MySQL Day 2

Index

1. DATATYPE

reference : https://dev.mysql.com/doc/refman/5.7/en/data-types.html

데이터 타입은 컴퓨터의 자원을 효율적으로 사용하기 위해 사용되는 방법입니다. 많이 사용되는 데이터 타입의 종류는 숫자형, 문자형, 날짜형 등이 있습니다. 저장할 데이터의 타입을 지정하면 저장공간의 할당을 효율적으로 할수 있어 DBMS의 성능을 증가 시킬수 있는 장점이 있습니다.

(1) Numberic : 숫자형 데이터 타입

reference : https://dev.mysql.com/doc/refman/5.7/en/numeric-types.html

 # 정수 타입 (integer types)

reference : https://dev.mysql.com/doc/refman/5.7/en/integer-types.html

Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2^63	0	2^63-1	2^64-1

TINYINT 테이블을 생성해서 해당 범위의 값이 들어가는지 확인합니다.

```
CREATE TABLE number1(
        data TINYINT
);
```

지정된 데이터 타입이 표현할수 있는 숫자의 범위를 벗어났기 때문에 데이터가 입력되지 않습니다.

```
INSERT INTI number1
VALUE (128);
```

Unsigned 조건을 추가하면 0 ~ 255까지의 숫자를 입력할수 있습니다.

```
CREATE TABLE number2(
        data TINYINT UNSIGNED
);

INSERT INTI number2
VALUE (128);
```

* 테이블의 컬럼에 어느정도 범위로 데이터가 들어가는지 확인을 해서 컬럼의 데이터 타입을 결정해야 합니다.

 # 실수 타입 (float types)

reference : https://dev.mysql.com/doc/refman/5.7/en/floating-point-types.html

소수점을 나타내기 위한 데이터 타입으로 아래의 두가지 데이터 타입이 있습니다. 두가지의 데이터 타입은 데이터 저장공간의 차이가 있습니다.

FLOAT (4byte) : 정수부와 실수부를 합쳐 6자리의 숫자를 입력할수 있습니다.

DOUBLE (8byte) : 정수부와 실수부를 합쳐 17자리의 숫자를 입력할수 있습니다.

FLOAT 데이터 타입의 테이블 생성

```
CREATE TABLE number3(
        data FLOAT
);
```

FLOAT 데이터 입력

```
INSERT INTI number3
VALUE (12.3456789);
```

```
# DOUBLE 테이블 생성
CREATE TABLE number4(
        data DOUBLE
);

# DOUBLE 데이터 입력
INSERT INTI number4
VALUE (1234567890.1234567890);
```

 # DECIMAL, NUMERIC

reference : https://dev.mysql.com/doc/refman/5.7/en/fixed-point-types.html

실수에서 전체 자리수와 소수점 자리수를 설정할수 있습니다.
DECIMAL(5, 2) 로 설정하면 전체 자리수를 5자리 소수점을 2자리로 설정한다는 의미 입니다.
DECIMAL 예약어 대신에 NUMERIC 예약어를 사용해도 됩니다.

```
# DECIMAL 데이터 타입을 사용하는 테이블 생성
CREATE TABLE number5(
        data DECIMAL(5, 2)
);

# 데이터 입력 : 소수점 3번째 자리에서 반올림 되어 123.46 이 입력됩니다.
insert into number5
value (123.456);

# 데이터 입력 : 소수점 3번째 자리에서 반올림 되어 12.35 가 입력됩니다.
insert into number5
value (12.345);
```

(2) String : 문자열 데이터 타입

reference : https://dev.mysql.com/doc/refman/5.7/en/string-types.html

 # CHAR & VARCHAR

CHAR : 고정길이 문자열 데이터 타입으로 255(2^8)자 까지 입력이 가능

VARCHAR : 가변길이 문자열 데이터 타입으로 65535(2^{16})자 까지 입력이 가능

CHAR & VARCHAR 의 차이

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

CHAR 데이터 타입이 들어간 테이블 생성

```
CREATE TABLE str1(
        data CHAR(255) # 256 이상으로 설정하면 테이블이 생성되지 않습니다.
);
```

데이터 입력

```
INSERT INTO str1
VALUE ("문자열 입력");
```

 # TEXT

CHAR와 VARCHAR는 대체로 크기가 작은 문자열을 저장할때 사용되며 크기가 큰 문자열을 저장할 때는 TEXT 데이터 타입을 사용합니다. TEXT의 타입에 따라서 아래와 같이 크기를 가집니다.

Type	Maximum length
TINYTEXT	255 (2^8−1) bytes
TEXT	65,535 (2^{16}−1) bytes = 64 KiB
MEDIUMTEXT	16,777,215 (2^{24}−1) bytes = 16 MiB
LONGTEXT	4,294,967,295 (2^{32}−1) bytes = 4 GiB

(3) Date & Time

reference : https://dev.mysql.com/doc/refman/5.7/en/date-and-time-types.html

DATE

DATE는 날짜를 저장하는 데이터 타입이며, 기본 포멧은 "년-월-일" 입니다.

DATETIME

DATETIME은 날짜와 시간을 저장하는 데이터 타입이며, 기본 포멧은 "년-월-일 시:분:초" 입니다.

TIMESTAME

TIMESTAME는 날짜와 시간을 저장하는 데이터 타입이며, DATETIME과 다른점은 날짜를 입력하지 않으면 현재 날짜와 시간을 자동으로 저장할수 있는 특징이 있습니다.

TIME

TIME은 시간을 저장하는 데이터 타입이며, 기본 포멧은 "시:분:초" 입니다.

YEAR

YEAR는 연도를 저장할수 있는 데이터 타입입니다.

YEAR(2)는 2자리의 연도를 저장할수 있으며 YEAR(4)는 4자리의 연도를 저장할수 있습니다.

2. Constraint : 제약조건

데이터 베이스의 테이블을 생성할때 각 컬럼은 각각의 제약조건을 갖습니다.

NOT NULL

NULL 값 (비어있는 값)을 저장할수 없습니다.

UNIQUE

같은 값을 저장할수 없습니다.

PRIMARY KEY

NOT NULL과 UNIQUE 의 제약조건을 동시에 만족해야 합니다. 그러므로 컬럼에 비어 있는 값과 동일한 값을 저장할수 없습니다. 하나의 테이블에 하나의 컬럼만 조건을 설정할수 있습니다.

FOREIGN KEY

다른 테이블과 연결되는 값이 저장됩니다.

DEFAULT

데이터를 저장할때 해당 컬럼에 별도의 저장값이 없으면 DEFAULT로 설정된 값이 저장됩니다.

AUTO_INCREMENT

주로 테이블의 PRIMARY KEY 데이터를 저장할때 자동으로 숫자를 1씩 증가시켜 주는 기능으로 사용합니다.

3. CREATE USE ALTER DROP (DDL)

CREATE

```
# 데이터 베이스 생성
CREATE DATABASE <database_name>;

# test 데이터 베이스 생성
CREATE DATABASE test;
```

USE

```
# test 데이터 베이스 선택
USE test;

# 현재 데이터 베이스 확인
SELECT DATABASE()

# 테이블 생성
CREATE TABLE <table_name> (
        column_name_1 column_data_type_1 column_constraint_1,
        column_name_2 column_data_type_2 column_constraint_2,
        ...
)

# 제약조건이 없는 user1 테이블 생성
CREATE TABLE user1(
        user_id INT,
        name Varchar(20),
```

```
    email Varchar(30),
    age INT(3),
    rdate DATE
)

# 제약조건이 있는 user2 테이블 생성
CREATE TABLE user2(
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    name Varchar(20) NOT NULL,
    email Varchar(30) UNIQUE NOT NULL,
    age INT(3) DEFAULT '30',
    rdate TIMESTAMP
)
```

ALTER

```
# Database

# 사용중인 데이터베이스의 인코딩 방식 확인
SHOW VARIABLES LIKE "character_set_database"

# test 데이터 베이스의 문자열 인코딩을 utf8으로 변경
ALTER DATABASE world CHARACTER SET = ascii
ALTER DATABASE world CHARACTER SET = utf8

# 사용중인 데이터베이스의 인코딩 방식 확인
SHOW VARIABLES LIKE "character_set_database"

# Table
```

ALTER를 이용하여 Table의 컬럼을 추가하거나 삭제하거나 수정할수 있습니다.

ADD

user2 테이블에 TEXT 데이터 타입을 갖는 tmp 컬럼을 추가

ALTER TABLE user2 ADD tmp TEXT

MODIFY

user2 테이블에 INT 데이터 타입을 갖는 tmp 컬럼으로 수정

ALTER TABLE user2 MODIFY COLUMN tmp INT

CONVERT TO

테이블 인코딩 확인

show full columns from test2;

테이블의 모든 인코딩 변환

ALTER TABLE user2 CONVERT TO character set utf8;

DROP

user2 테이블의 tmp 컬럼을 삭제

ALTER TABLE user2 DROP tmp;

DROP

DATABASE

tmp 데이터 베이스 생성

CREATE DATABASE tmp;

SHOW DATABASES;

tmp 데이터 베이스 삭제

DROP DATABASE tmp;

```
SHOW DATABASES;


# TABLE
# tmp 데이터 베이스 생성
CREATE DATABASE tmp;

# tmp 데이터 베이스 선택
USE tmp;

# tmp 테이블 생성
CREATE TABLE tmp( id INT );

# tmp 테이블 삭제
DROP TABLE tmp;
```

4. INSERT

테이블에 데이터를 입력할때 사용합니다. 테이블 이름 뒤에 오는 컬럼이름은 생략이 가능하며 대신에 VALUES 뒤에 value 값이 순서대로 와야 합니다.

```
INSERT INTO <table_name>(<column_name_1>, <column_name_2>, ...)
VALUES(<value_1>, <value_2>, …)

# test 데이터 베이스 선택
sql> USE test;

# user1 테이블에 user_id, namex, email, age, rdate를 입력

INSERT INTO user1(user_id, name, email, age, rdate)
```

```
VALUES (1, "jin", "pdj@gmail.com", 30, now()),
(2, "peter", "peter@daum.net", 33, '2017-02-20'),
(3, "alice", "alice@naver.com", 23, '2018-01-05'),
(4, "po", "po@gmail.com", 43, '2002-09-16'),
(5, "andy", "andy@gmail.com", 17, '2016-04-28'),
(6, "jin", "jin1224@gmail.com", 33, '2013-09-02');

# city_2 테이블 생성
CREATE TABLE city_2 (
        Name VARCHAR(50),
        CountryCode CHAR(3),
        District VARCHAR(50),
        Population INT
)

# select 절에서 나온 결과데이터를 Insert
INSERT INTO city_2
SELECT Name, CountryCode, District, Population
FROM city
WHERE Population > 8000000;
```

5. UPDATE SET

업데이트시에는 항상 select-where로 변경할 데이터를 확인하고 update 해줘야 실수를 줄일수 있습니다. 또한 limit도 함께 사용해주면 좋습니다.

```
UPDATE <table_name>
SET <column_name_1> = <value_1>, <column_name_2> = <value_2>,
WHERE <condition>
```

jin 이름을 가지고 있는 사람의 나이를 20, 이메일을 pdj@daum.net으로 변경

sql> UPDATE user1

SET age=20, email="pdj@daum.net"

WHERE name="jin"

6. DELETE, DROP, TRUNCATE

조건을 설정하여 데이터를 삭제할수 있습니다.

DELETE FROM <table_name>

WHERE <condition>

2016-01-01 이전 데이터 삭제 (DML)

sql> DELETE FROM user1

WHERE rdate < "2016-01-01"

테이블 구조를 남기고 모든 데이터를 삭제 (DLL)

sql> TRUNCATE FROM user1

테이블 전체를 모두 삭제 (DLL)

sql> DROP FROM user1

7. FOREIGN KEY

Foreign key를 설정하면 데이터의 무결성을 지킬수 있다.

UNIQUE 나 PRAMARY 제약조건이 있어야 설정이 가능하다.

```
# user 테이블 생성
create table user(
   user_id int primary key auto_increment,
    name varchar(20),
    addr varchar(20)
);

# money 테이블 생성
create table money(
    money_id int primary key auto_increment,
    income int,
    user_id int,
    # 외래키 설정
    FOREIGN KEY (user_id) REFERENCES user(user_id)
);

desc money;

# 수정해서 생성
alter table money
add constraint fk_user
foreign key (user_id)
references user (user_id);

desc money;

# 데이터 입력
```

```
insert into user(name, addr)
values ("jin", "Seoul"), ("andy", "Pusan");

# 데이터 확인
select * from user;

# 데이터 입력
insert into money(income, user_id)
values (5000, 1), (7000, 2);

# 데이터 확인
select * from money;

# user 테이블에 user_id가 3이 없으므로 에러
insert into money(income, user_id)
values (8000, 3);

delete from money
where user_id = 2;

# money 테이블에 user_id가 있어서 삭제할수 없다.
delete from user
where user_id = 1;

# 테이블 삭제도 안된다.
drop table user;
```

ON DELETE, ON UPDATE 설정

FOREIGN KEY로 참조되는 데이터를 수정 및 삭제할때 참조되는 데이터까지 수정이나 삭제하는 설정입니다.

- CASCADE : 참조되는 테이블에서 데이터를 삭제하거나 수정하면, 참조하는 테이블에서도 삭제와 수정
- SET NULL : 참조되는 테이블에서 데이터를 삭제하거나 수정하면, 참조하는 테이블의 데이터는 NULL로 변경
- NO ACTION : 참조되는 테이블에서 데이터를 삭제하거나 수정해도, 참조하는 테이블의 데이터는 변경되지 않음
- SET DEFAULT : 참조되는 테이블에서 데이터를 삭제하거나 수정하면, 참조하는 테이블의 데이터는 필드의 기본값으로 설정
- RESTRICT : 참조하는 테이블에 데이터가 남아 있으면, 참조되는 테이블의 데이터를 삭제하거나 수정할 수 없음

업데이트되면 같이 업데이트, 삭제되면 NULL 값으로 변경

```
select * from user;
drop table money;

create table money(
    money_id int primary key auto_increment,
    income int,
    user_id int,
    # 외래키 설정
    FOREIGN KEY (user_id) REFERENCES user(user_id)
    ON UPDATE CASCADE ON DELETE SET NULL
);

# money에 데이터 추가
insert into money(income, user_id)
values (5000, 1), (7000, 2);


# 데이터 추가 확인
select * from money;
```

```
# user 테이블 업데이트 : money 테이블의 user_id도 같이 업데이트 됨
update user
set user_id = 3
where user_id = 2;

select * from user;
select * from money;

# user 테이블의 데이터 삭제 : money 테이블의 fk로 설정되어 있는 데이터가 NULL로 변경
delete from user
where user_id = 3;

select * from user;
select * from money;
```

8. Functions 1 (CEIL, ROUND, TRUNCATE, DATE_FORMAT, CONCAT)

CEIL, ROUND, TRUNCATE는 소수점 올림, 반올림, 버림 함수입니다.

CEIL

CEIL는 실수 데이터를 올림 할 때 사용합니다.

```
# 12.345를 올림하여 정수로 나타냄
SELECT CEIL(12.345);

# 국가별 언어 사용 비율을 소수 첫번째자리에서 올림하여 정수로 나타냄
SELECT CountryCode, Language, Percentage, CEIL(Percentage)
```

```
FROM countrylanguage;
```

ROUND

ROUND는 실수데이터를 반올림 할 때 사용합니다.

```
# 12.345를 소수 둘째자리까지 나타내고 소수 셋째자리에서 반올림
SELECT ROUND(12.345, 2);

# 국가별 언어 사용 비율을 소수 첫번째자리에서 반올림하여 정수로 나타냄
SELECT CountryCode, Language, Percentage, ROUND(Percentage, 0)
FROM countrylanguage;
```

TRUNCATE

TRUNCATE는 실수 데이터를 버림 할 때 사용합니다.

```
# 12.345를 소수 둘째자리까지 나타내고 소수 셋째자리에서 버림
SELECT TRUNCATE(12.345, 2);

# 국가별 언어 사용 비율을 소수 첫번째자리에서 버림하여 정수로 나타냄
SELECT CountryCode, Language, Percentage, TRUNCATE(Percentage, 0)
FROM countrylanguage;

SELECT CountryCode, Language, Percentage, ROUND(Percentage, 0),
TRUNCATE(Percentage, 0)
FROM countrylanguage;
```

DATE_FORMAT

DATE_FORMAT은 날짜 데이터에 대한 포멧을 바꿔줍니다.

reference : https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html

sakila 데이터 베이스에서 월별 총 수입

```
SELECT DATE_FORMAT(payment_date, "%Y-%m") AS monthly, SUM(amount) AS amount
FROM payment
GROUP BY monthly;
```

CONCAT

concat은 문자열을 합쳐주는 기능을 합니다.

world 데이터베이스의 country 테이블에서 국가코드, 대륙이름과 국가이름이 " / " 구분자로 구분해서 하나의 컬럼에 출력

```
SELECT code, CONCAT(continent, " / ", name) as continent_name
FROM country;
```

9. Functions 2 (IF, IFNULL, CASE)

SQL에서도 다른 언어에서 처럼 조건문 사용이 가능합니다. IF, CASE 에 대해서 설명합니다.

IF

IF(조건, 참, 거짓)

도시의 인구가 100만이 넘으면 "big city" 그렇지 않으면 "small city"를 출력하는 city_scale 컬럼을 추가

```
SELECT name, population, IF(population > 1000000, "big city", "small city") AS city_scale
FROM city;
```

IFNULL

IFNULL(참, 거짓)

```
# 독립년도가 없는 데이터는 0으로 출력
SELECT IndepYear, IFNULL(IndepYear, 0) as IndepYear
FROM country;
```

CASE

```
CASE
        WHEN (조건1) THEN (출력1)
        WHEN (조건2) THEN (출력2)
END AS (컬럼명)
```

```
# 나라별로 인구가 10억 이상, 1억 이상, 1억 이하인 컬럼을 추가하여 출력
SELECT name, population,
        CASE
                WHEN population > 1000000000 THEN "upper 1 bilion"
                WHEN population > 100000000 THEN "upper 100 milion"
                ELSE "below 100 milion"
        END AS result
FROM country;
```

10. GRUOP BY HAVING

GROUP BY는 여러개의 동일한 데이터를 가지는 특정 컬럼을 합쳐주는 역할을 하는 명령입니다.

SQL에는 아래와 같은 그룹함수가 있습니다.

COUNT, MAX, MIN, AVG 등등

world 데이터 베이스로 이동

world 데이터 베이스는 city, country, countrylanguage 테이블이 있는 데이터 베이스 입니다.

sql> use world;

COUNT

city 테이블의 CountryCode를 묶고 각 코드마다 몇개의 데이터가 있는지 확인

```
sql> SELECT CountryCode, COUNT(CountryCode)
        FROM city
        GROUP BY CountryCode;
```

countrylanguage 테이블에서 전체 언어가 몇개 있는지 구하시오.

DISTINCT 중복을 제거해주는 문법

```
sql > SELECT COUNT(DISTINCT(Language)) as language_count
         FROM countrylanguage;
```

MAX

대륙별 인구수와 GNP 최대 값을 조회

```
sql> SELECT continent, MAX(Population) as Population, MAX(GNP) as GNP
        FROM country
        GROUP BY continent;
```

MIN

대륙별 인구수와 GNP 최소 값을 조회 (GNP와 인구수가 0이 아닌 데이터 중에서)

```
sql> SELECT continent, MIN(Population) as Population, MIN(GNP) as GNP
     FROM country
     WHERE GNP != 0 AND Population != 0
     GROUP BY continent;
```

SUM

대륙별 총 인구수와 총 GNP

```
sql> SELECT continent, SUM(Population) as Population, SUM(GNP) as GNP
     FROM country
     WHERE GNP != 0 AND Population != 0
     GROUP BY continent;
```

AVG

대륙별 평균 인구수와 평균 GNP 결과를 인구수로 내림차순 정렬

```
sql> SELECT continent, AVG(Population) as Population, AVG(GNP) as GNP
     FROM country
     WHERE GNP != 0 AND Population != 0
     GROUP BY continent
     ORDER BY Population DESC;
```

HAVING

GROUP BY에서 반환되는 결과에 조건을 줄수 있습니다.

대륙별 전체인구를 구하고 5억이상인 대륙만 조회

```
sql> SELECT continent, SUM(Population) as Population
        FROM country
        GROUP BY continent
        HAVING Population > 500000000;
```

대륙별 평균 인구수, 평균 GNP, 1인당 GNP한 결과를 1인당 GNP가 0.01 이상인 데이터를 조회하고 1인당 GNP를 내림차순으로 정렬

```
sql> SELECT continent, AVG(Population) as Population, AVG(GNP) as GNP,
                AVG(GNP) / AVG(Population) * 1000 as AVG
        FROM country
        WHERE GNP != 0 AND Population != 0
        GROUP BY continent
        HAVING AVG > 0.01
        ORDER BY AVG DESC;
```

WITH ROLLUP

여러개의 컬럼을 GROUP BY 하고 각 컬럼별 총 합을 row에 출력하는 방법입니다.

sakila 데이터 베이스에서 고객과 스탭별 매출과 고객별 매출의 총합을 출력

```
sql> SELECT customer_id, IFNULL(staff_id, "total") as staff_id, SUM(amount) as amount
        FROM payment
        GROUP BY customer_id, staff_id
        WITH ROLLUP;
```

world 데이터 베이스의 country 테이블에서 대륙별 지역별 전체 인구수와 대륙에 대한 전체 인구수를 출력

변수선언 : RANK 설정

```
# 변수선언
SET @data = 1;

# 선언된 변수 출력
SELECT @data;

# city 테이블에서
SET @RANK = 0;

SELECT @RANK := @RANK + 1 AS ranking, countrycode, name, population
FROM city
ORDER BY population DESC
LIMIT 5;
```

데이터 타입의 형 변환

```
# 실수 데이터 타입의 결과를 정수 데이터 타입으로 변환해서 출력
SELECT CAST(AVG(gnp) as SIGNED INTEGER)
FROM country;

SELECT CONVERT(AVG(gnp), SIGNED INTEGER)
FROM country;

# DATETIME으로 형변환
SELECT CAST("2020@01#01 12^32*21" AS DATETIME) as date;
```