MySQL

※ [DB] 무결성 제약조건 (Integrity Constraint)

▶ 무결성 (Integrity)이란?

무결성이란 데이터의 정확성, 일관성을 나타냅니다.
데이터에 결함이 없는 상태, 즉 데이터를 정확하고 일관되게 유지하는 것을 의미합니다.

▶ 무결성 제약조건 (Integrity Constraint)

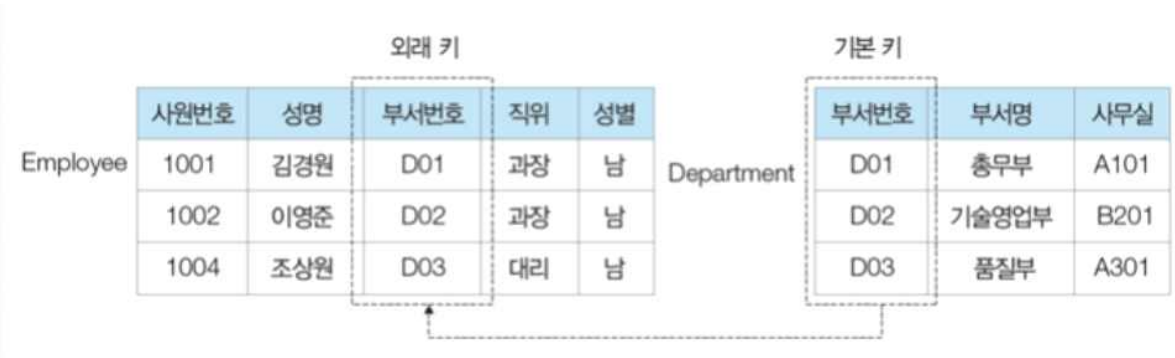
무결성 제약조건이란 데이터베이스의 정확성, 일관성을 보장하기 위해 저장, 삭제, 수정 등을 제약하기 위한 조건을 뜻합니다.
주요 목적은 데이터베이스에 저장된 데이터의 무결성을 보장하고 데이터베이스의 상태를 일관되게 유지하는 것입니다.

1. 개체 무결성

각 릴레이션의 기본키를 구성하는 속성은 널(NULL) 값이나 중복된 값을 가질 수 없습니다.

2. 참조 무결성

외래키 값은 NULL이거나 참조하는 릴레이션의 기본키 값과 동일해야 합니다.
즉, 각 릴레이션은 참조할 수 없는 외래키 값을 가질 수 없습니다.



3. 도메인 무결성

속성들의 값은 정의된 도메인에 속한 값이어야 합니다.

'성별'이라는 속성에서 '남', '여'를 제외한 데이터는 제한되어야 한다.

4. 고유 무결성

특정 속성에 대해 고유한 값을 가지도록 조건이 주어진 경우, 릴레이션의 각 튜플이 가지는 속성 값들은 서로 달라야 합니다.

[학생] 릴레이션에서 '이름', '나이'는 서로 같은 값을 가질 수 있지만 '학번'의 경우, 각 튜플은 서로 다른 값을 가져야 한다.

5. NULL 무결성

릴레이션의 특정 속성 값은 NULL 될 수 없습니다.

[학생] 릴레이션 정의 시 '과목' 속성에 NULL 값이 올 수 없도록 제한했다면 '과목' 속성에 NULL이 있어서는 안된다.

6. 키 무결성

각 릴레이션은 최소한 한 개 이상의 키가 존재해야 합니다.

1장 시작하기. (DB 생성, 테이블 생성, SELECT)

1. 데이터베이스 생성 (CREATE DATABASE)
2. 데이터베이스를 사용할 사용자 추가 (GRANT PRIVILEGES)
3. 테이블 생성 (CREATE TABLE)
4. 데이터 삽입 (INSERT)
5. 데이터 선택 (SELECT)
6. 데이터 조건 선택 (WHERE)
7. 와일드 카드 (LIKE, %, _)

1. 데이터베이스 생성 (CREATE DATABASE)

```
CREATE DATABASE study_db default CHARACTER SET UTF8mb4;
SHOW DATABASES # > #은 mysql에서 주석 입니다.
```

1 : study_db 라는 데이터베이스를 생성하고 한글을 사용할 수 있는 UTF8mb4로 문자열을 저장

2 : 데이터베이스 목록 보기

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| study_db           |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

데이터베이스의 목록을 살펴보면 우리가 만든 db 외에 다른 db가 보입니다. 그것은 mySQL의 시스템에서 사용하는 db라 신경 안 쓰셔도 됩니다.

데이터베이스의 모든 이름(데이터베이스, 테이블), 칼럼에는 소문자를 사용하는 것이 좋으며 공백 대신 _를 사용합니다.

데이터베이스 조작어 (CREATE, SELECT 등)는 대문자를 사용하는 것이 좋습니다.

2. 데이터베이스를 사용할 사용자 추가 (GRANT PRIVILEGES)

```
mysql> create user study_user@'%' identified by 'study' ;
mysql> grant all privileges on study_db.* to study_user@'%' with grant option;
mysql> flush privileges;
EXIT;
mysql> mysql –u study_user -p
USE study_db;
```

1 : GRANT는 사용자에게 데이터베이스의 사용 권한을 적용합니다.
- ALL PRIVILEGES 는 데이터베이스에 대한 모든 권한입니다. (디비 삭제도 가능)
- ON study_db.* 권한 대상은 study_db 이며, study_db.* 이라하는 것은 study_db의 모든 테이블을 의미(나중에 자세히)
- TO study_user@localhost 사용 권한을 받는 사용자는 study_user이며(없는 유저라면 새롭게 생성) localhost는 말 그대로 로컬에서만 연결 가능(127.0.0.1)
- IDENTIFIED BY 'study' 사용자의 비밀번호 설정
(locahost가 아닌 외부에서 접근하려고 한다면 접근 권한을 따로 설정해주면 가능)

2 : exit 현재 연결된 mysql을 닫습니다. (현재에는 root 권한으로 로그인 되어있었고 방금 사용자로 mysql에 접속하기 위함)

3 : mysql -u study_user -p (이렇게 입력하시면 방금 만든 study를 입력하여 사

용자 로그인 합니다)

4 : use study_db 해당 사용자가 study_db라는 데이터베이스를 사용하는 것입니다. (하나의 사용자는 여러 데이터베이스를 사용 가능합니다.)

3. 테이블 생성 (CREATE TABLE)

```
CREATE TABLE professor
(
        _id INT PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(32) NOT NULL,
        belong VARCHAR(12) DEFAULT 'FOO',
        phone VARCHAR(12)
);
DESCRIBE professor;
```

3 : _id 란 이름의 칼럼을 추가하며, 데이터 타입은 INT, PRIMARY KEY 기본키 키로 설정, AUTO_INCREMENT 자동 인덱스 증가

- auto_increment 는 테이블에 새로운 레코드가 들어올 때 사용자가 입력하지 않아도 _id 값 중 가장 큰 값에 +1 한 값을 설정해 줌으로써 중복된 값이 없도록 함

- default : 보면 모두 NULL로 되어 있는데, default 값을 따로 설정하지 않으면 모두 NULL)

4 : name 이란 칼럼을 추가, 데이터 타입은 VARCHAR(32) , NOT NULL (자료를 입력할 때 항상 값을 넣어주어야 함

5 : DEFAULT 는 아무런 값을 입력하지 않을 때 자동으로 입력되는 값

7 : 테이블 구조 확인 (줄여서 DESC professor; 도 같은 기능)

```
mysql> DESCRIBE professor;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| _id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name   | varchar(32) | NO   |     | NULL    |                |
| belong | varchar(12) | YES  |     | FOO     |                |
| phone  | varchar(12) | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

※ 기본키를 설정하는 다른 방법

```
CREATE TABLE professor
(
    _id INT AUTO_INCREMENT,
    name VARCHAR(32) NOT NULL,
    belong VARCHAR(12) DEFAULT 'FOO',
    phone VARCHAR(12),
    PRIMARY KEY(_id)
)
```

※ 데이터 타입

문자형 데이터타입

데이터 유형	정의
CHAR(n)	고정 길이 데이터 타입(최대 255byte)- 지정된 길이보다 짧은 데이터 입력될 시 나머지 공간 공백으로 채워진다.
VARCHAR(n)	가변 길이 데이터 타입(최대 65535byte)- 지정된 길이보다 짧은 데이터 입력될 시 나머지 공간은 채우지 않는다.
TINYTEXT(n)	문자열 데이터 타입(최대 255byte)
TEXT(n)	문자열 데이터 타입(최대 65535byte)
MEDIUMTEXT(n)	문자열 데이터 타입(최대 16777215byte)
LONGTEXT(n)	문자열 데이터 타입(최대 4294967295byte)

숫자형 데이터 타입

데이터 유형	정의
TINYINT(n)	정수형 데이터 타입(1byte) -128 ~ +127 또는 0 ~ 255수 표현 가능하다.
SMALLINT(n)	정수형 데이터 타입(2byte) -32768 ~ 32767 또는 0 ~ 65536수 표현 가능하다.
MEDIUMINT(n)	정수형 데이터 타입(3byte) -8388608 ~ +8388607 또는 0 ~ 16777215수 표현 가능하다.
INT(n)	정수형 데이터 타입(4byte) -2147483648 ~ +2147483647 또는 0 ~ 4294967295수 표현 가능하다.
BIGINT(n)	정수형 데이터 타입(8byte) - 무제한 수 표현 가능하다.
FLOAT(길이,소수)	부동 소수형 데이터 타입(4byte) -고정 소수점을 사용 형태이다.
DECIMAL(길이, 소수)	고정 소수형 데이터 타입고정(길이+1byte) -소수점을 사용 형태이다.
DOUBLE(길이, 소수)	부동 소수형 데이터 타입(8byte) -DOUBLE을 문자열로 저장한다.

날짜형 데이터 타입

데이터 유형	정의
DATE	날짜(년도, 월, 일) 형태의 기간 표현 데이터 타입(3byte)
TIME	시간(시, 분, 초) 형태의 기간 표현 데이터 타입(3byte)
DATETIME	날짜와 시간 형태의 기간 표현 데이터 타입(8byte)
TIMESTAMP	날짜와 시간 형태의 기간 표현 데이터 타입(4byte) -시스템 변경 시 자동으로 그 날짜와 시간이 저장된다.
YEAR	년도 표현 데이터 타입(1byte)

이진 데이터 타입

데이터 유형	정의
BINARY(n) & BYTE(n)	CHAR의 형태의 이진 데이터 타입 (최대 255byte)
VARBINARY(n)	VARCHAR의 형태의 이진 데이터 타입 (최대 65535byte)
TINYBLOB(n)	이진 데이터 타입 (최대 255byte)
BLOB(n)	이진 데이터 타입 (최대 65535byte)
MEDIUMBLOB(n)	이진 데이터 타입 (최대 16777215byte)
LONGBLOB(n)	이진 데이터 타입 (최대 4294967295byte)

4. 데이터 삽입 (INSERT)

```
INSERT INTO professor
(name, belong, phone)
VALUES('유재석', 'IDE','01012345678');

INSERT INTO professor
(name, belong, phone)
VALUES('황영조', 'MSE', '01021342443');

INSERT INTO professor
(name, belong, phone)
VALUES('케이멀', 'ESE', '01023424343');

INSERT INTO professor
(_id, name, belong, phone)
VALUES(256, '호날두', 'IME', '01034343222');
```

```
INSERT INTO professor
(name, belong, phone)
VALUES( '리오넬', 'IDE', '01023432432');
SELECT _id, belong, phone FROM professor;
SELECT * FROM professor;
```

1 : INSERT INTO professor > professor에 레코드를 삽입,

2 : (name, belong, phone) professor가 가지는 컬럼을 명시(_id는 생략했지만 AUTO_INCREMENT가 대신 값을 넣어줌)

3 : VALUES('유재석', 'IDE', '01012345678') name, belong, phone 과 같은 순서로 name : '유재석', belong : 'IDE', phone : '01012345678'

4 : 뒤 부터 해석하시면 됩니다. professor 테이블로 부터 _id, belong, phone 을 선택한다.

5 : *은 테이블이 가진 모든 필드 입니다.

호날두를 입력하는 4번째 입력을 보라, _id를 직접 입력한다. 물론 가능하다. 그리고 다음에 리오넬을 입력할 때는 _id를 생략했다. 어떻게 됐을까??

```
mysql> select * from professor;
+-----+-----------+--------+-------------+
| _id | name      | belong | phone       |
+-----+-----------+--------+-------------+
|   1 | 유재석    | IDE    | 01112345678 |
|   2 | 황영조    | MSE    | 01121342443 |
|   3 | 케이멀    | ESE    | 01123424343 |
| 256 | 호날두    | IME    | 01134343222 |
| 257 | 리오넬    | IDE    | 01123432432 |
+-----+-----------+--------+-------------+
5 rows in set (0.00 sec)
```

유재석을 입력했을 때 테이블에는 아무런 값이 없다. AUTO_INCEREMENT는 1 값부터 넣는다고 생각하면 된다. 그러면 호날두의 경우 임의로 256 이라는 값을 설정했다. 그러면 그 뒤에 _id값이 없이 입력되면 테이블 내 _id값 중 가장 큰 값에 + 1 한 값으로 채워지게 된다.

지금은 모든 테이블 내에 모든 데이터를 가져왔지만, 우리가 필요한 데이터만 얻는 것은 매우 중요한 일입니다. 지금 부터 우리가 원하는 데이터를 찾아보도록 하겠습니다.

학생 테이블을 생성하고 데이터를 추가합니다.

```
CREATE TABLE student
(
	_id CHAR(9),
	name VARCHAR(48) NOT NULL,
	belong VARCHAR(5),
	phone VARCHAR(11),
	status INT DEFAULT 0
);
INSERT INTO student VALUES('20090101', '루피', 'IDE', '01012345678', 1);
INSERT INTO student VALUES('20100102', '조로', 'CSE', '01023435343', 4);
INSERT INTO student VALUES('20110103', '상디', 'MSE', '01021342443', 1);
INSERT INTO student VALUES('20100204', '버기', 'ESE', '01023424343', 2);
INSERT INTO student VALUES('20110106', '프랑키', 'IME', '01034343222' , 0);
INSERT INTO student VALUES('20080104', '나미', 'IDE', '01023432432', 6);
INSERT INTO student VALUES('20090105', '쵸파', 'CSE', '01012342433', 8);
INSERT INTO student VALUES('20090301', '에릭', 'ESE', '01032424244', 5);
INSERT INTO student VALUES('20090302', '전진', 'IDE', '01012321313', 3);
INSERT INTO student VALUES('20100505', '오공', 'CSE', '01023534644', 2);
INSERT INTO student VALUES('20110506', '오천', 'MSE', '01021334525', 8);
INSERT INTO student VALUES('20100507', '베지터', 'ESE', '01023423623', 0);
```

```
INSERT INTO student VALUES('20110502', '부우', 'IME', '01034332634', 1);
INSERT INTO student VALUES('20080501', '크리링', 'IDE', '01023436346', 2);
INSERT INTO student VALUES('20090503', '피콜로', 'CSE', '01013634645', 3);
INSERT INTO student VALUES('20090509', '셀', 'ESE', '01032427535', 0);
```

위 테이블 생성문은 크게 어렵지 않을 것입니다.

TIP : mySQL은 데이터 타입에 상관없이 DEAFULT 가 NULL 입니다.

삽입할 때 삽입할 칼럼을 생략할 수 있는데, 이 때는 모든 필드를 채워야 합니다.

5. 데이터 선택 (SELECT)

데이터를 선택할 때는 선택할 데이터의 칼럼을 선택하거나 레코드의 순서도 지정할 수 있습니다.

SELECT 할 때 *을 사용하면 모든 데이터를 가져올 수 있습니다. 그러나 원하는 데이터만 가져오는 것은 매우 중요합니다.

```
mysql> SELECT name, _id FROM student;
+-----------+----------+
| name      | _id      |
+-----------+----------+
| 루피      | 20090101 |
| 조로      | 20100102 |
| 상디      | 20110103 |
| 버기      | 20100204 |
| 프랑키    | 20110106 |
| 나미      | 20080104 |
| 쵸파      | 20090105 |
| 에릭      | 20090301 |
| 전진      | 20090302 |
| 오공      | 20100505 |
| 오천      | 20110506 |
| 베지터    | 20100507 |
| 부우      | 20110502 |
| 크리링    | 20080501 |
| 피콜로    | 20090503 |
| 셀        | 20090509 |
+-----------+----------+
16 rows in set (0.00 sec)
```

SELECT 에서 입력한 칼럼 순서로 출력되는 것을 확인할 수 있습니다.

```
SELECT * FROM student;
SELECT * FROM student ORDER BY _id;
```

2 : ORDER BY 라는 키워드가 처음 나왔습니다. 가져온 데이터를 특정 칼럼을 기준으로 정렬하는 것이죠.

```
mysql> SELECT * FROM student ORDER BY _id;
+----------+-----------+--------+-------------+--------+
| _id      | name      | belong | phone       | status |
+----------+-----------+--------+-------------+--------+
| 20080104 | 나미      | IDE    | 01123432432 |      0 |
| 20080501 | 크리링    | IDE    | 01123436346 |      2 |
| 20090101 | 루피      | IDE    | 01112345678 |      0 |
| 20090105 | 쵸파      | CSE    | 01112342433 |      0 |
| 20090301 | 에릭      | ESE    | 01132424244 |      0 |
| 20090302 | 전진      | IDE    | 01112321313 |      0 |
| 20090503 | 피콜로    | CSE    | 01113634645 |      3 |
| 20090509 | 셀        | ESE    | 01132427535 |      0 |
| 20100102 | 조로      | CSE    | 01123435343 |      0 |
| 20100204 | 버기      | ESE    | 01123424343 |      0 |
| 20100505 | 오공      | CSE    | 01123534644 |      0 |
| 20100507 | 베지터    | ESE    | 01123423623 |      0 |
| 20110103 | 상디      | MSE    | 01121342443 |      1 |
| 20110106 | 프랑키    | IME    | 01134343222 |      0 |
| 20110502 | 부우      | IME    | 01134332634 |      1 |
| 20110506 | 오천      | MSE    | 01121334525 |      0 |
+----------+-----------+--------+-------------+--------+
16 rows in set (0.00 sec)
```

6. 데이터 조건 선택 (WHERE)

원하는 데이터를 가져오는 것은 매우 중요합니다. 다음은 기본 조건을 살펴보겠습니다.

belong 이 'IDE'인 데이터를 뽑아보겠습니다.

```
SELECT * FROM student WHERE belong = 'IDE';
```

WHERE은 조건을 나타냅니다. 위 예시에서는 belong값이 IDE와 일치하는 것을 찾았습니다.

```
mysql> SELECT * FROM student WHERE belong = 'IDE';
+----------+-----------+--------+-------------+--------+
| _id      | name      | belong | phone       | status |
+----------+-----------+--------+-------------+--------+
| 20090101 | 루피      | IDE    | 01112345678 |      0 |
| 20080104 | 나미      | IDE    | 01123432432 |      0 |
| 20090302 | 전진      | IDE    | 01112321313 |      0 |
| 20080501 | 크리링    | IDE    | 01123436346 |      2 |
+----------+-----------+--------+-------------+--------+
4 rows in set (0.01 sec)
```

다음의 결과를 예상하실 수 있나요?

```
SELECT * FROM student WHERE status > 0;
```

```
mysql> SELECT * FROM student WHERE status > 0;
+----------+-----------+--------+-------------+--------+
| _id      | name      | belong | phone       | status |
+----------+-----------+--------+-------------+--------+
| 20110103 | 상디      | MSE    | 01121342443 |      1 |
| 20110502 | 부우      | IME    | 01134332634 |      1 |
| 20080501 | 크리링    | IDE    | 01123436346 |      2 |
| 20090503 | 피콜로    | CSE    | 01113634645 |      3 |
+----------+-----------+--------+-------------+--------+
4 rows in set (0.00 sec)
```

이러한 조건은 AND, OR 연산도 가능합니다.

```
SELECT * FROM student WHERE status > 0 AND belong = 'IDE'
SELECT * FROM student WHERE status > 0 OR belong = 'IDE';
```

다음의 결과를 예상해보시겠어요?

그리고 이런건 어떤가요?

```
SELECT * FROM student WHERE _id > '2010';
```

문자열도 비교 연산이 가능합니다. 그러나 숫자와는 다릅니다. 사전의 순서라고 생각하시면 편합니다.

(참고 , 순서 : !"#$%&'()*+,-./0-9:;<=>?@A-Z[₩]^_`a-z{|}

7. 와일드 카드 (LIKE, %, _)

와일드카드라는 개념아시나요? 트럼프에서 조커 처럼 어떤 카드든 될 수 있는 카드입니다.

'%', '_' 두 가지에 대해 알아보겠습니다.

먼저, '%'

```
SELECT * FROM student WHERE _id LIKE '2009%';
```

_id 가 2009로 시작하는 모든 값을 가져옵니다.

```
mysql> SELECT * FROM student WHERE _id LIKE '2009%';
+----------+-----------+--------+-------------+--------+
| _id      | name      | belong | phone       | status |
+----------+-----------+--------+-------------+--------+
| 20090101 | 루피      | IDE    | 01112345678 |      0 |
| 20090105 | 쵸파      | CSE    | 01112342433 |      0 |
| 20090301 | 에릭      | ESE    | 01132424244 |      0 |
| 20090302 | 전진      | IDE    | 01112321313 |      0 |
| 20090503 | 피콜로    | CSE    | 01113634645 |      3 |
| 20090509 | 셀        | ESE    | 01132427535 |      0 |
+----------+-----------+--------+-------------+--------+
6 rows in set (0.01 sec)
```

그리고 , '_'

```
SELECT * FROM student WHERE NOT _id > '2010';
SELECT * FROM student WHERE _id <= '2010';
#위와 완전 똑같은 식입니다.
SELECT * FROM student WHERE NOT belong = 'IDE';
SELECT * FROM student WHERE belong <> 'IDE';
# 위와 완전 똑같은 식입니다. <> 은 != 을 의미 합니다.

SELECT * FROM student WHERE NOT _id LIKE '2009%';
SELECT * FROM student WHERE _id NOT LIKE '2009%'; (권장)
```

아래 LIKE 에 대한 NOT은 두 표현 모두 mySQL에서는 같은 결과를 보여줬습니다. 그러나 아래의 표현이 더 일반적인 표현입니다.

2장 검색하기

1. 조건 검색, 효율적인 방법 (BETWEEN, IN)
2. 데이터베이스, 테이블, 데이터 삭제 (DROP, DELETE)
3. 테이블 변경 (UPDATE, CHANGE)
4. 데이터 변경
5. 생성문 보기 (SHOW)

1. 조건 검색, 효율적인 방법 (BETWEEN, IN)

저번 시간에 배웠던 WHERE 절을 AND 또는 OR로 연결하는 것을 기억할 것이다.

아래와 같은 student 테이블이 있을 때

```
mysql> SELECT * FROM student ORDER BY _id;
+----------+-----------+--------+-------------+--------+
| _id      | name      | belong | phone       | status |
+----------+-----------+--------+-------------+--------+
| 20080104 | 나미      | IDE    | 01123432432 |      0 |
| 20080501 | 크리링    | IDE    | 01123436346 |      2 |
| 20090101 | 루피      | IDE    | 01112345678 |      0 |
| 20090105 | 쵸파      | CSE    | 01112342433 |      0 |
| 20090301 | 에릭      | ESE    | 01132424244 |      0 |
| 20090302 | 전진      | IDE    | 01112321313 |      0 |
| 20090503 | 피콜로    | CSE    | 01113634645 |      3 |
| 20090509 | 셀        | ESE    | 01132427535 |      0 |
| 20100102 | 조로      | CSE    | 01123435343 |      0 |
| 20100204 | 버기      | ESE    | 01123424343 |      0 |
| 20100505 | 오공      | CSE    | 01123534644 |      0 |
| 20100507 | 베지터    | ESE    | 01123423623 |      0 |
| 20110103 | 상디      | MSE    | 01121342443 |      1 |
| 20110106 | 프랑키    | IME    | 01134343222 |      0 |
| 20110502 | 부우      | IME    | 01134332634 |      1 |
| 20110506 | 오천      | MSE    | 01121334525 |      0 |
+----------+-----------+--------+-------------+--------+
16 rows in set (0.00 sec)
```

저번에 배웠던 조건절을 리마인딩 해보자.

```
SELECT * FROM student
WHERE _id > '2010';
# 학번이 2010 으로 시작하는 학생을 검색할 수 있다.
```

그렇다면 2009 ~ 2010 학번인 학생을 구할 때는 어떻게 하는 것이 좋을까?
아래의 두 예를 보자.

```
SELECT * FROM student
WHERE _id > '2009' AND _id < '2011';

SELECT * FROM student
WHERE _id BETWEEN '2009' AND '2011';
```

두 쿼리 모두 같은 결과를 내놓는다. 그러나 두 쿼리의 성능은 다르다. '2009'보다 큰 값을 모두 구하고, '2011' 보다 작은 값을 모두 구하여 교집합을 다시 구하는 연산과,

'2009' 부터 시작하여 '2011'보다 작을 때 까지 순차적으로 구하는 것과는 차이가 있을 것이기 때문이다.

또, 아래의 예도 살펴 보자.

```
SELECT * FROM student
WHERE belong = 'MSE' OR belong = 'ESE' OR belong = 'CSE';

SELECT * FROM student
WHERE belong IN ('MSE', 'ESE', 'CSE');
```

이 둘 또한 같은 결과를 내놓는다. 그러나 아래의 예가 더욱 좋다. 간단히 말 해 첫 번 째 쿼리는 for 문을 총 3번 돈다면, 아래의 쿼리는 for문을 한 번 도는 것 이기 때문이다.

2. 데이터베이스, 테이블 삭제

```
DROP DATABASE db_name;
DROP TABLE table_name;

DELETE FROM table_name;
DELETE FROM table_name WHERE _id = 20070103;
```

데이터베이스와 테이블을 제거할 때는 DROP을, 테이블의 데이터를 제거할 때는 DELETE를 사용한다. **데이터가 한 번 삭제되면 복구할 수 없으므로 삭제할 때는 조심히 해야 한다.**

데이터를 삭제할 때는 원하는 데이터를 SELECT 할 때와 마찬가지로 조건을 주어 삭제할 수 있다. (삭제할 때 SELECT를 통해 결과를 확인 후 DELETE하는 습관을 가지는 것이 좋다)

3. 테이블 변경 (ALTER)

테이블 변경에서는 아래의 일을 할 수 있다.

- 테이블 이름 변경 (RENAME)
- 테이블 칼럼 , 제약조건 추가 (ADD)
- 테이블 변경 (CHANGE, MODIFY)
- 테이블 제약 조건 제거 (DROP)

우선 현재 테이블의 상태를 확인해보자.

```
mysql> DESC student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| _id    | char(9)     | YES  |     | NULL    |       |
| name   | varchar(48) | NO   |     | NULL    |       |
| belong | varchar(5)  | YES  |     | NULL    |       |
| phone  | varchar(11) | YES  |     | NULL    |       |
| status | int(11)     | YES  |     | 0       |       |
+--------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

A. 테이블 이름 변경 (RENAME)

```
ALTER TABLE student
RENAME TO hero;
DESC hero;
```

1 : student라는 테이블을 변경

2 : hero라는 새로운 이름으로 변경

B. 테이블 칼럼, 제약조건 추가 (ADD)

```
ALTER TABLE hero
ADD COLUMN star INT NOT NULL AFTER _id,
ADD PRIMARY KEY (_id);
DESC hero;
```

```
mysql> DESC hero;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| _id    | char(9)     | NO   | PRI | NULL    |       |
| name   | varchar(48) | NO   |     | NULL    |       |
| belong | varchar(5)  | YES  |     | NULL    |       |
| phone  | varchar(11) | YES  |     | NULL    |       |
| status | int(11)     | YES  |     | 0       |       |
| star   | int(11)     | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

start 라는 새로운 칼럼이 추가가 되었고, _id가 기본키로 변경되었다. 특이한 점은 AFTER _id 로 추가될 칼럼의 위치를 선정하는 점이다.

MySQL에서는 FIRST를 사용하면 첫 번째 칼럼으로 추가된다. (첫 번째 칼럼은 보통 기본키다.) 그런데 책에서는 SECOND, THIRD, LAST, BEFORE 등의

C. 테이블 변경 (CHANGE, MODIFY)

```
ALTER TABLE hero
MODIFY COLUMN star BIGINT DEFAULT 10000;
DESC hero;

ALTER TABLE hero
CHANGE COLUMN star rating INT DEFAULT 100;
DESC hero;
```

MODIFY와 CHANGE 모두 테이블의 칼럼의 상태를 변경하는 공통점이 있다. 그러나 CHANGE는 칼럼의 이름도 변경이 가능 하다.

D. 테이블 제약 조건 제거 (DROP)

```
ALTER TABLE hero
DROP COLUMN star,
DROP PRIMARY KEY;
DESC hero
```

테이블에서 star 칼럼을 제거하고, 기본키를 제거 한다.

4. 데이터 변경 (UPDATE)

```
UPDATE hero SET belong = 'ZER' WHERE status = 0;
SELECT * FROM hero;
UPDATE hero SET status = status + 200 WHERE _id LIKE '2009%';
SELECT * FROM hero;
```

첫 번째 쿼리는 status 값이 0인 레코드의 belong을 'ZER'로 바꾼다.

```
mysql> SELECT * FROM hero;
+----------+-----------+--------+-------------+--------+
| _id      | name      | belong | phone       | status |
+----------+-----------+--------+-------------+--------+
| 20080104 | 나미      | ZER    | 01123432432 |      0 |
| 20080501 | 크리링    | IDE    | 01123436346 |      2 |
| 20090101 | 루피      | ZER    | 01112345678 |      0 |
| 20090105 | 쵸파      | ZER    | 01112342433 |      0 |
| 20090301 | 에릭      | ZER    | 01132424244 |      0 |
| 20090302 | 전진      | ZER    | 01112321313 |      0 |
| 20090503 | 피콜로    | CSE    | 01113634645 |      3 |
| 20090509 | 셀        | ZER    | 01132427535 |      0 |
| 20100102 | 조로      | ZER    | 01123435343 |      0 |
| 20100204 | 버기      | ZER    | 01123424343 |      0 |
| 20100505 | 오공      | ZER    | 01123534644 |      0 |
| 20100507 | 베지터    | ZER    | 01123423623 |      0 |
| 20110103 | 상디      | MSE    | 01121342443 |      1 |
| 20110106 | 프랑키    | ZER    | 01134343222 |      0 |
| 20110502 | 부우      | IME    | 01134332634 |      1 |
| 20110506 | 오천      | ZER    | 01121334525 |      0 |
+----------+-----------+--------+-------------+--------+
16 rows in set (0.00 sec)
```

두 번째 쿼리는 _id 가 '2009'로 시작하는 레코드의 status 값을 현재 값에서 200 만큼 더한 값으로 변경한다.

```
mysql> SELECT * FROM hero;
+----------+-----------+--------+-------------+--------+
| _id      | name      | belong | phone       | status |
+----------+-----------+--------+-------------+--------+
| 20080104 | 나미      | ZER    | 01123432432 |      0 |
| 20080501 | 크리링    | IDE    | 01123436346 |      2 |
| 20090101 | 루피      | ZER    | 01112345678 |    200 |
| 20090105 | 쵸파      | ZER    | 01112342433 |    200 |
| 20090301 | 에릭      | ZER    | 01132424244 |    200 |
| 20090302 | 전진      | ZER    | 01112321313 |    200 |
| 20090503 | 피콜로    | CSE    | 01113634645 |    203 |
| 20090509 | 셀        | ZER    | 01132427535 |    200 |
| 20100102 | 조로      | ZER    | 01123435343 |      0 |
| 20100204 | 버기      | ZER    | 01123424343 |      0 |
| 20100505 | 오공      | ZER    | 01123534644 |      0 |
| 20100507 | 베지터    | ZER    | 01123423623 |      0 |
| 20110103 | 상디      | MSE    | 01121342443 |      1 |
| 20110106 | 프랑키    | ZER    | 01134343222 |      0 |
| 20110502 | 부우      | IME    | 01134332634 |      1 |
| 20110506 | 오천      | ZER    | 01121334525 |      0 |
+----------+-----------+--------+-------------+--------+
16 rows in set (0.00 sec)
```

5. 생성문 보기 (SHOW)

```
SHOW CREATE DATABASE study_db;
SHOW CREATE TABLE hero;
SHOW INDEX FROM hero;
SHOW WARNINGS;
```

```
mysql> SHOW CREATE DATABASE study_db;
+----------+------------------------------------------------------------------+
| Database | Create Database                                                  |
+----------+------------------------------------------------------------------+
| study_db | CREATE DATABASE `study_db` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+----------+------------------------------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql> SHOW CREATE TABLE hero;
+-------+-----------------------------------------------------------------
| Table | Create Table
                                                                         |
+-------+-----------------------------------------------------------------
| hero  | CREATE TABLE `hero` (
  `_id` char(9) NOT NULL,
  `name` varchar(48) NOT NULL,
  `belong` varchar(5) DEFAULT NULL,
  `phone` varchar(11) DEFAULT NULL,
  `status` int(11) DEFAULT '0'
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-------+-----------------------------------------------------------------
1 row in set (0.00 sec)
```

각 테이블, 칼럼 이름 사이에 `로 감싸진 것을 볼 수 있다. 지금은 변수를 `로 감싸는구나 하고 생각하자. 저 구문을 복사하여 그대로 `가 있는 상태로 사용해도 상관없고, 제거해서 사용해도 좋다.

3장 내부 함수, 그룹 묶기
(SUM, COUNT, SUBSTRING, GROUP BY)

저번 시간을 통해 데이터를 선택하고, 변경하는 것에 대해 알아봤다. 이번 시간에 알아볼 것은 mySQL 에서 효율적으로 데이터를 처리하는 방법에 대해 공부하려 한다.

데이터베이스에서 쓰지 않는 데이터를 모두 가져오는 일은 좋은 것이 아니다. 테이블 조차도 필요한 칼럼에 중복을 최소화 하는 것이 좋다.

1. 내부 함수 사용하기 (SUM, COUNT, AVG)
2. 문자열을 다루는 내부함수 사용하기 (RIGHT, SUBSTRING_INDEX, UPPER)
3. 데이터의 그룹별 선택 (GROUP BY)

현재 student 테이블의 내용입니다. (2장에서 hero로 바꾸셨던 분은 같은 방법으로 student로 바꾸시길 바랍니다.)

```
mysql> SELECT * FROM student;
+----------+-----------+--------+-------------+--------+
| _id      | name      | belong | phone       | status |
+----------+-----------+--------+-------------+--------+
| 20090101 | 루피      | IDE    | 01112345678 |      1 |
| 20100102 | 조로      | CSE    | 01123435343 |      4 |
| 20110103 | 상디      | MSE    | 01121342443 |      1 |
| 20100204 | 버기      | ESE    | 01123424343 |      2 |
| 20110106 | 프랑키    | IME    | 01134343222 |      0 |
| 20080104 | 나미      | IDE    | 01123432432 |      6 |
| 20090105 | 쵸파      | CSE    | 01112342433 |      8 |
| 20090301 | 에릭      | ESE    | 01132424244 |      5 |
| 20090302 | 전진      | IDE    | 01112321313 |      3 |
| 20100505 | 오공      | CSE    | 01123534644 |      2 |
| 20110506 | 오천      | MSE    | 01121334525 |      8 |
| 20100507 | 베지터    | ESE    | 01123423623 |      0 |
| 20110502 | 부우      | IME    | 01134332634 |      1 |
| 20080501 | 크리링    | IDE    | 01123436346 |      2 |
| 20090503 | 피콜로    | CSE    | 01113634645 |      3 |
| 20090509 | 셀        | ESE    | 01132427535 |      0 |
+----------+-----------+--------+-------------+--------+
16 rows in set (0.00 sec)
```

1. 내부 함수를 사용하기 (SUM, COUNT, AVG)

내부 함수를 사용하면 데이터를 효율적으로 처리할 수 있습니다. 다음의 경우를 생각해봅시다.

→ belong 이 'IDE'인 사람이 몇명인지 알고 싶다면??
→ belong 인 'IDE'인 사람들의 status 값의 합은?

```
SELECT * FROM student
WHERE belong = 'IDE';

SELECT COUNT(status) FROM student
WHERE belong = 'IDE';

SELECT SUM(status) FROM student
WHERE belong = 'IDE';
```

첫 번째 쿼리 : 저번 시간에 배웠던 belong 이 'IDE' 값들을 택하는 쿼리다.
두 번째 쿼리 : 결과 4, 선택된 아이템의 개수
세 번째 쿼리 : 선택된 결과들의 status 총합

```
mysql> SELECT * FROM student
    -> WHERE belong = 'IDE';
+----------+-----------+--------+-------------+--------+
| _id      | name      | belong | phone       | status |
+----------+-----------+--------+-------------+--------+
| 20090101 | 루피      | IDE    | 01112345678 |      1 |
| 20080104 | 나미      | IDE    | 01123432432 |      6 |
| 20090302 | 전진      | IDE    | 01112321313 |      3 |
| 20080501 | 크리링    | IDE    | 01123436346 |      2 |
+----------+-----------+--------+-------------+--------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT COUNT(status) FROM student
    -> WHERE belong = 'IDE';
+---------------+
| COUNT(status) |
+---------------+
|             4 |
+---------------+
1 row in set (0.00 sec)

mysql> SELECT SUM(status) FROM student WHERE belong = 'IDE';
+-------------+
| SUM(status) |
+-------------+
|          12 |
+-------------+
1 row in set (0.00 sec)
```

COUNT, SUM 외에도 AVG(평균), MIN(최소값), MAX(최대값) 도 사용할 수 있다.

2. 문자열을 다루는 내부함수 사용하기 (RIGHT, SUBSTRING_INDEX, UPPER)

흔히, 학번을 물어보면, 11학번, 16학번 이런식으로 대답을 하게 된다. 우리는 _id 값을 통해서 학번을 내부함수를 통해 구해볼 것이다.

예시를 보면 '20100303' 에서 학번을 나타내는 값은 **10**** 이다. 우리는 저 두 문자를 통해 학번을 알 수 있다.

```
SELECT _id, phone, RIGHT(phone, 4)
FROM student
WHERE belong = 'IDE';

SELECT _id, phone, RIGHT(phone, 4) AS phone_last
FROM student
WHERE belong = 'IDE';
```

```
SELECT _id, SUBSTRING(_id, 3, 6)
FROM student
WHERE belong = 'IDE';

SELECT _id, SUBSTRING_INDEX(_id, '1', 2)
FROM student
WHERE belong = 'IDE';
```

첫 쿼리와 두 번째 쿼리의 차이점이 보이는가?

```
mysql> SELECT _id, phone, RIGHT(phone, 4)
    -> FROM student
    -> WHERE belong = 'IDE';
+----------+-------------+-----------------+
| _id      | phone       | RIGHT(phone, 4) |
+----------+-------------+-----------------+
| 20090101 | 01112345678 | 5678            |
| 20080104 | 01123432432 | 2432            |
| 20090302 | 01112321313 | 1313            |
| 20080501 | 01123436346 | 6346            |
+----------+-------------+-----------------+
4 rows in set (0.00 sec)

mysql> SELECT _id, phone, RIGHT(phone, 4) AS phone_last
    -> FROM student
    -> WHERE belong = 'IDE';
+----------+-------------+------------+
| _id      | phone       | phone_last |
+----------+-------------+------------+
| 20090101 | 01112345678 | 5678       |
| 20080104 | 01123432432 | 2432       |
| 20090302 | 01112321313 | 1313       |
| 20080501 | 01123436346 | 6346       |
+----------+-------------+------------+
4 rows in set (0.00 sec)
```

실제 사용 시 우리는 쿼리 결과의 칼럼명을 통해 필드값에 접근한다. 이럴 때 의미에 맞는 변수명을 지어줄 때 AS를 사용할 수 있다. AS는 쿼리를 더욱 간결하게 만들 때도 사용한다. 그것은 나중에 다루도록 하겠다.

```
mysql> SELECT _id, SUBSTRING(_id, 3, 6)
    -> FROM student
    -> WHERE belong = 'IDE';
+----------+----------------------+
| _id      | SUBSTRING(_id, 3, 6) |
+----------+----------------------+
| 20090101 | 090101               |
| 20080104 | 080104               |
| 20090302 | 090302               |
| 20080501 | 080501               |
+----------+----------------------+
4 rows in set (0.00 sec)

mysql> SELECT _id, SUBSTRING_INDEX(_id, '1', 2)
    -> FROM student
    -> WHERE belong = 'IDE';
+----------+-------------------------------+
| _id      | SUBSTRING_INDEX(_id, '1', 2) |
+----------+-------------------------------+
| 20090101 | 2009010                       |
| 20080104 | 20080104                      |
| 20090302 | 20090302                      |
| 20080501 | 20080501                      |
+----------+-------------------------------+
4 rows in set (0.00 sec)
```

SUBSTRING('20100103', 3, 3) : 3 번째 위치부터 3개의 문자열을 반환 -> '100'

SUBSTRING_INDEX('20100103', '1', 2) : '1'문자가 두 번째로 나오기 전까지 모든 문자열 -> '20100'

이것 외에도 문자열을 다루는 내부 함수는 UPPER, LOWER, REVERSE, LTRIM, LENGTH 등이 있다.

3. 데이터의 그룹별 선택 (GROUP BY)

이런 경우는 어떨까? 만약 특정 belong 별로 인원수가 몇명인지 알고 싶을 때.

```
SELECT belong, COUNT(*)
FROM student
GROUP BY belong;
```

```
mysql> SELECT belong, COUNT(*) FROM student GROUP BY belong;
+--------+----------+
| belong | COUNT(*) |
+--------+----------+
| CSE    |        4 |
| ESE    |        4 |
| IDE    |        4 |
| IME    |        2 |
| MSE    |        2 |
+--------+----------+
5 rows in set (0.00 sec)
```

쿼리 결과는 다음과 같다. belong을 기준으로 값은 값 끼리 묶어서 가져온다.
그러나 GROUP BY는 주의해야 할 점이 있다. 아래의 결과는 오류를 만든다.

```
SELECT name, belong, COUNT(*)
FROM student
GROUP BY belong;
```

그룹을 지었을 때 name을 특정 지을 수 없기 때문이다.
그러면 아래의 쿼리 결과를 예측해보자.

```
SELECT belong, COUNT(*) , MAX(status)
FROM student
GROUP BY belong
ORDER BY MAX(status) DESC, COUNT(*) ASC;
```

```
mysql> SELECT belong, COUNT(*) , MAX(status)
    -> FROM student
    -> GROUP BY belong
    -> ORDER BY MAX(status) DESC, COUNT(*) ASC;
+--------+----------+-------------+
| belong | COUNT(*) | MAX(status) |
+--------+----------+-------------+
| MSE    |        2 |           8 |
| CSE    |        4 |           8 |
| IDE    |        4 |           6 |
| ESE    |        4 |           5 |
| IME    |        2 |           1 |
+--------+----------+-------------+
5 rows in set (0.00 sec)
```

쿼리에 GROUP BY, ORDER BY 등이 모두 등장하여 조금 어려울 수 있지만, 차례대로 생각해보자. 우선 GROUP BY 를 이용하여 belong값으로 묶었다.

우리가 얻은 결과값은 belong, 각 그룹에 속한 아이템의 개수, 그룹 내 status의 최대값을 볼 수 있다. 우리는 이 결과를 얻은 3가지 필드값으로 정렬할 수 있다.

여기서 ORDER BY에 두 개의 기준이 제시된다. 첫 째, MAX(status)값을 통해서 내림차순으로 정렬된다. 여기서 중복값이 있는 것들 중에 내부 정렬을 두 번째기준인 COUNT(*) 오름차순으로 하게 되는 것이다.

결과적으로 최대값이 가장 큰 값을 기준으로 정렬되고, 가장 큰 값이 중복이 된다면, 사람 수가 적은 그룹이 우선순위가 되는 것이다.

4장 조인 : JOIN (INNER, LEFT, RIGHT)

조인은 관계형 데이터베이스에서 일반적으로 많이 사용하며, 충분한 이해 없이 사용하기 쉬운 내용이다.

이번 시간에는 여러 조인의 종류가 그 사용하는 예에 대해서 공부해보자.

1. INNER 조인
2. LEFT OUTER, RIGHT OUTER, OUTER 조인
3. 카티전 조인 (CROSS 조인)
4. 셀프 조인

우선 JOIN을 할 두 테이블을 생성하자.

```
CREATE TABLE girl_group
(
  _id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(32) NOT NULL,
  debut DATE NOT NULL,
  hit_song_id INT
);

CREATE TABLE song
(
  _id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(32) NOT NULL,
  lyrics VARCHAR(32)
);
INSERT INTO song (_id, title, lyrics) VALUES (101, 'Tell Me', 'tell me tell me tetetete tel me');
INSERT INTO song (title, lyrics) VALUES ('Gee', 'GEE GEE GEE GEE GEE BABY BABY');
```

```
INSERT INTO song (title, lyrics) VALUES ('미스터', '이름이 뭐야 미스터');
INSERT INTO song (title, lyrics) VALUES ('Abracadabra', '이러다 미쳐 내가 여리여리');
INSERT INTO song (title, lyrics) VALUES ('8282', 'Give me a call Baby baby');
INSERT INTO song (title, lyrics) VALUES ('기대해', '기대해');
INSERT INTO song (title, lyrics) VALUES ('I Don₩'t car', '다른 여자들의 다리를');
INSERT INTO song (title, lyrics) VALUES ('Bad Girl Good Girl', '앞에선 한 마디 말도');
INSERT INTO song (title, lyrics) VALUES ('피노키오', '뉴예삐오');
INSERT INTO song (title, lyrics) VALUES ('별빛달빛', '너는 내 별빛 내 마음의 별빛');
INSERT INTO song (title, lyrics) VALUES ('A', 'A 워오우 워오우워 우우우');
INSERT INTO song (title, lyrics) VALUES ('나혼자', '나 혼자 밥을 먹고 나 혼자 영화 보고');
INSERT INTO song (title, lyrics) VALUES ('LUV', '설레이나요 ');
INSERT INTO song (title, lyrics) VALUES ('짧은치마', '짧은 치마를 입고 내가 길을 걸으면');
INSERT INTO song (title, lyrics) VALUES ('위아래', '위 아래 위위 아래');
INSERT INTO song (title, lyrics) VALUES ('Dumb Dumb' , '너 땜에 하루종일');
INSERT INTO girl_group (name, debut, hit_song_id)
VALUES ('원더걸스', '2007-09-12', 101);
INSERT INTO girl_group (name, debut, hit_song_id)
VALUES ('소녀시대', '2009-06-03', 102);
INSERT INTO girl_group (name, debut, hit_song_id)
VALUES ('카라', '2009-07-30', 103);
INSERT INTO girl_group (name, debut, hit_song_id)
VALUES ('브라운아이드걸스', '2008-01-17', 104);
INSERT INTO girl_group (name, debut, hit_song_id)
VALUES ('다비치', '2009-02-27', 105);
```

```
INSERT INTO girl_group (name, debut, hit_song_id)
VALUES ('2NE1', '2009-07-08', 107);
INSERT INTO girl_group (name, debut, hit_song_id)
VALUES ('f(x)', '2011-04-20', 109);
INSERT INTO girl_group (name, debut, hit_song_id)
VALUES ('시크릿', '2011-01-06', 110);
INSERT INTO girl_group (name, debut, hit_song_id)
VALUES ('레인보우', '2010-08-12', 111);
INSERT INTO girl_group (name, debut)
VALUES ('에프터 스쿨', '2009-11-25');
INSERT INTO girl_group (name, debut)
VALUES ('포미닛', '2009-08-28');
```

아래 사진과 같이 테이블이 만들어 진다.

```
mysql> select * from song;
+-----+--------------------+-----------------------------------------------+
| _id | title              | lyrics                                        |
+-----+--------------------+-----------------------------------------------+
| 101 | Tell Me            | tell me tell me tetetete tel me               |
| 102 | Gee                | GEE GEE GEE GEE GEE BABY BABY                 |
| 103 | 미스터             | 이름이 뭐야 미스터                            |
| 104 | Abracadabra        | 이러다 미쳐 내가 여리여리                     |
| 105 | 8282               | Give me a call Baby baby                      |
| 106 | 기대해             | 기대해                                        |
| 107 | I Don't car        | 다른 여자들의 다리를                          |
| 108 | Bad Girl Good Girl | 앞에선 한 마디 말도                           |
| 109 | 피노키오           | 뉴예삐오                                      |
| 110 | 별빛달빛           | 너는 내 별빛 내 마음의 별빛                  |
| 111 | A                  | A 워오우 워오우워 우우우                      |
| 112 | 나혼자             | 나 혼자 밥을 먹고 나 혼자 영화 보고          |
| 113 | LUV                | 설레이나요                                    |
| 114 | 짧은치마           | 짧은 치마를 입고 내가 길을 걸으면            |
| 115 | 위아래             | 위 아래 위위 아래                             |
| 116 | Dumb Dumb          | 너 땜에 하루종일                              |
+-----+--------------------+-----------------------------------------------+
16 rows in set (0.00 sec)
```

```
mysql> select * from girl_group;
+-----+--------------------------+------------+-------------+
| _id | name                     | debut      | hit_song_id |
+-----+--------------------------+------------+-------------+
|   1 | 원더걸스                 | 2007-09-12 |         101 |
|   2 | 소녀시대                 | 2009-06-03 |         102 |
|   3 | 카라                     | 2009-07-30 |         103 |
|   4 | 브라운아이드걸스         | 2008-01-17 |         104 |
|   5 | 다비치                   | 2009-02-27 |         105 |
|   6 | 2NE1                     | 2009-07-08 |         107 |
|   7 | f(x)                     | 2011-04-20 |         109 |
|   8 | 시크릿                   | 2011-01-06 |         110 |
|   9 | 레인보우                 | 2010-08-12 |         111 |
|  10 | 에프터 스쿨              | 2009-11-25 |        NULL |
|  11 | 포미닛                   | 2009-08-28 |        NULL |
+-----+--------------------------+------------+-------------+
11 rows in set (0.00 sec)
```

1. INNER 조인

INNER 조인은 우리가 생각하는 일반적인 용도에 사용한다.

```
mysql> SELECT gg._id, gg.name, s.title
    -> FROM girl_group AS gg
    -> JOIN song AS s
    -> ON s._id = gg.hit_song_id;
+-----+--------------------------+--------------+
| _id | name                     | title        |
+-----+--------------------------+--------------+
|   1 | 원더걸스                 | Tell Me      |
|   2 | 소녀시대                 | Gee          |
|   3 | 카라                     | 미스터       |
|   4 | 브라운아이드걸스         | Abracadabra  |
|   5 | 다비치                   | 8282         |
|   6 | 2NE1                     | I Don't car  |
|   7 | f(x)                     | 피노키오     |
|   8 | 시크릿                   | 별빛달빛     |
|   9 | 레인보우                 | A            |
+-----+--------------------------+--------------+
9 rows in set (0.00 sec)
```

```
SELECT gg._id, gg.name, s.title
FROM girl_group AS gg
JOIN song AS s
ON s._id = gg.hit_song_id;
```

*** ON 대신 WHERE를 쓸 수 있다.**

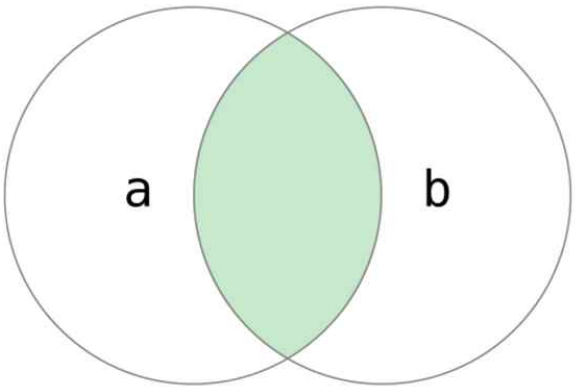
INNER 조인은 MySQL에서는 간략히 JOIN으로 나타낸다. 일반적으로 사용하는 JOIN 이다.

핵심은 JOIN뒤에 ON인데, 두 테이블이 결합하는 조건을 나타낸다.

두 테이블 모두 노래의 _id를 가지고 있으며, 서로 모두 포함하는 레코드를 합쳐서 표현한다.

'위 아래' 라는 곡의 id 는 115인데 이 값은 girl_group 테이블에 존재하지 않기 때문에 나오지 않는다.

집합으로 표현하자면 아래와 같다.



2. LEFT OUTER, RIGHT OUTER 조인

히트 곡이 저장되어 있지 않은 에프터 스쿨, 포미닛의 경우, 곡이 표시 되지 않더라도 보이고 싶을 때는 하나의 테이블 기준으로 합치는 조인을 사용할 수 있다.

```
mysql> SELECT gg._id, gg.name, s.title
    -> FROM girl_group AS gg
    -> LEFT JOIN song AS s
    -> ON s._id = gg.hit_song_id;
+-----+--------------------------+--------------+
| _id | name                     | title        |
+-----+--------------------------+--------------+
|   1 | 원더걸스                 | Tell Me      |
|   2 | 소녀시대                 | Gee          |
|   3 | 카라                     | 미스터       |
|   4 | 브라운아이드걸스         | Abracadabra  |
|   5 | 다비치                   | 8282         |
|   6 | 2NE1                     | I Don't car  |
|   7 | f(x)                     | 피노키오     |
|   8 | 시크릿                   | 별빛달빛     |
|   9 | 레인보우                 | A            |
|  10 | 에프터 스쿨              | NULL         |
|  11 | 포미닛                   | NULL         |
+-----+--------------------------+--------------+
11 rows in set (0.00 sec)
```

위 사진과 같이 일치 하지 않는 값을 가지고 있더라도 표시할 수 있게 된다.

```
SELECT gg._id, gg.name, s.title
FROM girl_group AS gg
LEFT OUTER JOIN song AS s
ON s._id = gg.hit_song_id;

SELECT s._id, s.title, gg.name
FROM girl_group AS gg
RIGHT OUTER JOIN song AS s
ON s._id = gg.hit_song_id;
```

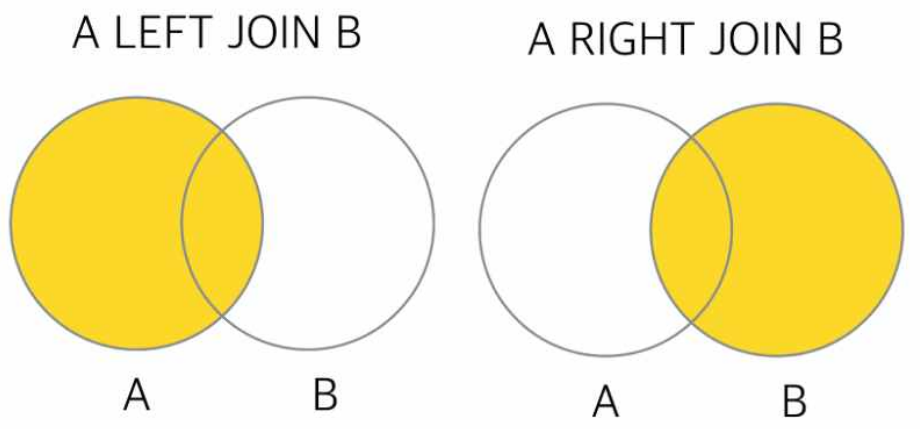
두 번째 쿼리는 song table을 기준으로 조인을 하였다.

```
mysql> SELECT s._id, s.title, gg.name
    -> FROM girl_group AS gg
    -> RIGHT JOIN song AS s
    -> ON s._id = gg.hit_song_id;
+-----+--------------------+--------------------------+
| _id | title              | name                     |
+-----+--------------------+--------------------------+
| 101 | Tell Me            | 원더걸스                 |
| 102 | Gee                | 소녀시대                 |
| 103 | 미스터             | 카라                     |
| 104 | Abracadabra        | 브라운아이드걸스         |
| 105 | 8282               | 다비치                   |
| 107 | I Don't car        | 2NE1                     |
| 109 | 피노키오           | f(x)                     |
| 110 | 별빛달빛           | 시크릿                   |
| 111 | A                  | 레인보우                 |
| 106 | 기대해             | NULL                     |
| 108 | Bad Girl Good Girl | NULL                     |
| 112 | 나혼자             | NULL                     |
| 113 | LUV                | NULL                     |
| 114 | 짧은치마           | NULL                     |
| 115 | 위아래             | NULL                     |
| 116 | Dumb Dumb          | NULL                     |
+-----+--------------------+--------------------------+
16 rows in set (0.00 sec)
```

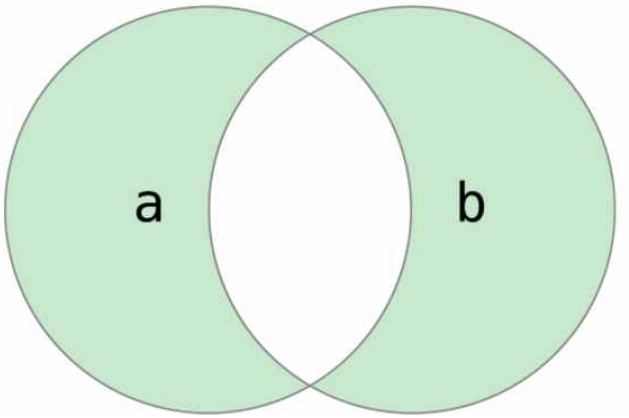
*** A LEFT JOIN B 와 B RIGHT JOIN A는 완전히 같은 식이다.**

*** LEFT OUTER 대신 LEFT, RIGHT OUTER 대신 RIGHT만 입력해도 같은 기능을 수행한다.**

LEFT 조인 과 RIGHT 조인을 집합으로 표현하면



MySQL에서는 OUTER JOIN을 지원하지 않지만, 유사한 처리를 할 수 있다. 아래는 OUTER JOIN의 범위다.




```
SELECT s._id, s.title, gg.name
FROM girl_group AS gg
JOIN song AS s
ON s._id <> gg.hit_song_id;
```

어떤 결과가 나올지 예상해보고 직접 해보면 좋겠다.

3. 카티전 조인 (CROSS JOIN)

집합에서 집합 곱의 개념이다.

```
A= {a, b, c, d} , B = {1, 2, 3} 일 때
A CROSS JOIN B 는

(a,1), (a, 2), (a,3), (b,1), (b,2), (b,3), (c, 1), (c,2), (c,3), (d, 1), (d, 2), (d,3)
```

와 같이 결과가 나타난다. 결과의 계수는 n(A) * n(B) = 4 * 3 = 12 이다.

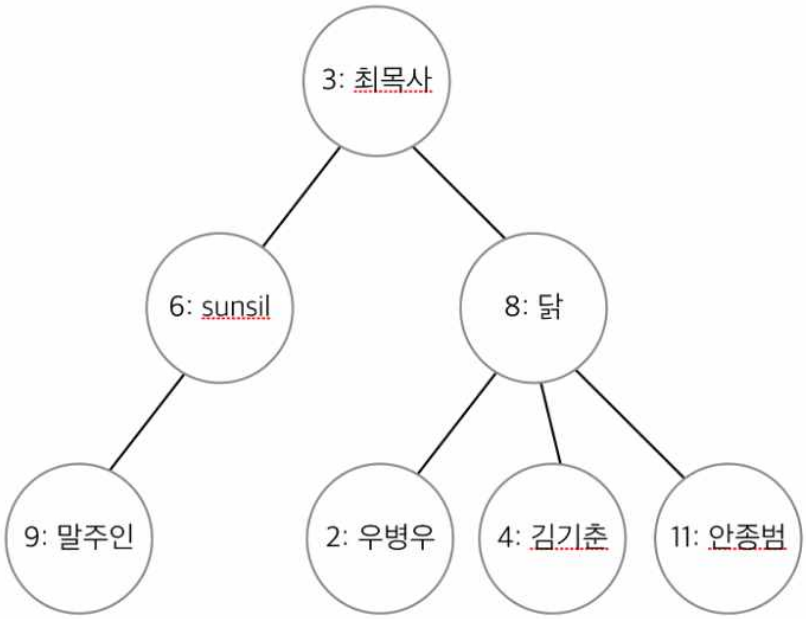
```
SELECT s._id, s.title, gg.name
FROM girl_group AS gg
CROSS JOIN song AS s;

SELECT s._id, s.title, gg.name
FROM girl_group AS gg, song AS s;
```

두 쿼리의 결과는 같다.
카티전 조인은 너무 많은 레코드를 생성할 위험이 있기 때문에, 많이 사용하지는 않는다.

*** NATURAL JOIN은 두 테이블에 칼럼명이 같은 것을 기준으로 INNER JOIN을 한다. 그렇기 때문에 JOIN 뒤에 ON을 생략할 수 있다.**

4. 셀프 조인

만약 아래와 같은 조직도를 생각해보자. 트리이기 때문에 닭의 부모 노드는 최목사다. 우병우의 부모 노드는 닭이다.



이를 테이블로 만들면

```
mysql> SELECT * FROM chicken_gate;
+-----+-----------+------+
| _id | name      | boss |
+-----+-----------+------+
|   2 | 우병우    |    8 |
|   3 | 최목사    |    3 |
|   4 | 김기춘    |    8 |
|   6 | sunsil    |    3 |
|   8 | 닭        |    3 |
|   9 | 말주인    |    6 |
|  11 | 안종범    |    8 |
+-----+-----------+------+
7 rows in set (0.00 sec)
```

```
CREATE TABLE chicken_gate
(
  _id INT PRIMARY KEY,
```

```
  name VARCHAR(16) NOT NULL,
  boss INT NOT NULL
);

INSERT INTO chicken_gate VALUES (3, '최목사', 3);
INSERT INTO chicken_gate VALUES (6, 'sunsil', 3);
INSERT INTO chicken_gate VALUES (8, '닭', 3);
INSERT INTO chicken_gate VALUES (9, '말주인', 6);
INSERT INTO chicken_gate VALUES (2, '우병우', 8);
INSERT INTO chicken_gate VALUES (4, '김기춘', 8);
INSERT INTO chicken_gate VALUES (11, '안종범', 8);
```

닭의 차일드는 우병우, 김기춘, 안종범이다. (닭과 sunsil은 파트너 관계다)

우리가 하고 싶은 것은 보스 아이디를 통해 보스의 이름을 함께 나타내는 테이블이다. 아래 그림과 같이 만드는 것이 목표다.

```
mysql> SELECT c.name AS child, p.name AS parent
    -> FROM chicken_gate AS p
    -> JOIN chicken_gate AS c
    -> ON p._id = c.boss;
+-----------+-----------+
| child     | parent    |
+-----------+-----------+
| 우병우    | 닭        |
| 최목사    | 최목사    |
| 김기춘    | 닭        |
| sunsil    | 최목사    |
| 닭        | 최목사    |
| 말주인    | sunsil    |
| 안종범    | 닭        |
+-----------+-----------+
7 rows in set (0.00 sec)
```

```
SELECT c.name AS child, p.name AS parent
FROM chicken_gate AS p
```

```
JOIN chicken_gate AS c
ON p._id = c.boss;
```

1 : 각 각 테이블의 이름중 하나는 child로 다른 하나는 parent로 보이도록 한다.
2, 3 : chicken_gate 라는 하나의 테이블을 p, c 두 개 처럼 사용하여 JOIN 했다.

지금 까지 다양한 조인을 살펴봤다. 이를 응용하면 원하는 결과를 얻을 수 있는 쿼리를 작성할 수 있을 것이다.

5장 유니온(UNION), 서브 쿼리

저번 시간에는 JOIN을 배웠다. JOIN을 잘 활용하면 많은 일을 할 수 있다. 그러나 상황에 따라서는 UNION을 함께 써야 될 때가 있다.

그리고 쿼리 결과를 INSERT 또는 UPDATE 할 필요가 있을 때, JOIN으로는 할 수 없다. 이럴 때 서브 쿼리가 유용하게 사용된다

1. 유니온 (UNION)

지금까지 테이블의 데이터는 아래와 같다.

```
mysql> SELECT * FROM professor;
+-----+-----------+--------+-------------+
| _id | name      | belong | phone       |
+-----+-----------+--------+-------------+
|   1 | 유재석    | IDE    | 01112345678 |
|   2 | 황영조    | MSE    | 01121342443 |
|   3 | 케이멀    | ESE    | 01123424343 |
| 256 | 호날두    | IME    | 01134343222 |
| 257 | 리오넬    | IDE    | 01123432432 |
+-----+-----------+--------+-------------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM girl_group;
+-----+--------------------------+------------+-------------+
| _id | name                     | debut      | hit_song_id |
+-----+--------------------------+------------+-------------+
|   1 | 원더걸스                 | 2007-09-12 |         101 |
|   2 | 소녀시대                 | 2009-06-03 |         102 |
|   3 | 카라                     | 2009-07-30 |         103 |
|   4 | 브라운아이드걸스         | 2008-01-17 |         104 |
|   5 | 다비치                   | 2009-02-27 |         105 |
|   6 | 2NE1                     | 2009-07-08 |         107 |
|   7 | f(x)                     | 2011-04-20 |         109 |
|   8 | 시크릿                   | 2011-01-06 |         110 |
|   9 | 레인보우                 | 2010-08-12 |         111 |
|  10 | 에프터 스쿨              | 2009-11-25 |        NULL |
|  11 | 포미닛                   | 2009-08-28 |        NULL |
+-----+--------------------------+------------+-------------+
11 rows in set (0.00 sec)
```

두 테이블에서 이름만 꺼내서 아래와 같이 하나의 테이블로 나타내고 싶다.

```
mysql> SELECT name FROM professor
    -> UNION
    -> SELECT name FROM girl_group;
+--------------------------+
| name                     |
+--------------------------+
| 유재석                   |
| 황영조                   |
| 케이멀                   |
| 호날두                   |
| 리오넬                   |
| 원더걸스                 |
| 소녀시대                 |
| 카라                     |
| 브라운아이드걸스         |
| 다비치                   |
| 2NE1                     |
| f(x)                     |
| 시크릿                   |
| 레인보우                 |
| 에프터 스쿨              |
| 포미닛                   |
+--------------------------+
16 rows in set (0.00 sec)
```

```
SELECT name FROM professor
UNION
SELECT name FROM girl_group;
```

UNION은 몇개 라도 계속해서 연결할 수 있다.

※ 유니온의 규칙

1. 하나의 ORDER BY만 사용할 수 있다.
2. 각 SELECT의 열수, 표현식가 같아야 한다.
3. SELECT 문들 끼리 순서는 상관없다.
4. 유니온을 한 결과가 중복되면 하나만 나온다. (DEFAULT)
5. 열의 타입은 같거나 반환 가능한 형태여야 한다.
6. 중복값을 나타내고 싶다면 UNION ALL

2. 서브 쿼리

서브 쿼리는 JOIN으로 할 수 있는 기능과 유사한 기능을 제공한다.

이번 시간에는 서브 쿼리가 어떤 경우에 유용하게 사용할 수 있을지에 대해 알아보자.

우선 저번 시간에 사용했던 girl_group 테이블과 song 테이블을 사용한다.

```
SELECT *
FROM girl_group
WHERE hit_song_id = (SELECT _id
                     FROM song
                     WHERE lyrics LIKE '%Give%'
                     );
```

```
SELECT *
FROM girl_group
WHERE hit_song_id IN (SELECT _id
                      FROM song
                      WHERE lyrics LIKE '%e%'
                      );
```

```
SELECT gg.name, gg.debut, s.title
FROM girl_group AS gg
JOIN song AS s
ON gg.hit_song_id = s._id
WHERE s.lyrics LIKE '%e%';
```

첫 번째 쿼리 결과

_id	name	debut	hit_song_id
5	다비치	2009-02-27	105

3~6 : (SELECT 부터 뒷 부분이 서브 쿼리다. 결과는 hit_song_id과 동일한 값인지 비교하게 된다. 이 때 서브 쿼리의 결과는 한 개 보다 크다면 오류가 발생한다. 여러 값을 비교하고 싶다면 두 번째 쿼리를 사용하면 된다.

두 번째 쿼리 결과

_id	name	debut	hit_song_id
1	원더걸스	2007-09-12	101
2	소녀시대	2009-06-03	102
5	다비치	2009-02-27	105

세 번째 쿼리 결과

name	debut	title
원더걸스	2007-09-12	Tell Me
소녀시대	2009-06-03	Gee
다비치	2009-02-27	8282

데이터를 유사하게 가져오지만, title 열은 song 테이블에 있는 열이다.

서브 쿼리의 위치는 다양하다. 위와 같이 사용하면 쿼리 결과에 song의 열은 포함할 수 없다. 위 join과 같은 결과를 얻기 위해선 아래와 같이 서브쿼리를 사용하면 된다.

```
SELECT gg.name, gg.debut,
(SELECT title
FROM song
WHERE title IS NOT NULL
AND lyrics LIKE '%e%'
 AND gg.hit_song_id = _id
) AS hitsong
FROM girl_group AS gg;
```

지금 부터 서브 쿼리의 유용성에 대해 말해보겠다. 사실, JOIN이 서브 쿼리에 비해 속도 면에서는 더 좋은 것은 분명하다. 그러나 JOIN은 그 결과를 SELECT 하는 것만 가능하다는 단점이 있다. 그렇다, 서브 쿼리는 SELECT, INSERT, UPDATE, DELETE 모두 가능하다.

```
DELETE FROM girl_group
WHERE hit_song_id IN (SELECT _id
                      FROM song
                      WHERE lyrics LIKE '%e%'
                       );
```

위 쿼리를 싱행하면 가사에 'e' 문자를 포함 하는 노래를 히트곡으로 하는 걸그룹이 삭제가 된다.

```
mysql> DELETE FROM girl_group
    -> WHERE hit_song_id IN (SELECT _id
    ->                       FROM song
    ->                       WHERE lyrics LIKE '%e%'
    ->                        );
Query OK, 3 rows affected (0.01 sec)

mysql> select * from girl_group;
+-----+--------------------------+------------+-------------+
| _id | name                     | debut      | hit_song_id |
+-----+--------------------------+------------+-------------+
|   3 | 카라                     | 2009-07-30 |         103 |
|   4 | 브라운아이드걸스         | 2008-01-17 |         104 |
|   6 | 2NE1                     | 2009-07-08 |         107 |
|   7 | f(x)                     | 2011-04-20 |         109 |
|   8 | 시크릿                   | 2011-01-06 |         110 |
|   9 | 레인보우                 | 2010-08-12 |         111 |
|  10 | 에프터 스쿨              | 2009-11-25 |        NULL |
|  11 | 포미닛                   | 2009-08-28 |        NULL |
+-----+--------------------------+------------+-------------+
8 rows in set (0.00 sec)
```

이번 시간에는 유니온과 서브 쿼리에 대해 배웠다. 이제 실무에서 사용할 수 있는 다양한 쿼리를 작성할 수 있는 베이스를 갖췄다. ORDER BY, GROUP BY, 서브 쿼리, 조인, 내부 함수 등을 잘 적용하면 원하는 결과를 얻어 낼 수 있는 쿼리를 작성할 수 있을 것이라 믿는다.

6장 사용자 권한 주기 (GRANT)

1. 권한 기본 설정 (GRANT)

- 사용자 암호 변경

```
ALTER USER 'mini'@'%' identified with mysql_native_password by '1111';
```

- 사용자 추가 및 테이블에 대한 권한 설정

```
CREATE USER test_user IDENTIFIED BY 'test';

GRANT SELECT ON study_db.color TO test_user;
```

1 : test_user라는 사용자를 추가하며 비밀번호는 'test'이다.

3 : test_user는 study_db 데이터베이스의 color 테이블에 대해 SELECT 할 수 있는 권한을 가진다.

2. 특정 데이터베이스에 대한 권한 주기

```
GRANT SELECT, INSERT ON study_db.* TO a_user;
GRANT ALL ON study_db.* TO a_user;
GRANT SELECT ON *.* TO a_user;
```

1 : a_user 사용자는 study_db 데이터베이스의 모든 table에 대해서 SELECT, INSERT 권한을 가진다.

2 : 데이터베이스의 모든 테이블에 대해 모든 권한을 준다.

3 : a_user는 모든 데이터베이스의 모든 테이블에 대해 SELECT 권한을 가진다.

3. 여러가지 권한

역할 만들어 권한 부여하기

```
CREATE ROLE insert_manager;
GRANT INSERT, SELECT ON some_db.* TO insert_manager;
GRANT insert_manager TO b_user;
```

권한을 수여할 권한 부여하기

```
GRANT ALL ON study_db.* TO test_user WITH GRANT OPTION;
```

WITH GRANT OPTION을 주면 test_user 가 가진 권한을 남에게 줄 수 있음(현재는 root에서 test_user에게 권한을 주었음)

역할을 수여할 권한 부여하기
WITH GRANT OPTION과 비슷하게 역할 또한 수여 가능하다.

```
GRANT inser_manager TO c_user WITH ADMIN OPTION;
```

사용자 생성과 동시에 권한 부여하기

```
GRANT SELECT ON ab_db TO e_user IDENTIFIED BY 'abcd33232';
```

4. 권한 취소

```
REVOKE SELECT ON ab_db.* FROM c_user;
```

c_user로 부터 ab_db 데이터베이스의 모든 테이블에 대해 SELECT 할 권한을 제거한다.

```
REVOKE GRANT OPTION ON SELECT ON ab_db.* FROM c_user
```

c_user로 부터 권한을 수여할 권한만 없앤다.

```
REVOKE SELECT ON ab_db.* FROM a_user CASCADE;
REVOKE SELECT ON ab_db.* FROM a_user RESTRICT;
```

위 : a_user가 다른 사람에게 수여한 모든 권한을 없앤다.
아래 : a_user의 권만한 없앤다.

* 위 CASCADE 또는 RESTRICT 둘 다 없을 경우 DEFAULT가 CASCADE

지금은 간단하게 내용을 다뤘지만, 몇몇 개는 더 공부하여 하나의 주제로 잡고 다시 다루는 날을 기약해본다.

7장 부록 트랜잭션, 뷰, Date

1. 뷰 (VIEW)
2. CHECK
3. 트랜잭션
4. DATE

1. 뷰 (VIEW)

뷰에 대한 설명은 다음과 같다.

쿼리를 저장하여 결과를 여러번 반복해서 확인하는 방법. 흔히, 가상 테이블이라 한다. 실제로 동작하는 것은 내부적으로 서브 쿼리를 호출한다.

※ 뷰의 유용성

1. 테이블 구조가 변경되어도 뷰를 계속 사용가능
2. 어플리케이션 코드에 복잡한 쿼리가 존재하지 않게 할 수 있다.
3. 쿼리가 어플리케이션에 노출 되지 않음
4. SELECT 할 수 있는 데이터는 모두 뷰로 만들 수 있다.

```
mysql> CREATE VIEW girl_group_song
    -> AS
    -> SELECT gg._id, gg.name, s.title
    -> FROM girl_group AS gg
    -> JOIN song AS s
    -> ON s._id = gg.hit_song_id;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from girl_group_song;
+-----+--------------------------+--------------+
| _id | name                     | title        |
+-----+--------------------------+--------------+
|   3 | 카라                     | 미스터       |
|   4 | 브라운아이드걸스         | Abracadabra  |
|   6 | 2NE1                     | I Don't car  |
|   7 | f(x)                     | 피노키오     |
|   8 | 시크릿                   | 별빛달빛     |
|   9 | 레인보우                 | A            |
+-----+--------------------------+--------------+
6 rows in set (0.01 sec)
```

2. CHECK

```
CREATE VIEW girl_group_song
AS
SELECT gg._id, gg.name, s.title
FROM girl_group AS gg
JOIN song AS s
ON s._id = gg.hit_song_id;


SELECT * FROM girl_group_song;
```

만약, 성별 데이터를 'W', 또는 'M'일 것이라 예상을 하는데 다른 값이 들어간다면? 또는 다른 값을 넣는 것을 방지하고 싶다면??

```
CREATE TABLE piggy
(
   _id INT AUTO_INCREMENT PRIMARY KEY,
   gender CHAR(1) CHECK (gender IN ('M', 'W'))
);

INSERT INTO piggy VALUES(0 , 'T');
```

* 데이터 무결성 : 특정 데이터에는 우리가 예상할 수 있는, 의미가 있는 데이터 이외의 데이터가 들어가면 안된다.

* MySQL 에서는 안된다.

3. 트랜잭션(TRANSACTION)

트랜잭션은 데이터베이스를 안정적으로 사용하기 위해 고안된 내용이다.

지금은 대략적인 내용만 살펴보자.

- 트랜잭션 : 어떤 처리 과정에서 어떤 단계에서든 문제가 발생하면 초기 상태로 되돌려서, 전체가 실행되거나, 전체 중 어떤 것도 반영되지 않게 하는 것.

```
A : ATOMICITY (일부만 수행할 수 없음)
C : CONSISTENCY
I : ISOLATION (다른 트랜잭션이 실행 중에는 관련된 데이터에 접근 조차 못함)
D : DUABILITY (결과 그대로 저장되고 유지)
```

```
START TRANSACTION;
  # SQL 추적 시작
COMMIT;
  # 적용
ROLLBACK;
  모든 것을 되돌린다.
```

* MySQL에는 여러 엔진이 있는데 그 중 트랜잭션이 가능한 엔진은 BDB, InnoDB 다. (Default 는 InnoDB)

```
ALTER TABLE your_table TYPE = InnoDB; # 변경 가능
```

4. DATE (DATETIME, TIME)

자료형	format	예
DATE	YYYY-MM-DD	2016-11-23
DATETIME	YYYY-MM-DD HH:MM:SS	2016-11-23 11:12:53
TIMESTAMP	YYYYMMDDHHMMSS	20161123111253
TIME	HH:MM:SS	11:12:53

TIMESTAMP는 시스템 변수를 통해 시간을 받아 오고, DATETIME은 임의로 입력한다. DATETIME은 미래의 일정에 적합하고, TIMESTAMP는 어떤 처리 시간을 저장할 때 유용할 것으로 보인다.

```
CREATE TABLE date_table
(
  _id INT PRIMARY KEY AUTO_INCREMENT,
  d_date DATE
);

INSERT INTO date_table VALUES(0, '2016-11-22');
```

```
SELECT DATE_FORMAT(d_date, '%M %Y') FROM date_table;
```

아래와 같이 일정한 형태로 날짜 정보를 출력도 가능하다.

```
mysql> SELECT DATE_FORMAT(d_date, '%M %Y') FROM date_table;
+-----------------------------+
| DATE_FORMAT(d_date, '%M %Y') |
+-----------------------------+
| November 2016               |
+-----------------------------+
1 row in set (0.00 sec)
```