQL> select length(c1), length(c2), length(v1), length(v2) from test where c1='a    ';
LENGTH(C1) LENGTH(C2) LENGTH(V1) LENGTH(V2)
---------- ---------- ---------- ----------
         5          5          1          2
         5          5          1          5
** c1 = 'a'와 'a    '의 결과값이 같다.



SQL> select length(c1), length(c2), length(v1), length(v2) from test where c1 like 'a';
```

```
no rows selected

SQL> select length(c1), length(c2), length(v1), length(v2) from test where c1 like 'a    ';
LENGTH(C1) LENGTH(C2) LENGTH(V1) LENGTH(V2)
---------- ---------- ---------- ----------
         5          5          1          2
         5          5          1          5
** c1 like 'a'와 'a    '의 결과값이 다르다.


SQL> select length(c1), length(c2), length(v1), length(v2) from test where v2='a';
no rows selected

SQL> select length(c1), length(c2), length(v1), length(v2) from test where v2='a    ';
LENGTH(C1) LENGTH(C2) LENGTH(V1) LENGTH(V2)
---------- ---------- ---------- ----------
         5          5          1          5
** v2 = 'a'와 'a    '의 결과값이 다르다


SQL> select length(c1), length(c2), length(v1), length(v2) from test where v2 like 'a';
no rows selected

SQL> select length(c1), length(c2), length(v1), length(v2) from test where v2 like 'a    ';
LENGTH(C1) LENGTH(C2) LENGTH(V1) LENGTH(V2)
---------- ---------- ---------- ----------
         5          5          1          5
** v2 like 'a'와 'a    '의 결과값이 다르다.



##Mysql 8.0
mysql> create table test
    -> (c1 char(5), c2 char(5), v1 varchar(5), v2 varchar(5),
    -> PRIMARY KEY(v2));

mysql> insert into test values
```

```
    -> ('a', 'a ', 'a', 'a '),
    -> ('a', 'a ', 'a', 'a    ');

mysql> SET sql_mode = 'PAD_CHAR_TO_FULL_LENGTH';

mysql> select concat('(',c1,')'),concat('(',c2,')'),concat('(',v1,')'),concat('(',v2,')')
from test;
+--------------------+--------------------+--------------------+--------------------+
| concat('(',c1,')') | concat('(',c2,')') | concat('(',v1,')') | concat('(',v2,')') |
+--------------------+--------------------+--------------------+--------------------+
| (a    )            | (a    )            | (a)                | (a )                |
| (a    )            | (a    )            | (a)                | (a    )            |
+--------------------+--------------------+--------------------+--------------------+
2 rows in set (0.00 sec)


mysql> select length(c1), length(c2), length(v1), length(v2) from test;
+-----------+-----------+-----------+-----------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+-----------+-----------+-----------+-----------+
|          5 |          5 |          1 |          2 |
|          5 |          5 |          1 |          5 |
+-----------+-----------+-----------+-----------+
2 rows in set (0.00 sec)


mysql> select length(c1), length(c2), length(v1), length(v2) from test where c1='a';
Empty set (0.00 sec)

mysql> select length(c1), length(c2), length(v1), length(v2) from test where c1='a    ';
+-----------+-----------+-----------+-----------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+-----------+-----------+-----------+-----------+
|          5 |          5 |          1 |          2 |
|          5 |          5 |          1 |          5 |
+-----------+-----------+-----------+-----------+
2 rows in set (0.00 sec)
```

```
** c1 = 'a'와 'a    '의 결과값이 다르다.

mysql> select length(c1), length(c2), length(v1), length(v2) from test where c1 like 'a';
Empty set (0.00 sec)

mysql> select length(c1), length(c2), length(v1),length(v2) from test where c1 like 'a    ';
+-----------+-----------+-----------+-----------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+-----------+-----------+-----------+-----------+
|          5 |          5 |          1 |          2 |
|          5 |          5 |          1 |          5 |
+-----------+-----------+-----------+-----------+
2 rows in set (0.00 sec)
** c1 like 'a'와 'a    '의 결과값이 다르다.

mysql> select length(c1), length(c2), length(v1), length(v2) from test where v2='a';
Empty set (0.00 sec)

mysql> select length(c1), length(c2), length(v1), length(v2) from test where v2='a    ';
+-----------+-----------+-----------+-----------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+-----------+-----------+-----------+-----------+
|          5 |          5 |          1 |          5 |
+-----------+-----------+-----------+-----------+
1 row in set (0.00 sec)
** v2 = 'a'와 'a    '의 결과값이 다르다.

mysql> select length(c1), length(c2), length(v1), length(v2) from test where v2 like 'a';
Empty set (0.00 sec)

mysql> select length(c1), length(c2), length(v1),length(v2) from test where v2 like 'a    ';
+-----------+-----------+-----------+-----------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+-----------+-----------+-----------+-----------+
|          5 |          5 |          1 |          5 |
+-----------+-----------+-----------+-----------+
1 row in set (0.00 sec)
```

** v2 like 'a'와 'a   '의 결과값이 다르다.

##Mysql 5.7

```
mysql> create table test
    -> (c1 char(5), c2 char(5), v1 varchar(5), v2 varchar(5),
    -> PRIMARY KEY(v2));
Query OK, 0 rows affected (0.01 sec)

mysql> insert into test values
    -> ('a', 'a ', 'a', 'a '),
    -> ('a', 'a ', 'a', 'a    ');
        ERROR 1062 (23000): Duplicate entry 'a ' for key 'PRIMARY'
        (5.7 에서는 'a ' 와 'a    '에서 Duplicate Error 가 발생)

mysql> alter table test drop primary key;

mysql> insert into test values
    -> ('a', 'a ', 'a', 'a '),
    -> ('a', 'a ', 'a', 'a    ');

mysql> SET sql_mode = 'PAD_CHAR_TO_FULL_LENGTH';

mysql> select length(c1), length(c2), length(v1), length(v2) from test where c1='a';
+------------+------------+------------+------------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+------------+------------+------------+------------+
|          5 |          5 |          1 |          2 |
|          5 |          5 |          1 |          5 |
+------------+------------+------------+------------+
2 rows in set (0.00 sec)

mysql> select length(c1), length(c2), length(v1), length(v2) from test where c1='a    ';
+------------+------------+------------+------------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+------------+------------+------------+------------+
|          5 |          5 |          1 |          2 |
|          5 |          5 |          1 |          5 |
```

```
+-----------+-----------+-----------+-----------+
2 rows in set (0.00 sec)
** c1 = 'a'와 'a    '의 결과값이 같다.

mysql> select length(c1), length(c2), length(v1), length(v2) from test where c1 like 'a';
Empty set (0.00 sec)

mysql> select length(c1), length(c2), length(v1),length(v2) from test where c1 like 'a    ';
+-----------+-----------+-----------+-----------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+-----------+-----------+-----------+-----------+
|          5 |          5 |          1 |          2 |
|          5 |          5 |          1 |          5 |
+-----------+-----------+-----------+-----------+
2 rows in set (0.00 sec)
** c1 like 'a'와 'a    '의 결과값이 다르다.

mysql> select length(c1), length(c2), length(v1), length(v2) from test where v2='a';
+-----------+-----------+-----------+-----------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+-----------+-----------+-----------+-----------+
|          5 |          5 |          1 |          2 |
|          5 |          5 |          1 |          5 |
+-----------+-----------+-----------+-----------+
2 rows in set (0.00 sec)

mysql> select length(c1), length(c2), length(v1), length(v2) from test where v2='a    ';
+-----------+-----------+-----------+-----------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+-----------+-----------+-----------+-----------+
|          5 |          5 |          1 |          2 |
|          5 |          5 |          1 |          5 |
+-----------+-----------+-----------+-----------+
2 row in set (0.00 sec)
** v2 = 'a'와 'a    '의 결과값이 같다.

mysql> select length(c1), length(c2), length(v1), length(v2) from test where v2 like 'a';
```

```
Empty set (0.00 sec)

mysql> select length(c1), length(c2), length(v1),length(v2) from test where v2 like 'a    ';
+------------+------------+------------+------------+
| length(c1) | length(c2) | length(v1) | length(v2) |
+------------+------------+------------+------------+
|          5 |          5 |          1 |          5 |
+------------+------------+------------+------------+
1 row in set (0.00 sec)
```

** v2 like 'a'와 'a   '의 결과값이 다르다.

##종합

비교	Oracle	MySQL 8.0	MySQL 5.7
c1 = 'a' , 'a '	eq	neq	eq
c1 like 'a' , 'a '	neq	neq	neq
v2 = 'a', 'a '	neq	neq	eq
v2 like 'a', 'a '	neq	neq	neq

3. Function

3.1. NVL / IFNULL

Function	NVL (Oracle)	IFNULL (MySQL)
Example	SELECT NVL(col1, 'A') nvl_col, col2 FROM *table*	SELECT IFNULL(col1, 'A') nvl_col, col2 FROM *table*

NVL(expr1, expr2) <-> IFNULL(expr1, expr2)

만일 expr1 이 NULL 이 아니라면, 결과 값은 expr1 을 리턴합니다. 그렇지 않을 경우에는 expr2 를 리턴합니다. 또한 사용된 문맥에 따라서 숫자 또는 스트링 값을 리턴합니다.

##Oracle 과 MySQL 의 차이

```
##Oracle
SQL> SELECT NVL('', 'A') FROM dual;
N
-
A

##MySQL
mysql> SELECT IFNULL('', 'A');
+----------------+
| IFNULL('', 'A') |
+----------------+
|                |
+----------------+
1 row in set (0.00 sec)

** Oracle 에서는 문자 길이가 0 이면 NULL 로 인식합니다.
```

3.2. SYSDATE / NOW()

Function	SYSDATE (Oracle)	NOW() (MySQL)
Example	SELECT SYSDATE FROM *dual*	SELECT NOW()

MySQL 에서 NOW()는 이 명령문이 실행을 시작하는 시점을 나타내는 시간을 상수 값으로 리턴합니다. (스토어드 루틴 또는 트리거 안에서는, NOW()는 루틴 또는 트리거링 명령문이 실행되는 시점 값을 리턴합니다.) 이 함수는 SYSDATE()와는 차이점을 가지는데, 함수가 실행된 정확한 시간을 리턴합니다.

또한, SET TIMESTAMP 명령문은 NOW()가 리턴하는 값에는 영향을 주지만, SYSDATE()가 리턴하는 값에는 영향을 주지 않습니다. 이것이 의미하는 것은 바이너리 로그에서 설정된 타임 스탬프가 SYSDATE() 함수 호출에는 영향을 주지 않는다는 것을 나타내는 것입니다.

SYSDATE()는 동일한 명령문 안에서도 서로 다른 값을 리턴하고, SET TIMESTAMP 에 의해서도

영향을 받지 않기 때문에, 이 함수는 논-디터미니스틱 (non-deterministic)이며 따라서 리플리케이션에서는 안전하지가 못하게 됩니다. 만일 이것이 문제가 된다면, 서버를 --sysdate-is-now 옵션과 함께 구동 시킴으로서 SYSDATE()가 NOW()의 별칭으로 동작하도록 할 수는 있습니다.

```
## MySQL 에서 NOW()와 SYSDATE() 차이

mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2019-11-18 21:02:11 |        0 | 2019-11-18 21:02:11 |
+---------------------+----------+---------------------+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2019-11-18 21:02:55 |        0 | 2019-11-18 21:02:57 |
+---------------------+----------+---------------------+

mysql> SET timestamp=1673826821;

mysql> SELECT NOW(), SYSDATE();
+---------------------+---------------------+
| NOW()               | SYSDATE()           |
+---------------------+---------------------+
| 2023-01-16 08:53:41 | 2019-11-18 21:09:48 |
+---------------------+---------------------+
```

3.3. || / CONCAT

Function	|| (Oracle)	CONCAT (MySQL)

Example	SELECT col1 || col2 FROM *table*	SELECT CONCAT(col1, col2) FROM *table*

str1 || str2 || ...

CONCAT(str1,str2,...)

인수를 연결 한 결과 문자열을 반환합니다.

Oracle 은 길이가 0 인 문자열을 null 로 취급하지만 길이가 0 인 문자열을 다른 피연산자와 연결하면 항상 다른 피연산자가 발생하므로 null 은 두 개의 null 문자열의 연결에서만 발생할 수 있습니다.

MySQL 은 모든 인수가 이진 문자열이 아닌 경우 결과는 이진이 아닌 문자열입니다. 인수에 이진 문자열이 포함 된 경우 결과는 이진 문자열입니다. 숫자 인수는 동등한 이진이 아닌 문자열 형식으로 변환됩니다.

##Oracle 과 MySQL 의 Null 처리방식 차이

Database	Oracle	MySQL
Value	'' || '' = null	concat('', '') = ''
	'' || null = null	concat('', null) = null
	'a' || null = 'a'	concat('a', null) = null
	'a' || '' = 'a'	concat('a', '') = 'a'

##Oracle

```
SQL> SELECT ('' || '') as T1, ('' || null) as T2, ('a' || null) as T3, ('a' || '') as
T4
  2  FROM dual;

T T T T
- - - -
    a a
```

```
SQL> SELECT empno || comm FROM emp ORDER BY 1;

EMPNO||COMM
--------------------------------------------------------------------------------
7369
7499300
7521500
7566
76541400
7698
7782
7788
7839
78440
7876
7900
7902
7934

14 rows selected.
```

##MySQL

```
mysql> SELECT concat('', ''), concat('', null), concat('a', null), concat('a', '')
    -> FROM dual;
+---------------+----------------+----------------+---------------+
| concat('', '') | concat('', null) | concat('a', null) | concat('a', '') |
+---------------+----------------+----------------+---------------+
|               | NULL           | NULL           | a             |
+---------------+----------------+----------------+---------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT concat(empno,comm) FROM emp ORDER BY 1;
+--------------------+
| concat(empno,comm) |
+--------------------+
| NULL               |
| NULL               |
| NULL               |
| NULL               |
| NULL               |
| NULL               |
| NULL               |
| NULL               |
| NULL               |
| NULL               |
| 7499300            |
| 7521500            |
| 76541400           |
| 78440              |
+--------------------+
14 rows in set (0.00 sec)
```

Oracle 에 사용하는 컬럼에 Null 이 포함되거나 문자 길이가 0 인 ''를 사용하는 경우에 Oracle 과 MySQL 의 결과를 같게 하려면 다음과 같이 변경합니다.

##Oracle

```
SELECT col1 || col2 FROM table
```

##MySQL

```
SELECT IF(concat(ifnull(col1,''),ifnull(col2,''))= '', null,
concat(ifnull(col1,''),ifnull(col2,'')))
```

```
mysql> SELECT concat(empno,ifnull(comm,'')) FROM emp ORDER BY 1;
+------------------------------+
| concat(empno,ifnull(comm,'')) |
+------------------------------+
| 7369                         |
| 7499300                      |
| 7521500                      |
| 7566                         |
| 76541400                     |
| 7698                         |
| 7782                         |
| 7788                         |
| 7839                         |
| 78440                        |
| 7876                         |
| 7900                         |
| 7902                         |
| 7934                         |
+------------------------------+
14 rows in set (0.00 sec)
```

3.4. TO_DATE / STR_TO_DATE

Function	TO_DATE (Oracle)	STR_TO_DATE (MySQL)
Example	SELECT TO_DATE('2019-10-11, 'YYYY-MM-DD') date_col FROM *dual*	SELECT STR_TO_DATE('2019-10-11, '%Y-%m-%d) date_col FROM *dual*

TO_DATE(str,format), STR_TO_DATE(str,format)

문자열 str 을 지정된 fomat 형태의 날짜로 변환합니다.

Date format 은 아래 링크 참조

Oracle : https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/Format-Models.html#GUID-DFB23985-2943-4C6A-96DF-DF0F664CED96

MySQL: https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html

3.5. TO_CHAR / DATE_FORMAT

Function	TO_CHAR (Oracle)	DATE_FORMAT (MySQL)
Example	SELECT TO_CAHR(SYSDATE, 'YYYY-MM-DD') date_col FROM *dual*	SELECT DATE_FORMAT(NOW(), '%Y-%m-%d) date_col FROM *dual*

TO_CHAR(date,format), DATE_FORMAT(date,format)

date 값에 따라 format 값을 형식화 합니다.

date format 은 2.4 TO_DATE / STR_TO_DATE 의 링크 이용

3.6. TO_CHAR / CAST

Function	TO_CHAR (Oracle)	CAST (MySQL)
Example	SELECT TO_CAHR(1011) FROM *dual*	SELECT CAST(1011 AS CHAR) FROM *dual*

TO_CHAR(character)

지정된 character 값을 문자형으로 변환합니다. NCHAR, NVARCHAR2, CLOB, 또는 NCLOB 문자 집합을 항상 VARCHAR2 로 반환합니다..

CAST(expr AS char)

지정된 expr 값을 문자형으로 변환합니다. CHAR(N)는 인수의 N 문자보다 작은 것을 사용해서 캐스트 되도록 만듭니다.

3.7. TO_NUMBER / CAST

Function	TO_NUMBER (Oracle)	CAST (MySQL)
Example	SELECT TO_NUMBER(1011) FROM *dual*	SELECT CAST(1011 AS UNSIGEND) FROM *dual*

TO_NUMBER(expr)

지정된 expr 값을 숫자로 변환합니다. expr 에는

CHAR, VARCHAR2, NCHAR, NVARCHAR2, BINARY_FLOAT, BINARY_DOUBLE 값들이 올 수 있습니다

CAST(expr AS unsigned)

지정된 expr 값을 숫자로 변환합니다.

3.8. DECODE / CASE

Function	DECODE (Oracle)	CASE (MySQL)
Example	SELECT DECODE(address, 'seoul', 1, 'busan', 2, 'daegu', 3, 4) FROM *table*	SELECT CASE address WHEN 'seoul' THEN 1 WHEN 'busan' THEN 2 WHEN 'deagu' THEN 3 ELSE 4 END FROM *table*

DECODE(expr, search, result
[, search, result]...
[, default]
)

Expr 이 각 search 값을 하나씩 비교합니다. 동일한 search 값이 있을 경우 해당하는 result 값을 반환합니다. 일치하는 값이 없을 경우 default 값을 반환 하고 default 를 생략할 경우 null 값을 반환합니다.

CASE case_value
WHEN when_value THEN statement_list
[WHEN when_value THEN statement_list] ...
[ELSE statement_list]
END CASE

case_value 이 WHEN 절 중 하나가 같을 때까지 각 절의 표현식과 비교합니다. 일치하는 값이 나오면 statement_list 를 반환합니다. 일치 값이 없을 경우 ELSE 절의 statement_list 를 반환합니다. ELSE 절을 생략할 경우 null 값을 반환합니다.

3.9. ROWNUM / LIMIT

Function	ROWNUM (Oracle)	LIMIT (MySQL)
Example	SELECT * FROM *table* WHERE ROWNUM <= 10 SELECT * FROM *table* WHERE ROWNUM < 10	SELECT * FROM *table* LIMIT 10 SELECT * FROM *table* LIMIT 9 (부등호에 따라 상이)

3.10. TRUNC / TRUNCATE

Function	TRUNC (Oracle)	TRUNCATE (MySQL)
Example	SELECT TRUNC(10/3) FROM *dual*	SELECT TRUNCATE(10/3, 0) FROM *dual*

TRUNC (n1 [, n2]) <-> TRUNCATE(n1, n2)

숫자 n1 을 n2 자릿수 뒤의 숫자를 없앤 후 리턴합니다. 만일 n2 가 0 이면, 리턴되는 숫자는 소수점 또는 소수 부분이 없게 됩니다. n2 는 음수로 표시할 수도 있는데, 이렇게 표시하면 숫자 n1 의 n2 자리로부터 왼쪽 부분이 0 으로 표시가 됩니다. MySQL 과는 다르게 Oralce 에서는 n2 를 생략할수 있고 생략시 n2 는 0 의 값을 가집니다.

3.11. CHR / CHAR

Function	CHR (Oracle)	CHAR (MySQL)
Example	SELECT CHR(100) FROM *dual*	SELECT CHAR(100 USING ASCII) FROM *dual*

CHR(N [USING NCHAR_CS]) <-> CHAR(N,... [USING charset_name])

각각의 인수 N 을 정수 형태로 해석을 하고 이러한 정수의 코드 값에 의해 주어지는 문자로 구성된 스트링을 리턴합니다. NULL 값은 무시(skipped)됩니다.

4. Sentence Structure

4.1. JOIN(+) / OUTER JOIN

Sentence	JOIN(+) (Oracle)	OUTER JOIN (MySQL)
Example	SELECT t1.*, t2.* FROM *t1, t2* WHERE t1.col1 = t2.col1(+) SELECT t1.*, t2.* FROM *t1, t2* WHERE t1.col1(+) = t2.col1	SELECT t1.*, t2.* FROM *t1* LEFT OUTER JOIN *t2* ON t1.col1 = t2.col1 SELECT t1.*, t2.* FROM *t1* RIGHT OUTER JOIN *t2* ON t1.col1 = t2.col1

Oracle의 outer join을 MySQL에서는 ANSI 방식의 join으로 변경 해야합니다.
Oracle에서 full outer join을 사용할 경우 MySQL에서는 left join 과 right join을 union 해야 합니다.

##Oracle

```
SQL> SELECT a.empno, a.deptno, a.ename, b.dname
  2  FROM emp a, dept b
  3  WHERE a.deptno(+) = b.deptno
  4  AND b.deptno > 20 ;
     EMPNO     DEPTNO ENAME      DNAME
---------- ---------- ---------- --------------
      7499         30 ALLEN      SALES
      7521         30 WARD       SALES
      7654         30 MARTIN     SALES
      7698         30 BLAKE      SALES
      7844         30 TURNER     SALES
      7900         30 JAMES      SALES
                                 OPERATIONS
7 rows selected.
```

##MySQL

```
mysql> SELECT a.empno, a.deptno, a.ename, b.dname
    -> FROM emp a RIGHT JOIN dept b
    -> ON a.deptno = b.deptno
    -> WHERE b.deptno > 20 ;
+-------+--------+--------+------------+
| empno | deptno | ename  | dname      |
+-------+--------+--------+------------+
|  7499 |     30 | ALLEN  | SALES      |
|  7521 |     30 | WARD   | SALES      |
|  7654 |     30 | MARTIN | SALES      |
|  7698 |     30 | BLAKE  | SALES      |
|  7844 |     30 | TURNER | SALES      |
|  7900 |     30 | JAMES  | SALES      |
|  NULL |   NULL | NULL   | OPERATIONS |
+-------+--------+--------+------------+
7 rows in set (0.00 sec)
```

4.2. OVER

Sentence	OVER (Oracle)	N/A (MySQL)
Example	SELECT col1, col2, col3, sum(col3) OVER (PARTITION BY col1) FROM *table* ORDER BY 1, 3	SELECT a.co1, a.col2, a.col3, b.col3 FROM *table* as a INNER JOIN (SELECT col1, sum(col3) as col3 FROM *table* GROUP BY col1) as b ON a.col1 = b.col1 ORDER BY 1, 3

##Oracle

```
SQL> SELECT deptno, ename, sal, sum(sal)
  2  OVER (PARTITION BY deptno) as sal
  3  FROM emp
  4  ORDER BY 1, 3 ;

    DEPTNO ENAME             SAL        SAL
---------- ---------- ---------- ----------
        10 MILLER           1300       8750
        10 CLARK            2450       8750
        10 KING             5000       8750
        20 SMITH             800      10875
        20 ADAMS            1100      10875
        20 JONES            2975      10875
        20 SCOTT            3000      10875
        20 FORD             3000      10875
        30 JAMES             950       9400
        30 MARTIN           1250       9400
        30 WARD             1250       9400
        30 TURNER           1500       9400
        30 ALLEN            1600       9400
        30 BLAKE            2850       9400
14 rows selected.
```

##MySQL

```
mysql> SELECT a.deptno, a.ename, a.sal, b.sal
    -> FROM emp as a
    -> INNER JOIN (SELECT deptno, sum(sal) as sal
    ->              FROM emp
    ->              GROUP BY deptno) as b
    -> ON a.deptno = b.deptno
    -> ORDER BY 1, 3;
+--------+--------+------+-------+
| deptno | ename  | sal  | sal   |
+--------+--------+------+-------+
|     10 | MILLER | 1300 |  8750 |
|     10 | CLARK  | 2450 |  8750 |
|     10 | KING   | 5000 |  8750 |
|     20 | SMITH  |  800 | 10875 |
|     20 | ADAMS  | 1100 | 10875 |
|     20 | JONES  | 2975 | 10875 |
|     20 | SCOTT  | 3000 | 10875 |
|     20 | FORD   | 3000 | 10875 |
|     30 | JAMES  |  950 |  9400 |
|     30 | WARD   | 1250 |  9400 |
|     30 | MARTIN | 1250 |  9400 |
|     30 | TURNER | 1500 |  9400 |
|     30 | ALLEN  | 1600 |  9400 |
|     30 | BLAKE  | 2850 |  9400 |
+--------+--------+------+-------+
14 rows in set (0.00 sec)
```

4.3. RANK() OVER

Sentence	RANK() OVER (Oracle)	N/A (MySQL)
Example	SELECT * FROM (SELECT col1, col2, col3, RANK() OVER (PARTITION BY col1 ORDER BY col3 desc) RANK FROM *table*) WHERE RANK <= 3 ORDER BY col1, RANK	SELECT * FROM (SELECT a.col1, a.col2, a.col3, (SELECT 1 + count(*) FROM table b WHERE b.col1 = a.col1 AND b.col3 > a.col3) RNK FROM table a) c WHERE c.RNK <= 3 ORDER BY c.col1, c.RNK

#Oracle

```
SQL> SELECT *
  2  FROM (SELECT deptno, ename, sal,
  3        RANK() OVER (PARTITION BY deptno ORDER BY sal desc) RANK
  4        FROM emp)
  5  WHERE RANK <= 3
  6  ORDER BY deptno, RANK;

    DEPTNO ENAME             SAL       RANK
---------- ---------- ---------- ----------
        10 KING             5000          1
        10 CLARK            2450          2
        10 MILLER           1300          3
        20 SCOTT            3000          1
        20 FORD             3000          1
        20 JONES            2975          3
        30 BLAKE            2850          1
        30 ALLEN            1600          2
        30 TURNER           1500          3
9 rows selected.
```

**오름차순으로 순위를 매기려면 line3 에서 asc 으로 sort 하면 됩니다.

```
##MySQL
mysql> SELECT *
    -> FROM (SELECT a.deptno, a.ename, a.sal,
    ->       (SELECT 1 + count(*) FROM emp b
    ->       WHERE b.deptno = a.deptno
    ->       AND b.sal > a.sal) RNK
    ->       FROM emp a) c
    -> WHERE c.RNK <= 3
    -> ORDER BY c.deptno, c.RNK;
+--------+--------+------+------+
| deptno | ename  | sal  | RNK  |
+--------+--------+------+------+
|     10 | KING   | 5000 |    1 |
|     10 | CLARK  | 2450 |    2 |
|     10 | MILLER | 1300 |    3 |
|     20 | SCOTT  | 3000 |    1 |
|     20 | FORD   | 3000 |    1 |
|     20 | JONES  | 2975 |    3 |
|     30 | BLAKE  | 2850 |    1 |
|     30 | ALLEN  | 1600 |    2 |
|     30 | TURNER | 1500 |    3 |
+--------+--------+------+------+
9 rows in set (0.00 sec)

**오름차순으로 순위를 매기려면 line5 에서 b.sal < a.sal 로 조건을 주면 됩니다.
```

4.4. COUNT() OVER

Sentence	COUNT() OVER (Oracle)	N/A (MySQL)
Example	SELECT col1, col2, COUNT(*) OVER() FROM *table*	SELECT col1, col2, (SELECT count(*) FROM *table*) FROM *table*

4.5. ROW_NUMBER() OVER

Sentence	ROW_NUMBER() OVER (Oracle)	N/A (MySQL)
Example	SELECT col1, col2, col3, col4, ROW_NUMBER() OVER(PARTITION BY col3 ORDER BY col4) as rnum FROM *table*	SELECT col1, col2, col3, col4, rnum FROM (SELECT a.*, (CASE @vcol3 WHEN a.col3 THEN @rnum := @rnum + 1 ELSE @rnum := 1 END) as rnum, (@vcol3 := a.col3) as vcol3 FROM *table* a JOIN (SELECT @vcol3 := '', @rnum := 0) b ORDER BY a.col3, a.col4) c

##Oracle

```
SQL> SELECT empno, ename, job, sal,
  2        ROW_NUMBER()
  3  OVER(PARTITION BY job
  4  ORDER BY sal, empno) as rnum
  5  FROM emp;
```

EMPNO	ENAME	JOB	SAL	RNUM
7788	SCOTT	ANALYST	3000	1
7902	FORD	ANALYST	3000	2
7369	SMITH	CLERK	800	1
7900	JAMES	CLERK	950	2
7876	ADAMS	CLERK	1100	3
7934	MILLER	CLERK	1300	4
7782	CLARK	MANAGER	2450	1
7698	BLAKE	MANAGER	2850	2
7566	JONES	MANAGER	2975	3
7839	KING	PRESIDENT	5000	1
7521	WARD	SALESMAN	1250	1
7654	MARTIN	SALESMAN	1250	2
7844	TURNER	SALESMAN	1500	3

```
      7499 ALLEN      SALESMAN        1600          4
14 rows
```

##MySQL

```
mysql> SELECT empno, ename, job, sal, rnum
    -> FROM (SELECT a.*,
    ->              (CASE @vjob
    ->                 WHEN a.job
    ->                 THEN @rnum := @rnum + 1
    ->                 ELSE @rnum := 1
    ->               END) as rnum,
    ->              (@vjob := a.job) as vjob
    ->         FROM emp a JOIN
    -> (SELECT @vjob := '', @rnum := 0) b
    -> ORDER BY a.job, a.sal, a.empno) c;
+-------+--------+-----------+------+------+
| empno | ename  | job       | sal  | rnum |
+-------+--------+-----------+------+------+
|  7788 | SCOTT  | ANALYST   | 3000 |    1 |
|  7902 | FORD   | ANALYST   | 3000 |    2 |
|  7369 | SMITH  | CLERK     |  800 |    1 |
|  7900 | JAMES  | CLERK     |  950 |    2 |
|  7876 | ADAMS  | CLERK     | 1100 |    3 |
|  7934 | MILLER | CLERK     | 1300 |    4 |
|  7782 | CLARK  | MANAGER   | 2450 |    1 |
|  7698 | BLAKE  | MANAGER   | 2850 |    2 |
|  7566 | JONES  | MANAGER   | 2975 |    3 |
|  7839 | KING   | PRESIDENT | 5000 |    1 |
|  7521 | WARD   | SALESMAN  | 1250 |    1 |
|  7654 | MARTIN | SALESMAN  | 1250 |    2 |
|  7844 | TURNER | SALESMAN  | 1500 |    3 |
|  7499 | ALLEN  | SALESMAN  | 1600 |    4 |
+-------+--------+-----------+------+------+
14 rows in set, 5 warnings (0.00 sec)
```

4.6. ROWNUM

Sentence	ROWNUM (Oracle)	N/A (MySQL)
Example	SELECT b.* FROM (SELECT ROWNUM as rnum, a.* FROM *table* a) b	SELECT b.* FROM (SELECT @ROWNUM := @ROWNUM + 1 as rnum, a.* FROM *table* a JOIN (SELECT @ROWNUM := 0) TMP) b

MySQL 에서는 결과 값이 많아질수록 모든 행에 rownum 을 매겨야 하므로 성능이 나빠집니다. 따라서 LIMIT OFFSET 구문으로 변경해주는 것이 좋습니다.

(구문 : LIMIT 가져올 행 수 OFFSET 시작 행 번호 - 1)

그러나 조건의 범위는 부등호 Equal 의 유무에 따라 값이 달라집니다. ROWNUM 도 마찬가지입니다. 해당 숫자를 포함하느냐 안하느냐에 따라서 같은 변수에도 결과값이 달라집니다.

A=10, B=20 일 때

rnum BETWEEN A AND B 는 10 과 20 을 포함합니다.

LIMIT OFFSET 구문으로 같은 값을 얻고 싶을 경우

LIMIT 11 OFFSET 9 를 입력하면 됩니다. 더 간단한 문장으로 OFFSET 을 ,로 바꿔주고 변수 자리를 바꿔주면 됩니다. Ex) LIMIT 9, 11

##Oracle

```
SQL> SELECT b.rnum, b.empno, b.ename, b.hiredate, b.sal, b.comm, b.deptno
  2  FROM (SELECT ROWNUM as rnum, a.*
  3          FROM emp a
  4          WHERE rownum <=10) b
  5  WHERE b.rnum >4;

      RNUM      EMPNO ENAME      HIREDATE          SAL       COMM     DEPTNO
---------- ---------- ---------- --------- ---------- ---------- ----------
         5       7654 MARTIN     28-SEP-81       1250       1400         30
         6       7698 BLAKE      01-MAY-81       2850                    30
```

```
         7       7782 CLARK      09-JUN-81       2450                    10
         8       7788 SCOTT      19-APR-87       3000                    20
         9       7839 KING       17-NOV-81       5000                    10
        10       7844 TURNER     08-SEP-81       1500          0         30

6 rows selected.
```

##MySQL

```
mysql> SELECT b.rnum, b.empno, b.ename, b.hiredate, b.sal, b.comm, b.deptno
    -> FROM (SELECT @ROWNUM := @ROWNUM + 1 as rnum, a.*
    ->         FROM emp a JOIN
    -> (SELECT @ROWNUM := 4) TMP
    -> LIMIT 4, 6 ) b;
+------+-------+--------+-----------+------+------+--------+
| rnum | empno | ename  | hiredate  | sal  | comm | deptno |
+------+-------+--------+-----------+------+------+--------+
|    5 |  7654 | MARTIN | 28-SEP-81 | 1250 | 1400 |     30 |
|    6 |  7698 | BLAKE  | 01-MAY-81 | 2850 | NULL |     30 |
|    7 |  7782 | CLARK  | 09-JUN-81 | 2450 | NULL |     10 |
|    8 |  7788 | SCOTT  | 19-APR-87 | 3000 | NULL |     20 |
|    9 |  7839 | KING   | 17-NOV-81 | 5000 | NULL |     10 |
|   10 |  7844 | TURNER | 08-SEP-81 | 1500 |    0 |     30 |
+------+-------+--------+-----------+------+------+--------+
6 rows in set, 2 warnings (0.00 sec)
```

구문	Oracle	MySQL
	rnum <= A AND rnum >= B	LIMIT B - 1, A - B + 1
	rnum <= A AND rnum > B	LIMIT B, A - B
	rnum < A AND rnum >= B	LIMIT B - 1, A - B
	rnum < A AND rnum > B	LIMIT B, A + 1

4.7. ALIAS

Sentence	ALIAS (Oracle)	N/A (MySQL)

Example	SELECT col1, col2 FROM (SELECT * FROM table WHERE col3 = 1) GROUP BY col1, col2	SELECT col1, col2 FROM (SELECT * FROM table WHERE col3 = 1) a GROUP BY col1, col2 모든 Query Block 에는 반드시 Alias 가 필요
Example	INSERT INTO *table1* a (col1, col2, col3) SELECT col4, col5, col6 FROM *table2*	INSERT INTO *table1* (col1, col2, col3) SELECT col4, col5, col6 FROM *table2* INSERT 절에는 Alias 사용 불가

4.8. HINT(INDEX)

Sentence	HINT(INDEX) (Oracle)	USE INDEX (MySQL)
Example	SELECT /*+ INDEX(a idx_01) */ * FROM table a WHERE col1 = 1 AND col2 = 2 AND col3 = 3	SELECT * FROM table USE INDEX(idx01) WHERE col1 = 1 AND col2 = 2 AND col3 = 3

Oracle 은 hint 가 잘못 적혀도 파싱이 되지만, MySQL 은 hint 가 잘못 적힐 경우 파싱이 되지 않으니 주의해야 합니다.

4.9. MERGE

Sentence	MERGE (Oracle)	ON DUPLICATE KEY (MySQL)
Example	MERGE INTO table a USING DUAL ON (col1 = val1 AND col2 = val2) WHEN MATCHED THEN UPDATE SET col3 = val3, col4 = val4 WHEN NOT MATCHED THEN INSERT (col1, col2, col3, col4) VALUES (val1, val2, val3, val4)	INSERT INTO table (col1, col2, col3, col4) VALUES (val1, val2, val3, val4) ON DUPLICATE KEY UPDATE col3 = val3, col4 = val4
Table 사용	MERGE INTO *table1* a USING *table2* b ON(a.col1 = b.col1 AND a.col2 = b.col2) WHEN MATCHED THEN UPDATE SET a.col3 = b.col3, a.col4 =	INSERT INTO *table1* (col1, col2, col3, col4) SELECT b.col1, b.col2, b.col3, b.col4 FROM *table1* a JOIN *table2* b ON a.col1 = b.col1 AND a.col2 = b.col2

	b.col4 WHEN NOT MATCHED THEN INSER (col1, col2, col3, col4) VALUES (b.col1, b.col2, b.col3, b.col4)	ON DUPLICATE KEY UPDATE col3 = b.col3, col4 = b.col4 마지막 LINE col3, col4 에 alias 넣으면 안됨

5. ETC

5.1. Order by

Oracle 과 MySQL 의 Null 에 대한 ORDER BY 가 서로 다릅니다. Ascending 으로 정렬할때 Oralce 은 가장 큰 값이 Null 이지만 MySQL 에서는 가장 작은 값이 Null 입니다. Descending 으로 정렬할 경우 그 반대입니다. MySQL 에서 Oracle 과 같이 정렬하려는 경우에는 asc, desc 에 따라 is null, is not null 을 선행으로 정렬해야 합니다.

```
##ASC
##Oracle

SQL> SELECT * FROM emp ORDER BY comm ASC;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7844 TURNER     SALESMAN         7698 08-SEP-81       1500          0         30
      7499 ALLEN      SALESMAN         7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN         7698 22-FEB-81       1250        500         30
      7654 MARTIN     SALESMAN         7698 28-SEP-81       1250       1400         30
      7788 SCOTT      ANALYST          7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT             17-NOV-81       5000                    10
      7876 ADAMS      CLERK            7788 23-MAY-87       1100                    20
      7900 JAMES      CLERK            7698 03-DEC-81        950                    30
      7902 FORD       ANALYST          7566 03-DEC-81       3000                    20
      7934 MILLER     CLERK            7782 23-JAN-82       1300                    10
      7698 BLAKE      MANAGER          7839 01-MAY-81       2850                    30
      7566 JONES      MANAGER          7839 02-APR-81       2975                    20
      7369 SMITH      CLERK            7902 17-DEC-80        800                    20
      7782 CLARK      MANAGER          7839 09-JUN-81       2450                    10

14 rows selected.

##MySQL
```

```
mysql> SELECT * FROM emp ORDER BY comm ASC;
+-------+--------+-----------+------+-----------+------+------+--------+
| empno | ename  | job       | mgr  | hiredate  | sal  | comm | deptno |
+-------+--------+-----------+------+-----------+------+------+--------+
|  7566 | JONES  | MANAGER   | 7839 | 02-APR-81 | 2975 | NULL |     20 |
|  7698 | BLAKE  | MANAGER   | 7839 | 01-MAY-81 | 2850 | NULL |     30 |
|  7782 | CLARK  | MANAGER   | 7839 | 09-JUN-81 | 2450 | NULL |     10 |
|  7788 | SCOTT  | ANALYST   | 7566 | 19-APR-87 | 3000 | NULL |     20 |
|  7839 | KING   | PRESIDENT | NULL | 17-NOV-81 | 5000 | NULL |     10 |
|  7876 | ADAMS  | CLERK     | 7788 | 23-MAY-87 | 1100 | NULL |     20 |
|  7900 | JAMES  | CLERK     | 7698 | 03-DEC-81 |  950 | NULL |     30 |
|  7902 | FORD   | ANALYST   | 7566 | 03-DEC-81 | 3000 | NULL |     20 |
|  7934 | MILLER | CLERK     | 7782 | 23-JAN-82 | 1300 | NULL |     10 |
|  7369 | SMITH  | CLERK     | 7902 | 17-DEC-80 |  800 | NULL |     20 |
|  7844 | TURNER | SALESMAN  | 7698 | 08-SEP-81 | 1500 |    0 |     30 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 20-FEB-81 | 1600 |  300 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 22-FEB-81 | 1250 |  500 |     30 |
|  7654 | MARTIN | SALESMAN  | 7698 | 28-SEP-81 | 1250 | 1400 |     30 |
+-------+--------+-----------+------+-----------+------+------+--------+
14 rows in set (0.00 sec)


mysql> SELECT * FROM emp ORDER BY comm IS NULL, comm ASC;
+-------+--------+-----------+------+-----------+------+------+--------+
| empno | ename  | job       | mgr  | hiredate  | sal  | comm | deptno |
+-------+--------+-----------+------+-----------+------+------+--------+
|  7844 | TURNER | SALESMAN  | 7698 | 08-SEP-81 | 1500 |    0 |     30 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 20-FEB-81 | 1600 |  300 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 22-FEB-81 | 1250 |  500 |     30 |
|  7654 | MARTIN | SALESMAN  | 7698 | 28-SEP-81 | 1250 | 1400 |     30 |
|  7566 | JONES  | MANAGER   | 7839 | 02-APR-81 | 2975 | NULL |     20 |
|  7698 | BLAKE  | MANAGER   | 7839 | 01-MAY-81 | 2850 | NULL |     30 |
|  7782 | CLARK  | MANAGER   | 7839 | 09-JUN-81 | 2450 | NULL |     10 |
|  7788 | SCOTT  | ANALYST   | 7566 | 19-APR-87 | 3000 | NULL |     20 |
|  7839 | KING   | PRESIDENT | NULL | 17-NOV-81 | 5000 | NULL |     10 |
|  7876 | ADAMS  | CLERK     | 7788 | 23-MAY-87 | 1100 | NULL |     20 |
|  7900 | JAMES  | CLERK     | 7698 | 03-DEC-81 |  950 | NULL |     30 |
```

```
|  7902 | FORD   | ANALYST   | 7566 | 03-DEC-81 | 3000 | NULL |     20 |
|  7934 | MILLER | CLERK     | 7782 | 23-JAN-82 | 1300 | NULL |     10 |
|  7369 | SMITH  | CLERK     | 7902 | 17-DEC-80 |  800 | NULL |     20 |
+-------+--------+-----------+------+-----------+------+------+--------+
14 rows in set (0.01 sec)
```

##Desc

##Oracle

```
SQL> SELECT * FROM emp ORDER BY comm DESC;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30

14 rows selected.
```

##MySQL

```
mysql> SELECT * FROM emp ORDER BY comm IS NOT NULL, comm DESC;
+-------+--------+-----------+------+-----------+------+------+--------+
| empno | ename  | job       | mgr  | hiredate  | sal  | comm | deptno |
+-------+--------+-----------+------+-----------+------+------+--------+
|  7566 | JONES  | MANAGER   | 7839 | 02-APR-81 | 2975 | NULL |     20 |
|  7698 | BLAKE  | MANAGER   | 7839 | 01-MAY-81 | 2850 | NULL |     30 |
|  7782 | CLARK  | MANAGER   | 7839 | 09-JUN-81 | 2450 | NULL |     10 |
```

```
|  7788 | SCOTT  | ANALYST   | 7566 | 19-APR-87 | 3000 | NULL |     20 |
|  7839 | KING   | PRESIDENT | NULL | 17-NOV-81 | 5000 | NULL |     10 |
|  7876 | ADAMS  | CLERK     | 7788 | 23-MAY-87 | 1100 | NULL |     20 |
|  7900 | JAMES  | CLERK     | 7698 | 03-DEC-81 |  950 | NULL |     30 |
|  7902 | FORD   | ANALYST   | 7566 | 03-DEC-81 | 3000 | NULL |     20 |
|  7934 | MILLER | CLERK     | 7782 | 23-JAN-82 | 1300 | NULL |     10 |
|  7369 | SMITH  | CLERK     | 7902 | 17-DEC-80 |  800 | NULL |     20 |
|  7654 | MARTIN | SALESMAN  | 7698 | 28-SEP-81 | 1250 | 1400 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 22-FEB-81 | 1250 |  500 |     30 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 20-FEB-81 | 1600 |  300 |     30 |
|  7844 | TURNER | SALESMAN  | 7698 | 08-SEP-81 | 1500 |    0 |     30 |
+-------+--------+-----------+------+-----------+------+------+--------+
14 rows in set (0.01 sec)
```

5.2. SEQUENCE

MySQL 은 Oracle 과는 다르게 sequence object 가 존재하지 않습니다. 유사한 형태의 auto_increment 옵션을 제공하지만 이는 해당 테이블에 대한 순차 번호만을 제공합니다. Oracle 과 유사한 기능을 제공하기 위해서 MySQL 은 시퀀스 테이블 및 함수를 생성하여 사용을 해야합니다.

```
##sequence DB 생성
mysql> CREATE DATABASE `sequence`;
Query OK, 1 row affected (0.00 sec)

##sequence table 생성
mysql> CREATE TABLE `sequence`.`sequence_data` (
    ->     `sequence_name` varchar(100) NOT NULL,
    ->     `sequence_increment` int(11) unsigned NOT NULL DEFAULT 1,
    ->     `sequence_min_value` int(11) unsigned NOT NULL DEFAULT 1,
    ->     `sequence_max_value` bigint(20) unsigned NOT NULL DEFAULT
18446744073709551615,
    ->     `sequence_cur_value` bigint(20) unsigned DEFAULT 1,
    ->     `sequence_cycle` boolean NOT NULL DEFAULT FALSE,
    ->     PRIMARY KEY (`sequence_name`)
    -> ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.01 sec)
```

```
##nextval function 생성
mysql> DELIMITER $$
mysql> CREATE FUNCTION `nextval` (`seq_name` varchar(100))
    -> RETURNS bigint(20) unsigned
    -> BEGIN
    ->     DECLARE cur_val bigint(20) unsigned;
    ->
    ->     SELECT
    ->         sequence_cur_value + sequence_increment  INTO cur_val
    ->     FROM
    ->         sequence.sequence_data
    ->     WHERE
    ->         sequence_name = seq_name
    ->     ;
    ->
    ->     IF cur_val IS NOT NULL THEN
    ->         UPDATE
    ->             sequence.sequence_data
    ->         SET
    ->             sequence_cur_value = IF (
    ->                 (sequence_cur_value + sequence_increment) > sequence_max_value,
    ->                 IF (
    ->                     sequence_cycle = TRUE,
    ->                     sequence_min_value,
    ->                     NULL
    ->                 ),
    ->                 sequence_cur_value + sequence_increment
    ->             )
    ->         WHERE
    ->             sequence_name = seq_name
    ->         ;
    ->     END IF;
    ->
    ->     RETURN cur_val;
    ->
    -> END $$
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;



##currval function 생성
mysql> DELIMITER $$
mysql> CREATE FUNCTION `currval` (`seq_name` varchar(100))
    -> RETURNS bigint(20) unsigned
    -> BEGIN
    ->     DECLARE cur_val bigint(20) unsigned;
    ->
    ->     SELECT
    ->         sequence_cur_value INTO cur_val
    ->     FROM
    ->         sequence.sequence_data
    ->     WHERE
    ->         sequence_name = seq_name
    ->     ;
    ->
    ->     RETURN cur_val;
    ->
    -> END $$
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql>



##sequence table 확인
mysql> desc sequence_data;
+--------------------+--------------------+------+-----+---------------------+-------+
| Field              | Type               | Null | Key | Default             | Extra |
+--------------------+--------------------+------+-----+---------------------+-------+
| sequence_name      | varchar(100)       | NO   | PRI | NULL                |       |
| sequence_increment | int(11) unsigned   | NO   |     | 1                   |       |
| sequence_min_value | int(11) unsigned   | NO   |     | 1                   |       |
```

```
| sequence_max_value | bigint(20) unsigned | NO   |     | 18446744073709551615 |       |
| sequence_cur_value | bigint(20) unsigned | YES  |     | 1                    |       |
| sequence_cycle     | tinyint(1)          | NO   |     | 0                    |       |
+--------------------+---------------------+------+-----+----------------------+-------+
6 rows in set (0.00 sec)
```

##sequence 생성

```
mysql> insert into sequence_data (sequence_name) values ('test');
Query OK, 1 row affected (0.01 sec)
```

currval 확인

```
mysql> SELECT currval('test');
+-----------------+
| currval('test') |
+-----------------+
|               1 |
+-----------------+
1 row in set (0.00 sec)
```

nextval 확인

```
mysql> SELECT nextval('test');
+-----------------+
| nextval('test') |
+-----------------+
|               2 |
+-----------------+
1 row in set (0.01 sec)
```

##auto_increment table 생성

```
mysql> CREATE TABLE `t1` (
    ->   `ID` bigint unsigned NOT NULL AUTO_INCREMENT,
    ->   `SEQ` bigint unsigned,
    -> PRIMARY KEY (`ID`)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

```
auto_increment table에 insert
mysql> INSERT INTO t1 (SEQ) VALUES (currval('test'));
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t1 (SEQ) VALUES (nextval('test'));
Query OK, 1 row affected (0.01 sec)


##auto_increment, sequence 값 확인
mysql> SELECT * FROM t1;
+----+------+
| ID | SEQ  |
+----+------+
|  1 |    2 |
|  2 |    3 |
+----+------+
2 rows in set (0.01 sec)
```

6. Reference

https://dev.mysql.com/doc/refman/5.7/en/identifier-case-sensitivity.html(Identifier Case Sensitivity)

https://dev.mysql.com/doc/refman/5.7/en/string-comparison-functions.html(String Comparison Functions and Operators)

https://dev.mysql.com/doc/refman/5.7/en/char.html(The CHAR and VARCHAR Types)

https://dev.mysql.com/doc/refman/5.7/en/functions.html(Functions and Operators)

< 끝 >