DML 요약

SQL의 분류

- DML (Data Manipulation Language, 데이터 조작 언어)
  - 데이터를 조작(선택, 삽입, 수정, 삭제)하는 데 사용되는 언어
  - DML 구문이 사용되는 대상은 **테이블의 행**
  - DML 사용하기 위해서는 **테이블이 정의되어 있어야 함**
  - SQL문 중 **SELECT, INSERT, UPDATE, DELETE**가 이 구문에 해당
  - 트랜잭션(Transaction)이 발생하는 SQL도 DML에 속함
    - 테이블의 데이터를 변경(입력/수정/삭제)할 때 실제 테이블에 완전히 적용하지 않고, **임시로 적용시키는 것**
    - **취소 가능**

SQL의 분류

- DDL (Data Definition Language, 데이터 정의 언어)
  - 데이터베이스, 테이블, 뷰, 인덱스 등의 데이터베이스 개체를 생성/삭제/변경하는 역할
  - **CREATE, DROP, ALTER** 자주 사용
  - **DDL은 트랜잭션 발생시키지 않음**
  - 되돌림(ROLLBACK)이나 완전적용(COMMIT) 사용 불가
  - 실행 즉시 MySQL에 적용

- DCL (Data Control Language, 데이터 제어 언어)
  - 사용자에게 어떤 권한을 부여하거나 빼앗을 때 주로 사용하는 구문
  - GRANT/REVOKE/DENY 구문

<SELECT... FROM>

- 원하는 데이터를 가져와 주는 기본적인 구문
- 가장 많이 사용되는 구문
- 데이터베이스 내 테이블에서 원하는 정보 추출하는 명령

```
SELECT select_expr
    [FROM table_references]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}]
```

```
SELECT 열 이름
FROM 테이블이름
WHERE 조건
```

SELECT와 FROM

- SELECT *
  - 선택된 DB가 employees 라면 다음 두 쿼리는 동일

```
SELECT * FROM employees.titles;
SELECT * FROM titles;
```

- SELECT 열 이름
  - 테이블에서 필요로 하는 열만 가져오기 가능

```
SELECT first_name FROM employees;
```

  - 여러 개의 열을 가져오고 싶을 때는 콤마로 구분

```
SELECT first_name, last_name, gender FROM employees;
```

  - 열 이름의 순서는 출력하고 싶은 순서대로 배열 가능

특정 조건의 데이터만 조회 - <SELECT … FROM … WHERE>

- 기본적인 WHERE절
  - 조회하는 결과에 특정한 조건을 줘서 원하는 데이터만 보고 싶을 때 사용
  - SELECT 필드이름 FROM 테이블이름 WHERE 조건식;
    - ex)

```
SELECT * FROM usertbl WHERE name = '김경호';
```

- 관계 연산자의 사용
  - OR 연산자 : '…했거나', '… 또는'
  - AND 연산자 : '…하고', '…면서', '… 그리고'
  - 조건 연산자(=, <, >, <=, >=, < >, != 등)와 관계 연산자(NOT, AND, OR 등)를 조합하여 데이터를 효율적으로 추출 가능
    - ex)

```
SELECT userID, Name FROM usertbl WHERE birthYear >= 1970 AND height >= 182;
```

특정 조건의 데이터만 조회 - <SELECT … FROM … WHERE>

◦ BETWEEN… AND와 IN( ) 그리고 LIKE

- 데이터가 숫자로 구성되어 있으며 연속적인 값 : **BETWEEN … AND 사용**
  - ex)

```
SELECT name,  height FROM usertbl WHERE height BETWEEN 180 AND 183;
```

- 이산적인(Discrete) 값의 조건 : **IN() 사용**
  - ex)

```
SELECT name,  addr FROM usertbl WHERE addr IN ('경남','전남','경북');
```

- 문자열의 내용 검색 : **LIKE 사용(문자뒤에 % - 무엇이든 허용, 한 글자와 매치 '_' 사용)**
  - ex)

```
SELECT name,  height FROM usertbl WHERE name LIKE '김%';
```

원하는 순서대로 정렬하여 출력 : ORDER BY

- ORDER BY절
  - 결과물에 대해 영향을 미치지는 않고 출력되는 순서를 조절하는 구문
  - **기본적으로 오름차순** (ASCENDING) 정렬
  - **내림차순**(DESCENDING)으로 정렬하려면 **열 이름 뒤에 DESC**
  - ORDER BY 구문을 혼합해 사용하는 구문도 가능
    - 키가 큰 순서로 정렬하되 만약 키가 같을 경우 이름 순으로 정렬

```
SELECT name,  height FROM usertbl ORDER BY height DESC, name ASC;
```

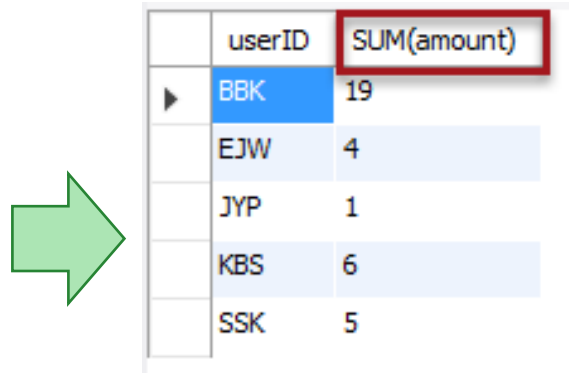
    - ASC(오름차순)는 디폴트 값이므로 생략 가능

- 중복된 것은 하나만 남기는 DISTINCT
  - 중복된 것을 골라서 세기 어려울 때 사용하는 구문
  - 테이블의 크기가 클수록 효율적
  - 중복된 것은 1개씩만 보여주면서 출력
- 출력하는 개수를 제한하는 LIMIT
  - 일부를 보기 위해 여러 건의 데이터를 출력하는 부담 줄임
  - 상위의 N개만 출력하는 'LIMIT N' 구문 사용
  - 개수의 문제보다는 MySQL의 부담을 많이 줄여주는 방법
- 테이블을 복사하는 CREATE TABLE ... SELECT
  - 테이블을 복사해서 사용할 경우 주로 사용
  - CREATE TABLE 새로운 테이블 (SELECT 복사할 열 FROM 기존테이블)
  - 지정한 일부 열만 복사하는 것도 가능
  - PK나 FK 같은 제약 조건은 복사되지 않음

GROUP BY 및 HAVING 그리고 집계 함수

- GROUP BY절
  - 그룹으로 묶어주는 역할
  - 집계 함수(Aggregate Function)와 함께 사용
    - 효율적인 데이터 그룹화 (Grouping)
    - ex) 각 사용자 별로 구매한 개수를 합쳐 출력

```
SELECT userID, SUM(amount) FROM buytbl GROUP BY userID;
```



userID	SUM(amount)
BBK	19
EJW	4
JYP	1
KBS	6
SSK	5

  - 읽기 좋게 하기 위해 별칭(Alias) AS 사용

```
SELECT userID AS '사용자 아이디', SUM(amount) AS '총 구매 개수'
   FROM buytbl GROUP BY userID;
```

사용자 아이디	총 구매 개수
BBK	19
EJW	4
JYP	1
KBS	6
SSK	5

GROUP BY 및 HAVING 그리고 집계 함수

◦ GROUP BY와 함께 자주 사용되는 집계 함수

함수명	설명
AVG( )	평균을 구한다.
MIN( )	최소값을 구한다.
MAX( )	최대값을 구한다.
COUNT( )	행의 개수를 센다.
COUNT(DISTINCT)	행의 개수를 센다(중복은 1개만 인정).
STDEV( )	표준편차를 구한다.
VAR_SAMP( )	분산을 구한다.

[표 6-1] GROUP BY와 함께 사용되는 집계 함수

- ex) 전체 구매자가 구매한 물품의 개수 평균

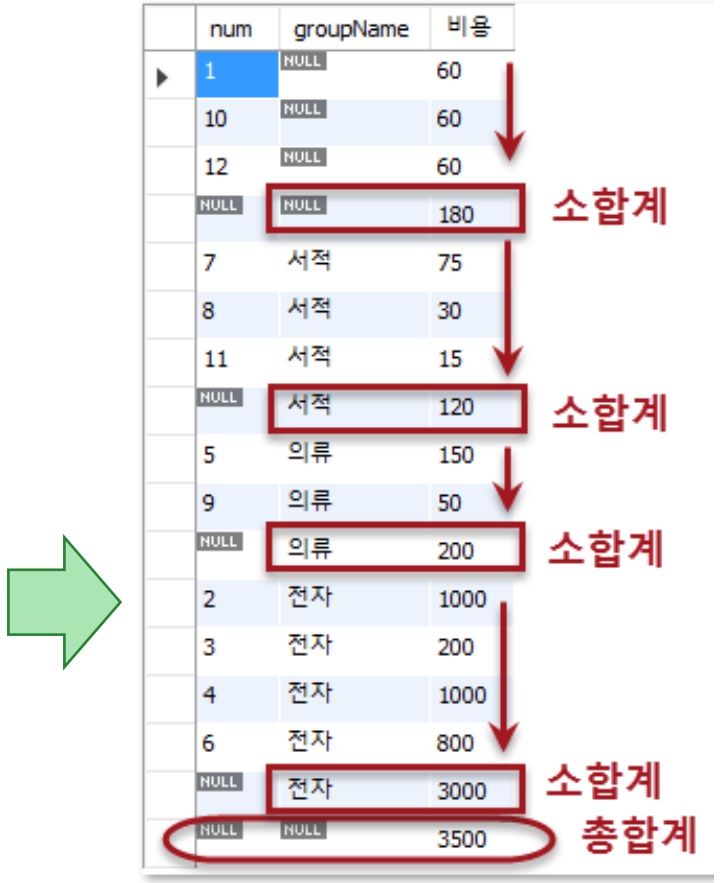
```
USE sqldb;
SELECT AVG(amount) AS '평균 구매 개수' FROM buytbl ;
```

	평균 구매 개수
▶	2.9167

GROUP BY 및 HAVING 그리고 집계 함수

- Having절
  - WHERE와 비슷한 개념으로 조건 제한하는 것이지만, 집계 함수에 대해서 조건을 제한하는 것
  - HAVING절은 꼭 GROUP BY절 다음에 나와야 함(순서 바뀌면 안됨)
- ROLLUP
  - 총합 또는 중간 합계가 필요할 경우 사용
  - GROUP BY절과 함께 WITH ROLLUP문 사용
    - ex) 분류(groupName) 별로 합계 및 그 총합 구하기

```
SELECT num, groupName, SUM(price * amount) AS '비용'
   FROM buytbl
   GROUP BY  groupName, num
   WITH ROLLUP;
```



조인(Join)

- 조인
  - 두 개 이상의 테이블을 서로 묶어서 하나의 결과 집합으로 만들어 내는 것
  - 종류 : INNER JOIN, OUTER JOIN, CROSS JOIN, SELF JOIN

- 데이터베이스의 테이블
  - 중복과 공간 낭비를 피하고 데이터의 무결성을 위해서 여러 개의 테이블로 분리하여 저장
  - 분리된 테이블들은 서로 관계(Relation)를 가짐
  - 1대 다 관계 보편적

INNER JOIN(내부 조인)

- 조인 중에서 가장 많이 사용되는 조인
  - 대개의 업무에서 조인은 INNER JOIN 사용
  - 일반적으로 JOIN이라고 얘기하는 것이 이 INNER JOIN 지칭
  - 사용 형식

```
SELECT <열 목록>
FROM <첫 번째 테이블>
       INNER JOIN <두 번째 테이블>
       ON <조인될 조건>
[WHERE 검색조건]
```

  - JOIN만 써도 INNER JOIN으로 인식함

INNER JOIN(내부 조인)

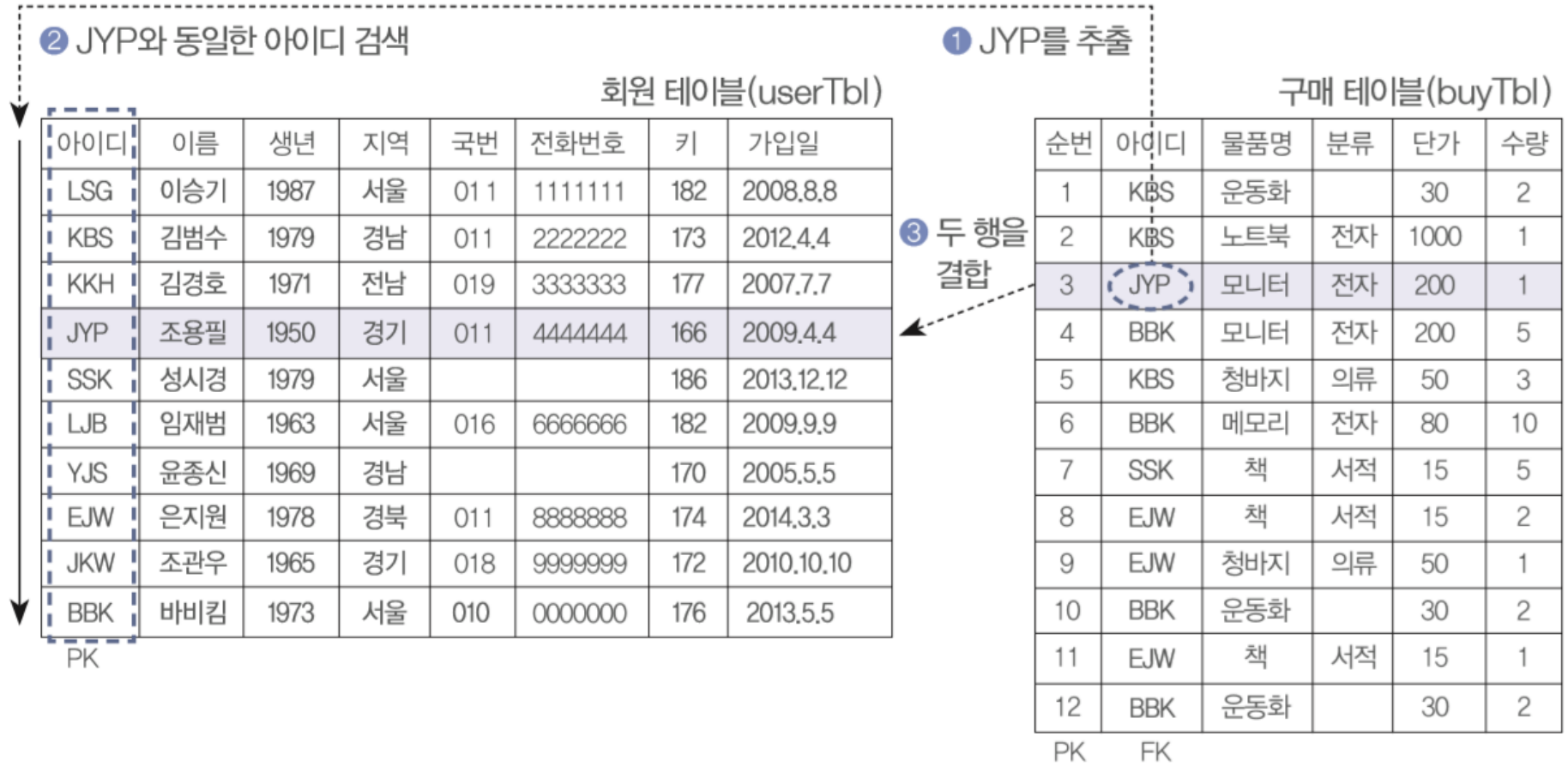
◦ 조인 중에서 가장 많이 사용되는 조인

```
USE sqldb;
SELECT *
   FROM buytbl
     INNER JOIN usertbl
        ON buytbl.userID = usertbl.userID
   WHERE buytbl.userID = 'JYP';
```

num	userID	prodName	groupName	price	amount	userID	name	birthYear	addr	mobile1	mobile2	height	mDate
3	JYP	모니터	전자	400	1	JYP	조용필	1950	경기	011	4444444	166	2009-04-04

INNER JOIN(내부 조인)

◦ 조인 중에서 가장 많이 사용되는 조인



아이디	이름	생년	지역	국번	전화번호	키	가입일
LSG	이승기	1987	서울	011	1111111	182	2008.8.8
KBS	김범수	1979	경남	011	2222222	173	2012.4.4
KKH	김경호	1971	전남	019	3333333	177	2007.7.7
JYP	조용필	1950	경기	011	4444444	166	2009.4.4
SSK	성시경	1979	서울			186	2013.12.12
LJB	임재범	1963	서울	016	6666666	182	2009.9.9
YJS	윤종신	1969	경남			170	2005.5.5
EJW	은지원	1978	경북	011	8888888	174	2014.3.3
JKW	조관우	1965	경기	018	9999999	172	2010.10.10
BBK	바비킴	1973	서울	010	0000000	176	2013.5.5

순번	아이디	물품명	분류	단가	수량
1	KBS	운동화		30	2
2	KBS	노트북	전자	1000	1
3	JYP	모니터	전자	200	1
4	BBK	모니터	전자	200	5
5	KBS	청바지	의류	50	3
6	BBK	메모리	전자	80	10
7	SSK	책	서적	15	5
8	EJW	책	서적	15	2
9	EJW	청바지	의류	50	1
10	BBK	운동화		30	2
11	EJW	책	서적	15	1
12	BBK	운동화		30	2

[그림 7-28] INNER JOIN의 작동

OUTER JOIN(외부 조인)

- 조인의 조건에 만족되지 않는 행까지도 포함시키는 것

```
SELECT <열 목록>
FROM <첫 번째 테이블(LEFT 테이블)>
 <LEFT ¦ RIGHT ¦ FULL> OUTER JOIN <두 번째 테이블(RIGHT 테이블)>
      ON <조인될 조건>
[WHERE 검색조건] ;
```

- LEFT OUTER JOIN
  - 왼쪽 테이블의 것은 모두 출력되어야 한다로 이해
  - 줄여서 LEFT JOIN으로 쓸수있음
- RIGHT OUTER JOIN
  - 오른쪽 테이블의 것은 모두 출력되어야 한다로 이해

OUTER JOIN(외부 조인)

◦ LEFT OUTER JOIN

```
USE sqldb;
SELECT U.userID, U.name, B.prodName, U.addr, CONCAT(U.mobile1, U.mobile2)  AS '연락처'
   FROM usertbl U
      LEFT OUTER JOIN buytbl B
         ON U.userID = B.userID
   ORDER BY U.userID;
```

userID	name	prodName	addr	연락처
BBK	바비킴	모니터	서울	0100000000
BBK	바비킴	메모리	서울	0100000000
BBK	바비킴	운동화	서울	0100000000
BBK	바비킴	운동화	서울	0100000000
EJW	은지원	책	경북	0118888888
EJW	은지원	청바지	경북	0118888888
EJW	은지원	책	경북	0118888888
JKW	조관우	NULL	경기	0189999999
JYP	조용필	모니터	경기	0114444444
KBS	김범수	운동화	경남	0112222222
KBS	김범수	노트북	경남	0112222222
KBS	김범수	청바지	경남	0112222222
KKH	김경호	NULL	전남	0193333333
LJB	임재범	NULL	서울	0166666666
LSG	이승기	NULL	서울	0111111111
SSK	성시경	책	서울	NULL
YJS	윤종신	NULL	경남	NULL

CROSS JOIN(상호 조인)

- 한쪽 테이블의 모든 행들과 다른 쪽 테이블의 모든 행을 조인시키는 기능
- CROSS JOIN의 결과 개수 = **두 테이블 개수를 곱한 개수**

회원 테이블(userTbl)

아이디	이름	생년	지역	국번	전화번호	키	가입일
LSG	이승기	1987	서울	011	1111111	182	2008.8.8
KBS	김범수	1979	경남	011	2222222	173	2012.4.4
KKH	김경호	1971	전남	019	3333333	177	2007.7.7
JYP	조용필	1950	경기	011	4444444	166	2009.4.4
SSK	성시경	1979	서울			186	2013.12.12
LJB	임재범	1963	서울	016	6666666	182	2009.9.9
YJS	윤종신	1969	경남			170	2005.5.5
EJW	은지원	1978	경북	011	8888888	174	2014.3.3
JKW	조관우	1965	경기	018	9999999	172	2010.10.10
BBK	바비킴	1973	서울	010	0000000	176	2013.5.5

PK

구매 테이블(buyTbl)

순번	아이디	물품명	분류	단가	수량
1	KBS	운동화		30	2
2	KBS	노트북	전자	1000	1
3	JYP	모니터	전자	200	1
4	BBK	모니터	전자	200	5
5	KBS	청바지	의류	50	3
6	BBK	메모리	전자	80	10
7	SSK	책	서적	15	5
8	EJW	책	서적	15	2
9	EJW	청바지	의류	50	1
10	BBK	운동화		30	2
11	EJW	책	서적	15	1
12	BBK	운동화		30	2

PK FK

[그림 7-43] CROSS JOIN(상호 조인) 방식

CROSS JOIN(상호 조인)

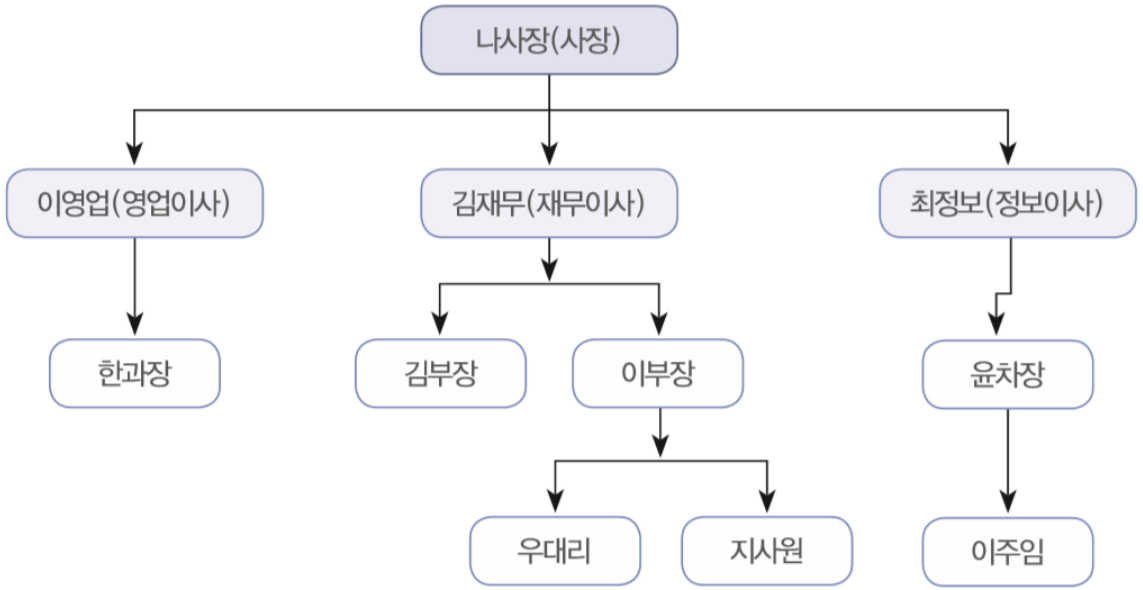
- 테스트로 사용할 많은 용량의 데이터를 생성할 때 주로 사용
- ON 구문을 사용할 수 없음
- 대량의 데이터를 생성하면 시스템이 다운되거나 디스크 용량이 모두 찰 수 있어 COUNT(*) 함수로 개수만 카운트

```
USE employees;
SELECT  COUNT(*) AS '데이터개수'
   FROM employees
     CROSS JOIN titles;
```

	데이터개수
▶	133003039392

SELF JOIN(자체 조인)

- 자기 자신과 자기 자신이 조인한다는 의미
  - 대표적인 예
    - 조직도와 관련된 테이블



[그림 7-45] 간단한 조직도 예

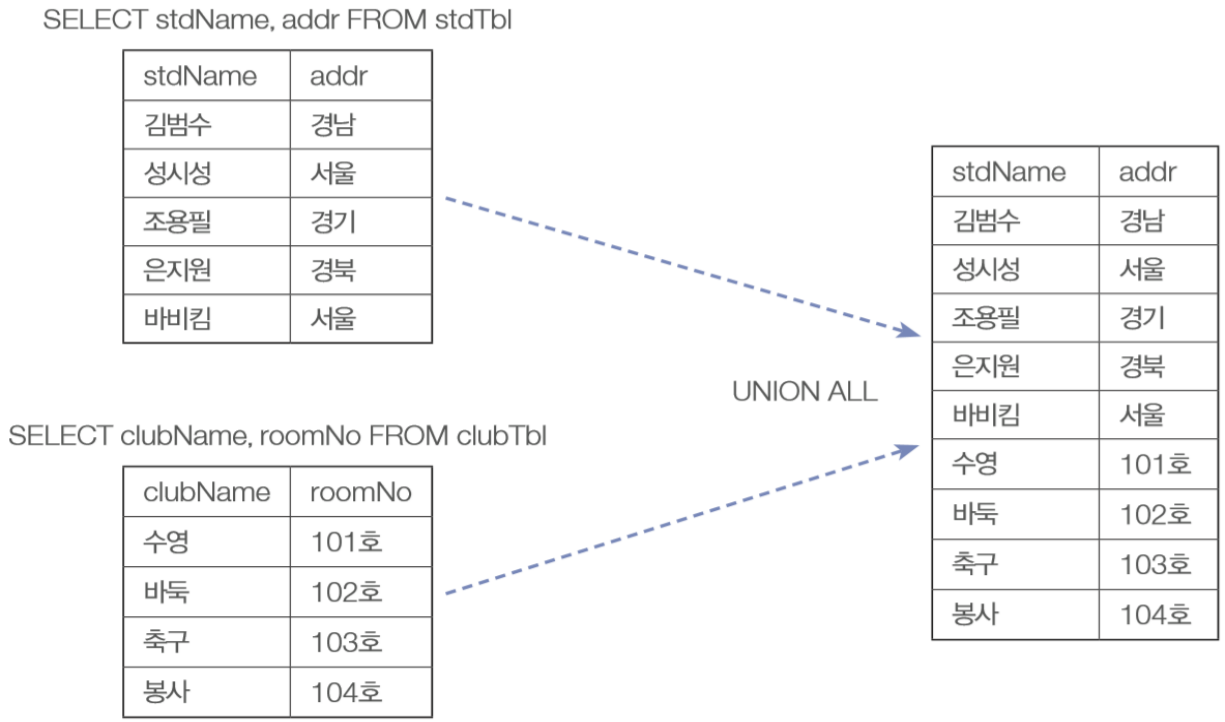
직원 이름(EMP) - 기본 키	상관 이름(MANAGER)	구내 번호
나사장	없음 (NULL)	0000
김재무	나사장	2222
김부장	김재무	2222-1
이부장	김재무	2222-2
우대리	이부장	2222-2-1
지사원	이부장	2222-2-2
이영업	나사장	1111
한과장	이영업	1111-1
최정보	나사장	3333
윤차장	최정보	3333-1
이주임	윤차장	3333-1-1

[표 7-5] 조직도 테이블

UNION / UNION ALL / NOT IN / IN

◦ 두 쿼리의 결과를 행으로 합치는 것

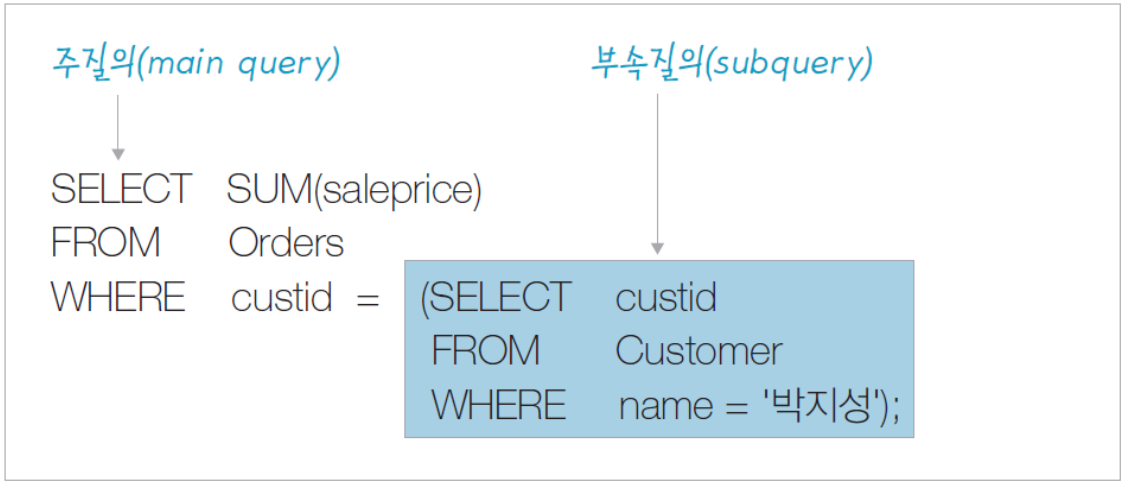
```
SELECT 문장1
   UNION [ALL]
SELECT 문장2
```



[그림 7-47] UNION의 결합과정

부속질의

- 하나의 SQL 문 안에 다른 SQL 문이 중첩된(nested) 질의
- 다른 테이블에서 가져온 데이터로 현재 테이블에 있는 정보를 찾거나 가공할 때 사용
- 보통 데이터가 대량일 때 데이터를 모두 합쳐서 연산하는 조인보다 필요한 데이터만 찾아서 공급해주는 부속질의가 성능이 더 좋음
- 주질의(main query, 외부질의)와 부속질의(sub query, 내부질의)로 구성됨

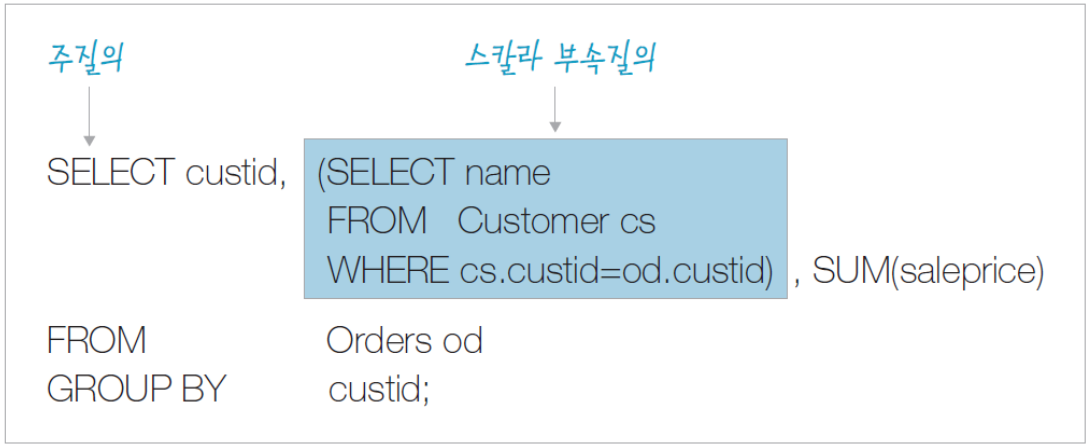


부속질의

명칭	위치	영문 및 동의어	설명
스칼라 부속질의	SELECT 절	scalar subquery	SELECT 절에서 사용되며 단일 값을 반환하기 때문에 스칼라 부속질의라고 함.
인라인 뷰	FROM 절	inline view, table subquery	FROM 절에서 결과를 뷰(view) 형태로 반환하기 때문에 인라인 뷰라고 함.
중첩질의	WHERE 절	nested subquery, predicate subquery	WHERE 절에 술어와 같이 사용되며 결과를 한정시키기 위해 사용됨. 상관 혹은 비상관 형태.

스칼라 부속질의

- SELECT 절에서 사용되는 부속질의로, 부속질의의 결과 값을 단일 행, 단일 열의 스칼라 값으로 반환함
- 원칙적으로 스칼라 값이 들어갈 수 있는 모든 곳에 사용 가능하며, 일반적으로 SELECT 문과 UPDATE SET 절에 사용됨
- 주질의와 부속질의와의 관계는 상관/비상관 모두 가능함



스칼라 부속질의

마당서점의 고객별 판매액을 보이시오(고객이름과 고객별 판매액을 출력).

```
SELECT      ( SELECT   name
              FROM    Customer cs
              WHERE   cs.custid=od.custid ) 'name', SUM(saleprice) 'total'
FROM         Orders od
GROUP BY     od.custid;
```

name	total
박지성	39000
김연아	15000
장미란	31000
추신수	33000

스칼라 부속질의 – SELECT 부속질의

```
SELECT      custid,
            SUM(saleprice)  'total'
FROM        Orders od
GROUP  BY custid ;
```

custid	total
1	39000
2	15000
3	31000
4	33000

```
SELECT      name
FROM        Customer cs
WHERE       cs.custid = od.custid
```

Custid 1의 이름은?

```
SELECT      SELECT      name
            FROM        Customer cs
            WHERE       cs.custid = od.custid   name,
            SUM(saleprice) 'total'
FROM        Orders od
GROUP BY od.custid ;
```

name	total
박지성	39000
김연아	15000
장미란	31000
추신수	33000

스칼라 부속질의 – SELECT 부속질의

Orders 테이블에 각 주문에 맞는 도서이름을 입력하시오.

```
UPDATE   Orders
SET      bookname = ( SELECT bookname
                      FROM Book
                      WHERE Book.bookid=Orders.bookid );
```

orderid	custid	bookid	saleprice	orderdate	bname
1	1	1	6000	2014-07-01	축구의 역사
2	1	3	21000	2014-07-03	축구의 이해
3	2	5	8000	2014-07-03	피겨 교본
4	3	6	6000	2014-07-04	역도 단계별기술
5	4	7	20000	2014-07-05	야구의 추억
6	1	2	12000	2014-07-07	축구아는 여자
7	4	8	13000	2014-07-07	야구를 부탁해
8	3	10	12000	2014-07-08	Olympic Champions
9	2	10	7000	2014-07-09	Olympic Champions
10	3	8	13000	2014-07-10	야구를 부탁해

인라인 뷰- FROM 부속질의

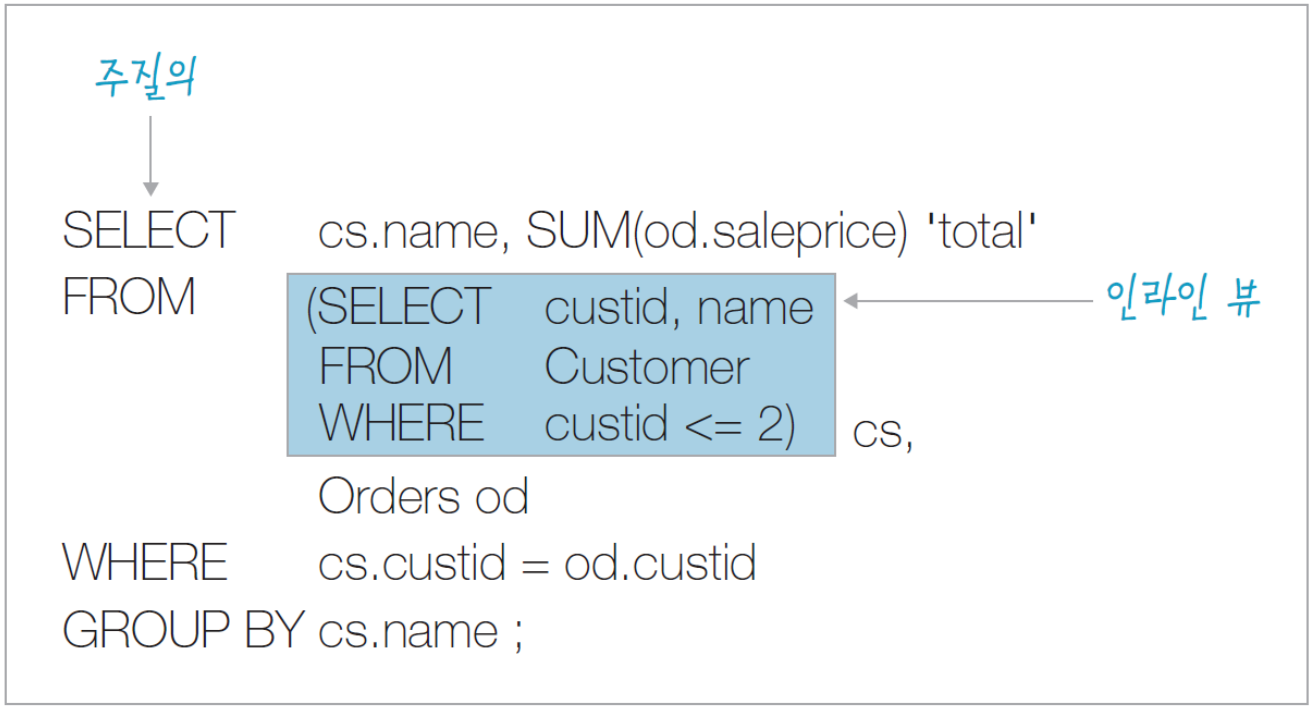
- FROM 절에서 사용되는 부속질의
- 테이블 이름 대신 인라인 뷰 부속질의를 사용하면 보통의 테이블과 같은 형태로 사용할 수 있음
- 부속질의 결과 반환되는 데이터는 다중 행, 다중 열이어도 상관없음
- 다만 가상의 테이블인 뷰 형태로 제공되어 상관 부속질의로 사용될 수는 없음

고객번호가 2 이하인 고객의 판매액을 보이시오(고객이름과 고객별 판매액 출력).

```
SELECT    cs.name, SUM(od.saleprice) 'total'
FROM      (SELECT  custid, name
           FROM   Customer
           WHERE  custid <= 2) cs,
           Orders   od
WHERE     cs.custid=od.custid
GROUP BY cs.name;
```

name	total
박지성	39000
김연아	15000

인라인 뷰- FROM 부속질의



중첩질의 – WHERE 부속질의

- **중첩질의(nested subquery) : WHERE 절에서 사용되는 부속질의**
- **WHERE 절은 보통 데이터를 선택하는 조건 혹은 술어(predicate)와 같이 사용됨 → 중첩질의를 술어 부속질의(predicate subquery)라고도 함**

술어	연산자	반환 행	반환 열	상관
비교	=, >, <, >=, <=, <>	단일	단일	가능
집합	IN, NOT IN	다중	단일	가능
한정(quantified)	ALL, SOME(ANY)	다중	단일	가능
존재	EXISTS, NOT EXISTS	다중	다중	필수

중첩질의 – WHERE 부속질의

❖ 비교 연산자

부속질의가 반드시 단일 행, 단일 열을 반환해야 하며, 아닐 경우 질의를 처리할 수 없음

평균 주문금액 이하의 주문에 대해서 주문번호와 금액을 보이시오.

```
SELECT    orderid, saleprice
FROM      Orders
WHERE     saleprice <= (SELECT AVG(saleprice)
                        FROM Orders);
```

orderid	saleprice
1	6000
3	8000
4	6000
9	7000

각 고객의 평균 주문금액보다 큰 금액의 주문 내역에 대해서 주문번호, 고객번호, 금액을 보이시오.

```
SELECT    orderid, custid, saleprice
FROM      Orders od
WHERE     saleprice > (SELECT AVG(saleprice)
                       FROM Orders so
                       WHERE od.custid=so.custid);
```

orderid	custid	saleprice
2	1	21000
3	2	8000
5	4	20000
8	3	12000
10	3	13000

중첩질의 – WHERE 부속질의

❖ IN, NOT IN

- IN 연산자는 주질의 속성 값이 부속질의에서 제공한 결과 집합에 있는지 확인하는 역할을 함
- IN 연산자는 부속질의의 결과 다중 행을 가질 수 있음
- 주질의는 WHERE 절에 사용되는 속성 값을 부속질의의 결과 집합과 비교해 하나라도 있으면 참이 됨
- NOT IN은 이와 반대로 값이 존재하지 않으면 참이 됨

대한민국에 거주하는 고객에게 판매한 도서의 총판매액을 구하시오.

```
SELECT    SUM(saleprice) 'total'
FROM      Orders
WHERE     custid IN (SELECT   custid
                     FROM    Customer
                     WHERE   address LIKE '%대한민국%');
```

total
46000

중첩질의 – WHERE 부속질의

❖ ALL, SOME(ANY)

- ALL은 모두, SOME(ANY)은 어떠한(최소한 하나라도)이라는 의미

3번 고객이 주문한 도서의 최고 금액보다 더 비싼 도서를 구입한 주문의 주문번호와 금액을 보이시오.

```
SELECT    orderid, saleprice
FROM      Orders
WHERE     saleprice > ALL (SELECT  saleprice
                           FROM   Orders
                           WHERE custid='3');
```

orderid	saleprice
2	21000
5	20000

중첩질의 – WHERE 부속질의

❖ EXISTS, NOT EXISTS

- 데이터의 존재 유무를 확인하는 연산자
- 주질의에서 부속질의로 제공된 속성의 값을 가지고 부속질의에 조건을 만족하여 값이 존재하면 참이 되고, 주질의는 해당 행의 데이터를 출력함
- NOT EXIST의 경우 이와 반대로 동작함
- 구문 구조

EXISTS 연산자로 대한민국에 거주하는 고객에게 판매한 도서의 총 판매액을 구하시오.

```
SELECT    SUM(saleprice) 'total'
FROM      Orders od
WHERE     EXISTS (SELECT   *
                  FROM    Customer cs
                  WHERE   address LIKE '%대한민국%' AND cs.custid=od.custid);
```

total
46000

데이터의 삽입 : INSERT

- INSERT문의 기본

```
INSERT [INTO] 테이블[(열1, 열2, …)] VALUES (값1, 값2 …)
```

  - 테이블 이름 다음에 나오는 열 생략 가능
    - 생략할 경우에 VALUES 다음에 나오는 값들의 순서 및 개수가 테이블이 정의된 열 순서 및 개수와 동일해야 함

- 자동으로 증가하는 AUTO_INCREMENT
  - INSERT에서는 해당 열이 없다고 생각하고 입력
    - INSERT문에서 NULL 값 지정하면 자동으로 값 입력
  - 1부터 증가하는 값 자동 입력
  - 적용할 열이 PRIMARY KEY 또는 UNIQUE일 때만 사용가능
  - 데이터 형은 숫자 형식만 사용 가능

데이터의 삽입 : INSERT

◦ 대량의 샘플 데이터 생성

- INSERT INTO ... SELECT 구문 사용

```
형식:
INSERT INTO 테이블이름 (열 이름1, 열 이름2, …)
   SELECT문  ;
```

- 다른 테이블의 데이터를 가져와 대량으로 입력하는 효과
- SELECT문의 열의 개수 = INSERT 할 테이블의 열의 개수
- 테이블 정의 까지 생략 하려면 CREATE TABLE ... SELECT 구문을 사용

데이터의 수정 : UPDATE

◦ 기존에 입력되어 있는 값 변경하는 구문

```
UPDATE 테이블이름
    SET 열1=값1, 열2=값2 …
    WHERE 조건 ;
```

◦ **WHERE절 생략 가능하나 WHERE절 생략하면 테이블의 전체 행의 내용 변경됨**

- 실무에서 실수가 종종 일어남, **주의 필요**
- 원상태로 복구하기 복잡하며, 다시 되돌릴 수 없는 경우도 있음

데이터의 삭제 : DELETE FROM

- 행 단위로 데이터 삭제하는 구문

```
DELETE FROM 테이블이름 WHERE 조건;
```

- WHERE절 생략되면 전체 데이터를 삭제함
- 테이블을 삭제하는 경우의 속도 비교
  - DML문인 DELETE는 트랜잭션 로그 기록 작업 때문에 삭제 느림
  - DDL문인 DROP과 TRUNCATE문은 트랜잭션 없어 빠름
    - 테이블 자체가 필요 없을 경우에는 DROP 으로 삭제
    - 테이블의 구조는 남겨놓고 싶다면 TRUNCATE로 삭제하는 것이 효율적

MySQL에서 지원하는 데이터 형식의 종류

- Data Type으로 표현
  - 데이터 형식, 데이터형, 자료형, 데이터 타입등 다양하게 불림
- 데이터 형식에 대한 이해가 필요한 이유
  - SELECT문 더욱 잘 활용
  - 테이블의 생성 효율적으로 하기 위해 필요
- MySQL에서 데이터 형식의 종류는 30개 정도
  - **중요하고 자주 쓰는 형식에 대해 중점 학습**

MySQL에서 지원하는 데이터 형식의 종류

◦ 숫자 데이터 형식

데이터 형식	바이트 수	숫자 범위	설명
BIT(N)	N/8		1~64bit를 표현. b'0000' 형식으로 표현
TINYINT	1	-128~127	정수
★SMALLINT	2	-32,768~32,767	정수
MEDIUMINT	3	-8,388,608~8,388,607	정수
★INT INTEGER	4	약 -21억 ~ +21억	정수
★BIGINT	8	약 -900경 ~ +900경	정수
★FLOAT	4	-3.40E+38~-1.17E-38	소수점 아래 7자리까지 표현
DOUBLE REAL	8	-1.22E-308~1.79E+308	소수점 아래 15자리까지 표현
★DECIMAL(m,[d]) NUMERIC(m,[d])	5~17	$-10^{38}+1 \sim +10^{38}-1$	전체 자릿수(m)와 소수점 이하 자릿수(d)를 가진 숫자형 예) decimal(5,2)는 전체 자릿수를 5자리로 하되, 그 중 소수점 이하를 2자리로 하겠다는 의미

[표 7-1] 숫자 데이터 형식

MySQL에서 지원하는 데이터 형식의 종류

◦ 문자 데이터 형식

데이터 형식		바이트 수	설명
★CHAR(n)		1~255	고정길이 문자형. n을 1부터 255까지 지정. character의 약자 그냥 CHAR만 쓰면 CHAR(1)과 동일
★VARCHAR(n)		1~65535	가변길이 문자형. n을 사용하면 1부터 65535까지 지정. Variable character의 약자
BINARY(n)		1~255	고정길이의 이진 데이터 값
VARBINARY(n)		1~255	가변길이의 이진 데이터 값
TEXT 형식	TINYTEXT	1~255	255 크기의 TEXT 데이터 값
	TEXT	1~65535	N 크기의 TEXT 데이터 값
	MEDIUMTEXT	1~16777215	16777215 크기의 TEXT 데이터 값
	★LONGTEXT	1~4294967295	최대 4GB 크기의 TEXT 데이터 값
BLOB 형식	TINYBLOB	1~255	255 크기의 BLOB 데이터 값
	BLOB	1~65535	N 크기의 BLOB 데이터 값
	MEDIUMBLOB	1~16777215	16777215 크기의 BLOB 데이터 값
	★LONGBLOB	1~4294967295	최대 4GB 크기의 BLOB 데이터 값
ENUM(값들…)		1 또는 2	최대 65535개의 열거형 데이터 값
SET(값들…)		1, 2, 3, 4, 8	최대 64개의 서로 다른 데이터 값

[표 7-2] 문자 데이터 형식

MySQL에서 지원하는 데이터 형식의 종류

◦ 날짜와 시간 데이터 형식

데이터 형식	바이트 수	설명
★DATE	3	날짜는 1001-01-01 ~ 9999-12-31까지 저장되며 날짜 형식만 사용 'YYYY-MM-DD' 형식으로 사용됨.
TIME	3	-838:59:59.000000 ~ 838:59:59.000000까지 저장되며. 'HH:MM:SS' 형식으로 사용
★DATETIME	8	날짜는 1001-01-01 00:00:00 ~ 9999-12-31 23:59:59까지 저장되며 형식은 'YYYY-MM-DD HH:MM:SS' 형식으로 사용
TIMESTAMP	4	날짜는 1001-01-01 00:00:00 ~ 9999-12-31 23:59:59까지 저장되며 형식은 'YYYY-MM-DD HH:MM:SS' 형식으로 사용. time_zone 시스템 변수와 관련이 있으며 UTC 시간대 변환하여 저장
YEAR	1	1901 ~ 2155까지 저장. 'YYYY' 형식으로 사용

[표 7-3] 날짜와 시간 데이터 형식

	DATE
▶	2020-10-19

	TIME
▶	12:35:29

	DATETIME
▶	2020-10-19 12:35:29

MySQL에서 지원하는 데이터 형식의 종류

◦ 기타 데이터 형식

데이터 형식	바이트 수	설명
★GEOMETRY	N/A	공간 데이터 형식으로 선, 점 및 다각형 같은 공간 데이터 개체를 저장하고 조작
★JSON	8	JSON(JavaScript Object Notation) 문서를 저장

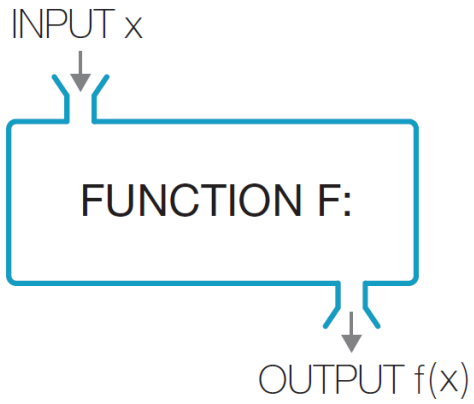
[표 7-4] 기타 데이터 형식

◦ LONGTEXT, LONGBLOB

- LOB(Large Object, 대량의 데이터)을 저장하기 위해 LONGTEXT, LONGBLOB 데이터 형식 지원
- 지원되는 데이터 크기는 약 4GB의 파일을 하나의 데이터로 저장 가능
- LONGTEXT
  - ex) 장편소설과 같은 큰 텍스트 파일
- LONGBLOG
  - ex) 동영상 파일과 같은 큰 바이너리 파일

SQL 내장 함수

- SQL에서는 함수의 개념을 사용
- 수학의 함수와 마찬가지로 특정 값이나 열의 값을 입력 받아 그 값을 계산하여 결과 값을 돌려줌



- SQL의 함수는 DBMS가 제공하는 내장 함수(built-in function),
  사용자가 필요에 따라 직접 만드는 사용자 정의 함수(user-defined function)로 나뉨

SQL 내장 함수

- SQL 내장 함수는 상수나 속성 이름을 입력 값으로 받아 단일 값을 결과로 반환함
- 모든 내장 함수는 최초에 선언될 때 유효한 입력 값을 받아야 함

구분		함수
단일행 함수	숫자 함수	ABS, CEIL, COS, EXP, FLOOR, LN, LOG, MOD, POWER, RAND, ROUND, SIGN, TRUNCATE
	문자 함수(문자 반환)	CHAR, CONCAT, LEFT, RIGHT, LOWER, UPPER, LPAD, RPAD, LTRIM, RTRIM, REPLACE, REVERSE, RIGHT, SUBSTR, TRIM
	문자 함수(숫자 반환)	ASCII, INSTR, LENGTH
	날짜·시간 함수	ADDDATE, CURRENT_DATE, DATE, DATEDIFF, DAYNAME, LAST_DAY, SYSDATE, TIME
	변환 함수	CAST, CONVERT, DATE_FORMAT, STR_TO_DATE
	정보 함수	DATABASE, SCHEMA, ROW_COUNR, USER, VERSION
	NULL 관련 함수	COALESCE, ISNULL, IFNULL, NULLIF
집계 함수		AVG, COUNT, MAX, MIN, STD, STDDEV, SUM
윈도우 함수(혹은 분석 함수)		CUME_DIST, DENSE_RANK, FIRST_VALUE, LAST_VALUE, LEAD, NTILE, RANK, ROW_NUMBER

숫자함수

함수	설명
ABS(숫자)	숫자의 절댓값을 계산 ABS(-4.5) => 4.5
CEIL(숫자)	숫자보다 크거나 같은 최소의 정수 CEIL(4.1) => 5
FLOOR(숫자)	숫자보다 작거나 같은 최소의 정수 FLOOR(4.1) => 4
ROUND(숫자, m)	숫자의 반올림, m은 반올림 기준 자릿수 ROUND(5.36, 1) => 5.40
LOG(n, 숫자)	숫자의 자연로그 값을 반환 LOG(10) => 2.30259
POWER(숫자, n)	숫자의 n제곱 값을 계산 POWER(2, 3) => 8
SQRT(숫자)	숫자의 제곱근 값을 계산(숫자는 양수) SQRT(9.0) => 3.0
SIGN(숫자)	숫자가 음수면 -1, 0이면 0, 양수면 1 SIGN(3.45) => 1

문자 함수

반환 구분	함수	설명
문자값 반환 함수 s : 문자열 c : 문자 n : 정수 k : 정수	CONCAT(s1,s2)	두 문자열을 연결, CONCAT('마당', ' 서점') => '마당 서점'
	LOWER(s)	대상 문자열을 모두 소문자로 변환, LOWER('MR. SCOTT') => 'mr. scott'
	LPAD(s,n,c)	대상 문자열의 왼쪽부터 지정한 자리수까지 지정한 문자로 채움 LPAD('Page 1', 10, '*') => '****Page 1'
	REPLACE(s1,s2,s3)	대상 문자열의 지정한 문자를 원하는 문자로 변경 REPLACE('JACK & JUE', 'J', 'BL') => 'BLACK & BLUE'
	RPAD(s,n,c)	대상 문자열의 오른쪽부터 지정한 자리수까지 지정한 문자로 채움 RPAD('AbC', 5, '*') => 'AbC**'
	SUBSTR(s,n,k)	대상 문자열의 지정된 자리에서부터 지정된 길이만큼 잘라서 반환 SUBSTR('ABCDEFG', 3, 4) => 'CDEF'
	TRIM(c FROM s)	대상 문자열의 양쪽에서 지정된 문자를 삭제(문자열만 넣으면 기본값으로 공백 제거) TRIM('=' FROM '==BROWNING==') => 'BROWNING'
	UPPER(s)	대상 문자열을 모두 대문자로 변환 UPPER('mr. scott') => 'MR. SCOTT'
숫자값 반환 함수	ASCII(c)	대상 알파벳 문자의 아스키 코드 값을 반환, ASCII('D') => 68
	LENGTH(s)	대상 문자열의 Byte 반환, 알파벳 1byte, 한글 3byte (UTF8) LENGTH('CANDIDE') => 7
	CHAR_LENGTH(s)	문자열의 문자 수를 반환, CHAR_LENGTH('데이터') => 3

날짜 함수

함수	반환형	설명
STR_TO_DATE(string, format))	DATE	문자열(STRING) 데이터를 날자형(DATE)으로 반환 STR_TO_DATE('2019-02-14', '%Y-%m-%d') => 2019-02-14
DATE_FORMAT(date, format)	STRING	날짜형(DATE) 데이터를 문자열(VARCHAR)로 반환 DATE_FORMAT('2019-02-14', '%Y-%m-%d') => '2019-02-14'
ADDDATE(date, interval)	DATE	DATE 형의 날짜에서 INTERVAL 지정한 시간만큼 더함 ADDDATE('2019-02-14', INTERVAL 10 DAY) => 2019-02-24
DATE(date)	DATE	DATE 형의 날짜 부분을 반환 SELECT DATE('2003-12-31 01:02:03'); => 2003-12-31
DATEDIFF(date1, date2)	INTEGER	DATE 형의 date1 – date2 날짜 차이를 반환 SELECT DATEDIFF('2019-02-14', '2019-02-04') => 10
SYSDATE	DATE	DBMS 시스템상의 오늘 날짜를 반환하는 함수 SYSDATE() => 2018-06-30 21:47:01

날짜 함수

인자	설명
%w	요일 순서(0~6, Sunday=0)
%W	요일(Sunday~Saturday)
%a	요일의 약자(Sun~Sat)
%d	1달 중 날짜(00~31)
%j	1년 중 날짜(001~366)
%h	12시간(01~12)
%H	24시간(00~23)
%i	분(0~59)
%m	월 순서(01~12, January=01)
%b	월 이름 약어(Jan~Dec)
%M	월 이름(January~December)
%s	초(0~59)
%Y	4자리 연도
%y	4자리 연도의 마지막 2 자리