



SQL

inky4832@daum.net

chapter **01.**

MariaDB 10.11.1 RC 설치

https://mariadb.org/download/?t=mariadb&p=mariadb&r=10.11.1&os=windows&cpu=x86_64&pkg=msi&m=blendbyte

**WATCH MORE!**
 **SUBSCRIBE**
to our YouTube channel

[Download MariaDB Server](#)
[REST API](#)
[Release Schedule](#)
[Reporting Bugs](#)
[MariaDB Server Statistics](#)
[Download MariaDB Server Documentation](#)
View all releases for:
[MariaDB Server](#)
[MariaDB Galera](#)
[Connector/C](#)
[Connector/J](#)
[Connector/ODBC](#)
[Connector/Python](#)
[Connector/Node.js](#)

Download MariaDB Server

MariaDB Server is one of the world's most popular open source relational databases and is available in the standard repositories of all major Linux distributions. Look for the package mariadb-server using the package manager of your operating system. Alternatively you can use the following resources:

MariaDB Server

MariaDB Repositories

Connectors

MariaDB Server Version
MariaDB Server 10.11.1 RC
☐ Display older releases:

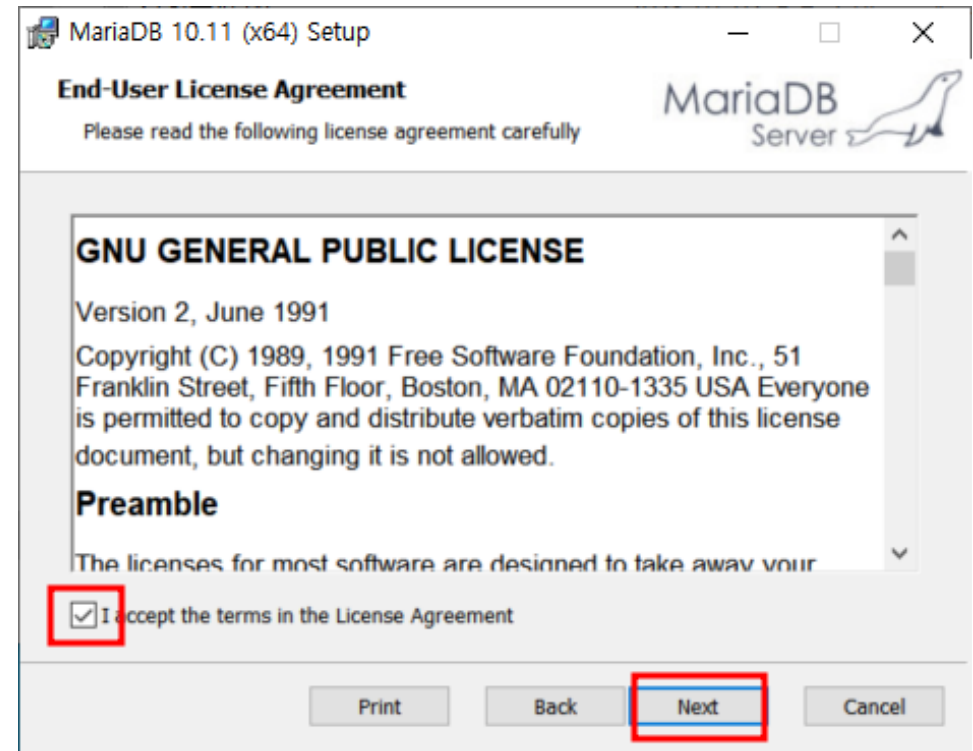
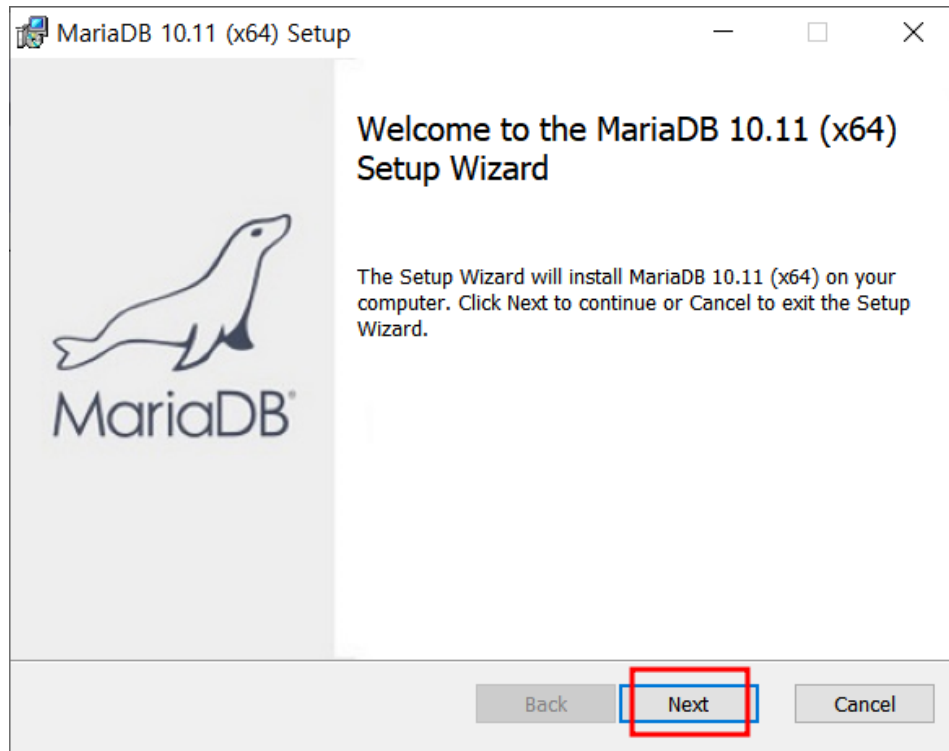
Operating System
Windows

Architecture
x86_64

Package Type
MSI Package

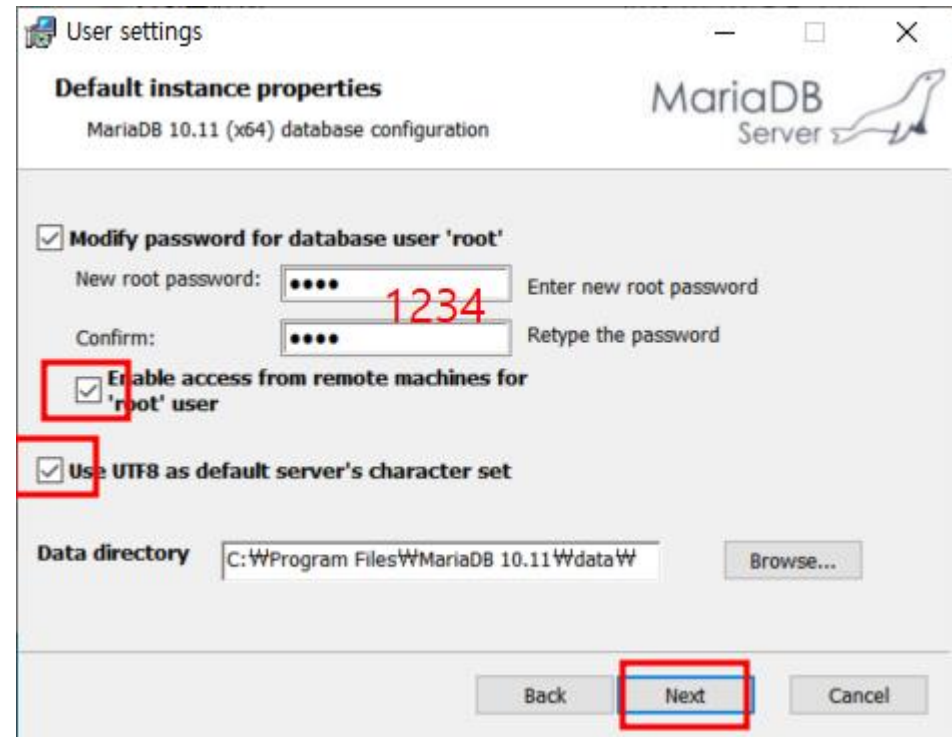
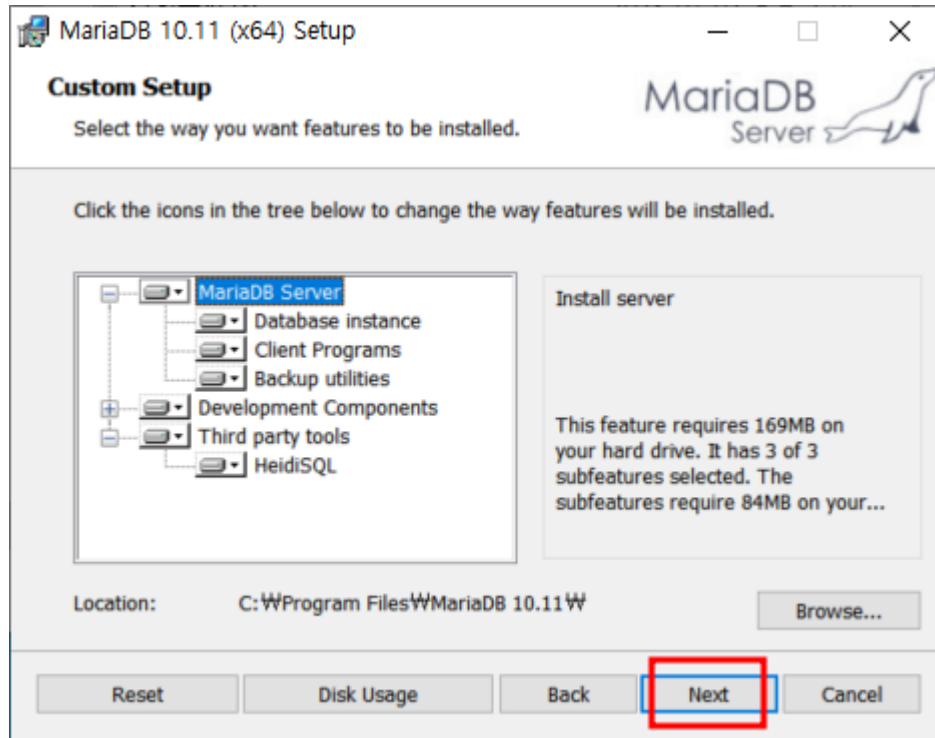
Mirror
Blendbyte - Taipei

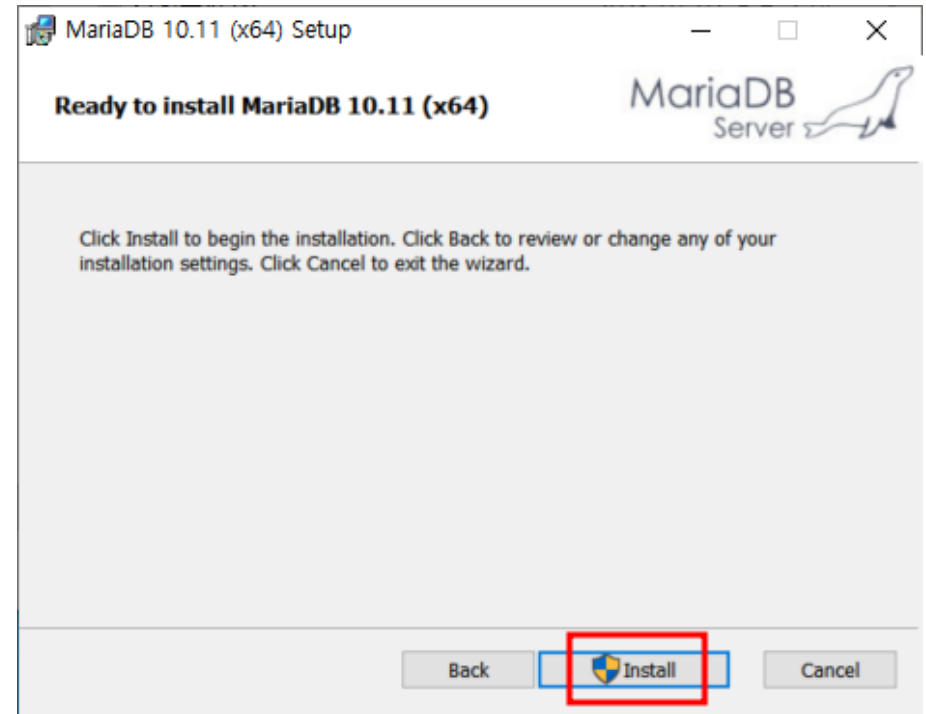
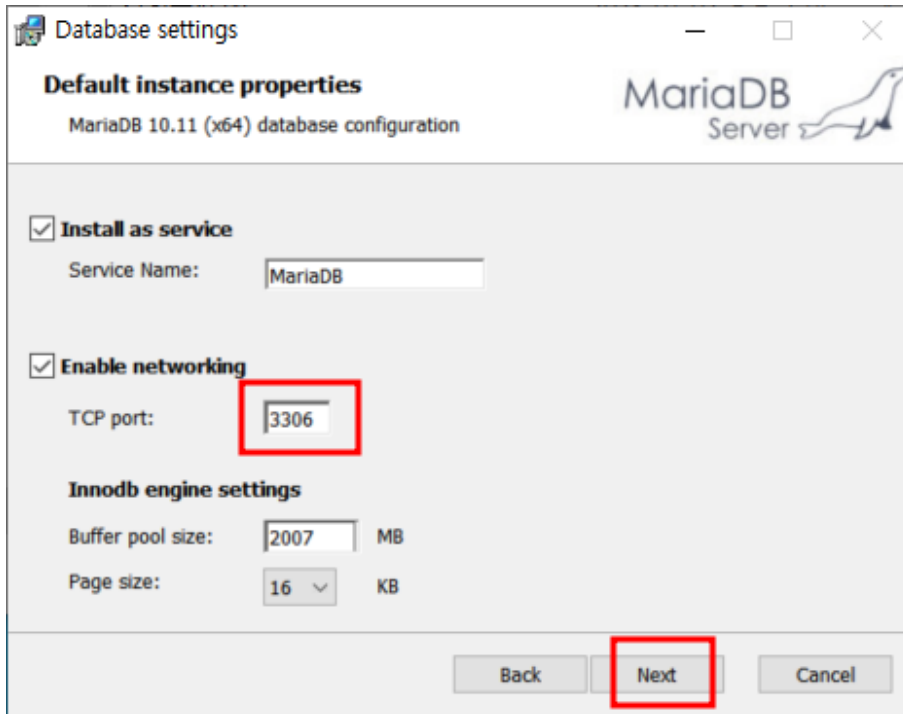
Download

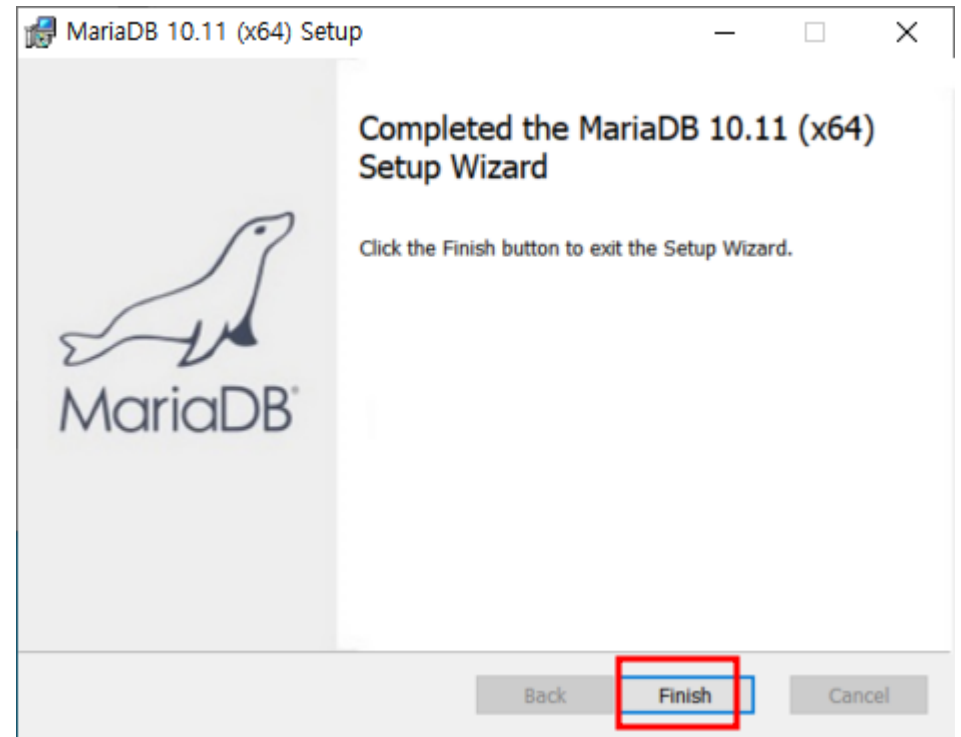
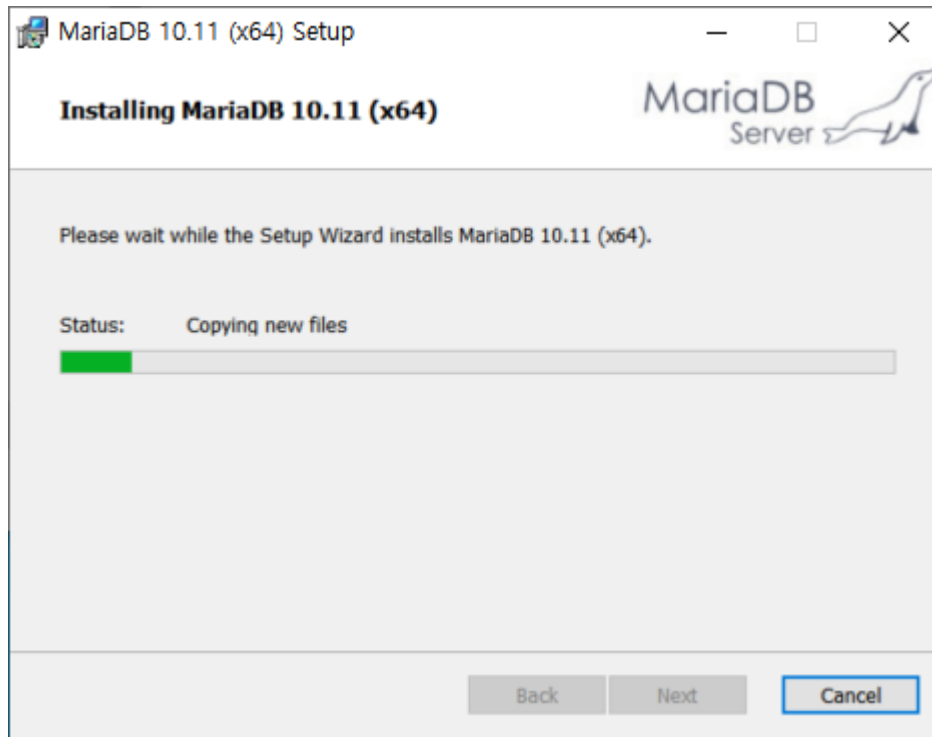


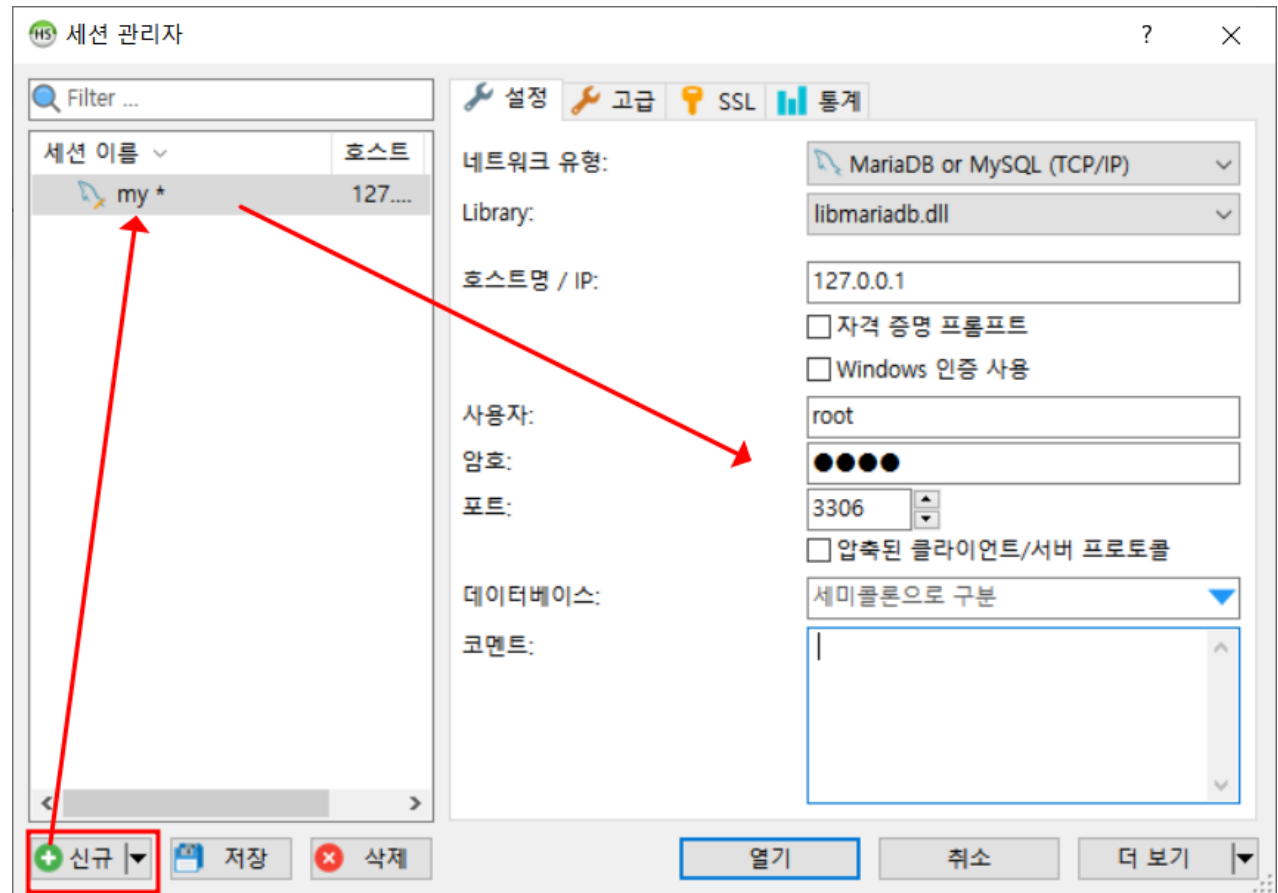
❑ MariaDB 10.11.1 RC 설치

SQL









The screenshot shows the HeidiSQL 11.3.0.6295 interface. The 'File' menu is open, and the 'New' (새로 생성(O)) option is highlighted. A red arrow points from 'New' to the 'Database' (데이터베이스(S)) option in the submenu. The 'Database Creation' dialog box is open, showing the name 'ktds' in the 'Name' field, which is also highlighted with a red box. The 'Collation' is set to 'utf8mb4_general_ci'. The 'Server default collation' is also 'utf8mb4_general_ci'. The 'Create Code' field shows the SQL command: `CREATE DATABASE `ktds` /*!40100 COLLATE 'u`.

HeidiSQL 11.3.0.6295

파일 편집 검색 쿼리 도구 이동 도움말

데이터베이스 필터 테이블 필터

호스트: 127.0.0.1 쿼리

변수목록 상태 프로세스 명령-통계

my

편집(K) Alt+Enter

삭제...(L)

테이블 비우기...(M) Shift+Del

루틴 실행 (N)

새로 생성(O)

데이터 탭 필터 초기화(P)

데이터베이스를 SQL로 내보내기(S)

유지보수(Q)

서버 내에서 텍스트 찾기(R) Shift+Ctrl+F

블록 테이블 편집기(T)

모두 펼치기(U)

모두 접기(V)

트리 방식 옵션(W)

인쇄...(X) Ctrl+P

새로고침(Y) F5

Connection properties

연결 해제(Z)

데이터베이스(S)

테이블(T)

테이블 복사(U)

뷰(V)

저장 프로시저(W)

저장 함수(X)

트리거(Y)

이벤트(Z)

데이터베이스 생성...

이름(N) ktds

조합(O) utf8mb4_general_ci

서버 기본값: utf8mb4_general_ci

확인 취소

CREATE 코드:

CREATE DATABASE `ktds` /*!40100 COLLATE 'u

33 SHOW FUNCTION STATUS WHERE `Db`='mysql';

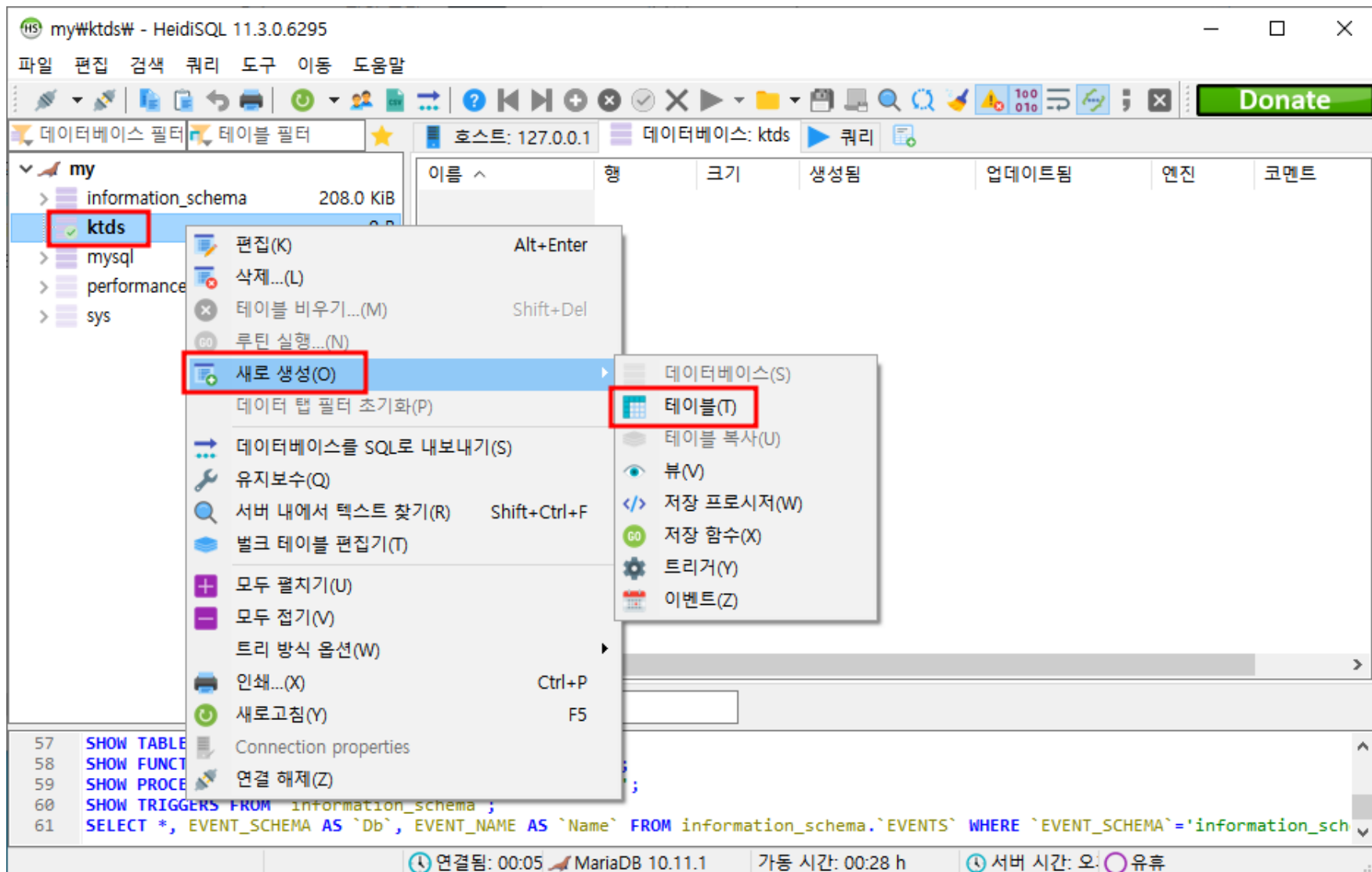
34 SHOW PROCEDURE STATUS WHERE `Db`='mysql';

36 SHOW TRIGGERS FROM `mysql`;

37 SELECT *, EVENT_SCHEMA AS `Db`, EVENT_NAME AS `Name` FROM information_schema.`EVENTS` WHERE

비어있는 새 데이터베이스 생성

연결됨: 00:01 MariaDB 10.11.1 가동 시간: 00:24 h 서버



제공된 scott_mariadb.sql 사용하여 실습 테이블 생성하기

```
1  # scott_mariadb.sql
2  CREATE TABLE IF NOT EXISTS DEPT
3  (DEPTNO INT PRIMARY KEY,
4   DNAME VARCHAR(14) ,
5   LOC VARCHAR(13) );
6
7  CREATE TABLE emp
8  (EMPNO INT,
9   ENAME VARCHAR(10),
10  JOB VARCHAR(9),
11  MGR INT,
12  HIREDATE DATE,
13  SAL FLOAT,
14  COMM INT,
15  DEPTNO INT,
16  PRIMARY KEY(EMPNO),
17  FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO) );
18
19  CREATE TABLE SALGRADE
20  ( GRADE INT,
21   LOSAL INT,
22   HISAL INT );
23
24
25  INSERT INTO DEPT VALUES
26  (10,'ACCOUNTING','NEW YORK');
27  INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
28  INSERT INTO DEPT VALUES
29  (30,'SALES','CHICAGO');
30  INSERT INTO DEPT VALUES
31  (40,'OPERATIONS','BOSTON');
32  INSERT INTO EMP VALUES
33  (7369,'SMITH','CLERK',7902,STR_TO_DATE('17-12-1980','%d-%m-%Y'),800,NULL,20);
34  INSERT INTO EMP VALUES
35  (7499,'ALLEN','SALESMAN',7698,STR_TO_DATE('20-2-1981','%d-%m-%Y'),1600,300,30);
36  INSERT INTO EMP VALUES
37  (7521,'WARD','SALESMAN',7698,STR_TO_DATE('22-2-1981','%d-%m-%Y'),1250,500,30);
38  INSERT INTO EMP VALUES
```

실습에서 사용할 테이블은 다음과 같은 3가지 테이블이다.

emp 테이블(사원 테이블)

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	80/12/17	800		20
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30
7566	JONES	MANAGER	7839	81/04/02	2975		20
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30
7698	BLAKE	MANAGER	7839	81/05/01	2850		30
7782	CLARK	MANAGER	7839	81/06/09	2450		10
7788	SCOTT	ANALYST	7566	87/04/19	3000		20
7839	KING	PRESIDENT		81/11/17	5000		10
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30
7876	ADAMS	CLERK	7788	87/05/23	1100		20
7900	JAMES	CLERK	7698	81/12/03	950		30
7902	FORD	ANALYST	7566	81/12/03	3000		20
7934	MILLER	CLERK	7782	82/01/23	1300		10

다음과 같이 총 8개의 컬럼으로 구성되어 있다.

- 사원번호(empno)
- 사원명 (ename)
- 직급 (job)
- 관리자 번호 (mgr)
- 입사일 (hiredate)
- 급여 (sal)
- 커미션(comm)
- 부서번호 (deptno)

dept 테이블 (부서 테이블)

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

부서 테이블 이름은 dept 이며 다음과 같이 총 3개의 컬럼으로 구성되어 있다.

- 부서번호 (deptno)
- 부서명 (dname)
- 부서위치 (loc)

salgrade 테이블 (sal 등급 테이블)

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

급여등급 테이블 이름은 salgrade 이며 다음과 같이 총 3개의 컬럼으로 구성되어 있다.

- 등급 (grade)
- 최소월급 (losal)
- 최대월급 (hisal)



chapter **02.**

SELECT

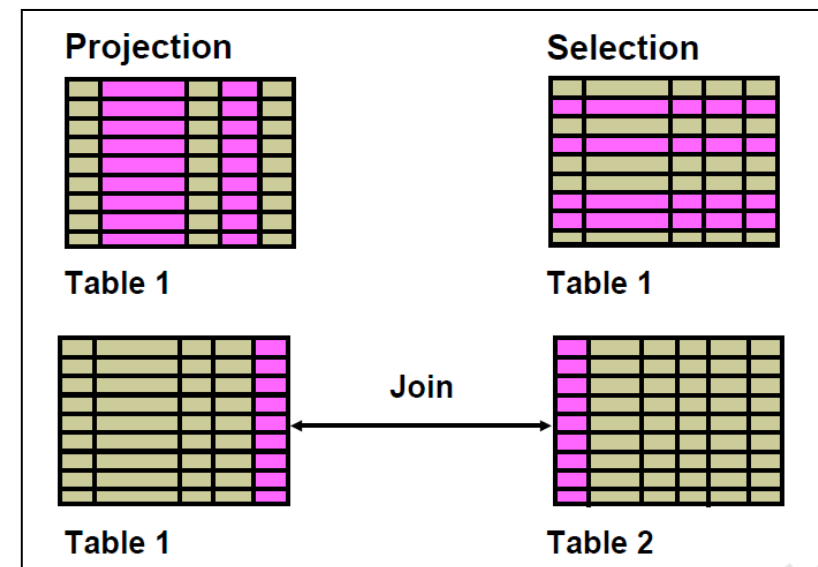
■ SELECT 기능

: 데이터베이스로부터 데이터를 검색하는 기능을 갖는다.

- selection : 질의에 대해 테이블의 행을 선택하기 위해 사용.
- projection : 질의에 대해 테이블의 열을 선택하기 위해 사용.
- join : 여러 테이블이 공통적으로 가진 컬럼을 이용해서 다른 테이블에 저장되어 있는 데이터를 가져오기 위해 사용.

■ 기본적인 SELECT 문법

```
SELECT [DISTINCT] { *, column [alias],... }  
FROM table;
```



테이블내의 모든 데이터 보기

```
SELECT * FROM dept;
```

dept (4r x 3c)			
DEPTNO		DNAME	LOC
10		ACCOUNTING	NEW YORK
20		RESEARCH	DALLAS
30		SALES	CHICAGO
40		OPERATIONS	BOSTON

테이블내의 특정 컬럼 데이터 보기

- SELECT 뒤에 해당 컬럼을 차례대로 기술. 쉼표로 구분해서 여러 개 지정 가능.

```
SELECT empno, ename, job, hiredate  
FROM emp
```

emp (14r x 4c)			
empno	ename	job	hiredate
7,369	SMITH	CLERK	1980-12-17
7,499	ALLEN	SALESMAN	1981-02-20
7,521	WARD	SALESMAN	1981-02-22
7,566	JONES	MANAGER	1981-04-02
7,654	MARTIN	SALESMAN	1981-09-28
7,698	BLAKE	MANAGER	1981-05-01
7,782	CLARK	MANAGER	1981-06-09
7,788	SCOTT	ANALYST	1987-06-28

□ 산술연산을 이용한 SQL 문

SQL

- SQL 문장내의 숫자 타입에는 + , - , * , / 사용 가능하다.
- 연산식이 컬럼명으로 표시된다.

```
SELECT empno, ename, sal * 1.1, ROUND(sal * 1.1)
FROM emp;
```

empno	ename	sal * 1.1	ROUND(sal * 1.1)
7,369	SMITH	880.00000000000001	880
7,499	ALLEN	1,760.00000000000002	1,760
7,521	WARD	1,375	1,375
7,566	JONES	3,272.50000000000005	3,273
7,654	MARTIN	1,375	1,375

- 컬럼에 별칭(alias) 사용.

```
SELECT empno AS 사번, ename AS 성명, sal 급여
FROM emp;
```

사번	성명	급여
7,369	SMITH	800
7,499	ALLEN	1,600
7,521	WARD	1,250
7,566	JONES	2,975

- 이용 불가능한(unavailable), 비교 자체가 불가능한
- 지정되지 않은(unassigned)
- 알 수 없는 (unknown)
- 적용할 수 없는(Inapplicable) 값을 의미한다.

* MariaDB 컬럼에 기본적으로 null 값을 허용하며 제약조건을 이용해서 null 값을 허용하지 않을 수도 있다.

주의할 점은 null 값의 연산결과는 null 값으로 나온다는 것이다.

* null값의 비교는 IS NULL , IS NOT NULL 이라는 정해진 문구를 사용해야 제대로 된 결과를 얻을 수 있다.

```
SELECT empno, ename, comm, comm + 10  
FROM emp;
```

emp (14r x 4c)			
empno	ename	comm	comm + 10
7,369	SMITH	(NULL)	(NULL)
7,499	ALLEN	300	310
7,521	WARD	500	510
7,566	JONES	(NULL)	(NULL)
7,654	MARTIN	1,400	1,410

null 값을 가진 컬럼을 연산하기 위해서는 NVL 함수를 사용할 수 있다.
NVL 함수의 사용법은 다음과 같으며 컬럼값이 null인 경우에 기본값으로 설정한다.

NVL (컬럼명, 기본값)

```
SELECT empno, ename, comm, NVL(comm, 0) + 100
FROM emp;
```

emp (14r × 4c)			
empno	ename	comm	NVL(comm, 0) + 100
7,369	SMITH	(NULL)	100
7,499	ALLEN	300	400
7,521	WARD	500	600
7,566	JONES	(NULL)	100
7,654	MARTIN	1,400	1,500

- DISTINCT 키워드 이용
중복된 값 제거하여 한번만 출력.

```
SELECT job  
FROM emp;
```

emp (14r x 1c)	
job	
CLERK	
SALESMAN	
SALESMAN	
MANAGER	
SALESMAN	
MANAGER	
MANAGER	
ANALYST	
PRESIDENT	
SALESMAN	
CLERK	
CLERK	
ANALYST	
CLERK	



```
SELECT distinct job  
FROM emp;
```

emp (5r x 1c)	
job	
CLERK	
SALESMAN	
MANAGER	
ANALYST	
PRESIDENT	

WHERE 기능

- 테이블내의 모든 행을 검색하는 대신 검색 조건을 지정하여 사용자가 원하는 행들만 검색.

기본적인 SELECT ~ WHERE 문법

```
SELECT [DISTINCT] { *, column [alias],... }
FROM table
[WHERE 조건식];
```

```
SELECT empno, ename, job, deptno
FROM emp
WHERE deptno = 30;
```

empno	ename	job	deptno
7,499	ALLEN	SALESMAN	30
7,521	WARD	SALESMAN	30
7,654	MARTIN	SALESMAN	30
7,698	BLAKE	MANAGER	30
7,844	TURNER	SALESMAN	30
7,900	JAMES	CLERK	30

```
SELECT empno, ename, job, deptno
FROM emp
WHERE job = 'salesman';
```

empno	ename	job	deptno
7,499	ALLEN	SALESMAN	30
7,521	WARD	SALESMAN	30
7,654	MARTIN	SALESMAN	30
7,844	TURNER	SALESMAN	30

연산자	의미
=	같다
>	보다 크다
>=	보다 크거나 같다
<	보다 작다
<=	보다 작거나 같다
<>	다르다 != 동일

```
SELECT empno, ename, job, deptno
FROM emp
WHERE sal <= 1000;
```

emp (2r x 3c)		
empno	ename	sal
7,369	SMITH	800
7,900	JAMES	950

연산자	의미
BETWEEN ... AND ...	두 값의 범위에 포함되는
IN (set)	괄호 안의 값과 일치하는
LIKE	문자의 조합이 같은
IS NULL	널 값

```
SELECT empno, ename, job, deptno
FROM emp
WHERE sal BETWEEN 800 AND 2000;
```

```
SELECT empno, ename, sal, comm
FROM emp
WHERE comm IS NULL;
```

```
SELECT empno, ename, sal
FROM emp
WHERE empno IN (7369, 7566, 7698);
```

- LIKE 연산자

- 검색하고자 하는 문자열을 정확히 알 수 없는 경우에 사용.
- 패턴 매칭 연산자 이용 (와일드 카드)

기호	설명
%	0 글자 이상의 임의 문자를 대표한다.
_	1 글자의 임의 문자를 대표한다

```
SELECT empno, ename, sal
FROM emp
WHERE ename LIKE 'A%';
```

```
SELECT empno, ename, sal
FROM emp
WHERE ename LIKE '%T%';
```

```
SELECT empno, ename, sal
FROM emp
WHERE ename LIKE '_L%';
```

- 검색하고자 하는 문자열에 패턴 매칭 연산자가 포함되어 있을 때는 ESCAPE 옵션을 사용한다.

```
SELECT empno, ename, sal
FROM emp
WHERE ename LIKE '%E_%' ESCAPE 'E';
```


- WHERE 절에 부여할 조건이 여러 개인 경우에 사용한다.

연산자	의미
AND	두개의 조건이 TRUE이면 TRUE를 리턴
OR	두개의 조건중 하나의 조건이 TRUE이면 TRUE를 리턴
NOT	조건이 FALSE이면 TRUE를 리턴

```
SELECT empno, ename, job, deptno, hiredate
FROM emp
WHERE job = 'salesman' AND sal >= 1500;
```

```
SELECT empno, ename, job, deptno, hiredate
FROM emp
WHERE job = 'salesman' OR sal >= 1500;
```

```
SELECT empno, ename, sal, comm
FROM emp
WHERE comm IS NOT NULL;
```

- WHERE 절에 사용 가능한 부정 연산자.

	연산자	의미
부정 비교 연산자	!=	같지 않다.
	^=	같지 않다.
	<>	같지 않다.
	NOT 컬럼명 =	~ 와 같지 않다.
	NOT 컬럼명 >	~ 보다 크지 않다.
부정 SQL 연산자	NOT BETWEEN a AND b	a와 b값 사이에 있지 않다.
	NOT IN (리스트)	리스트값과 일치하지 않는다.
	IS NOT NULL	NULL 값을 갖지 않는다.

- SELECT 문장에 의해 검색된 결과를 정렬 (기본은 오름차순, 다중정렬 가능)

```
SELECT [DISTINCT] { *, column [alias],... }
FROM table
[WHERE 조건식]
[ORDER BY {column|exp } [ASC|DESC] ];
```

- null
 - MariaDB은 null값을 가장 작은 값으로 간주한다.

```
SELECT empno, ename, hiredate
FROM emp
ORDER BY hiredate;
```

```
SELECT empno, ename, hiredate
FROM emp
ORDER BY hiredate DESC ;
```

```
SELECT empno, ename, sal * 12 AS Annual
FROM emp
ORDER BY Annual;
```

```
SELECT empno, ename, sal * 12 AS Annual
FROM emp
ORDER BY 1;
```

```
SELECT empno, ename, sal * 12 AS Annual  
FROM emp  
ORDER BY 3, 1 DESC;
```

```
SELECT empno, ename, comm  
FROM emp  
ORDER BY 3 DESC;
```

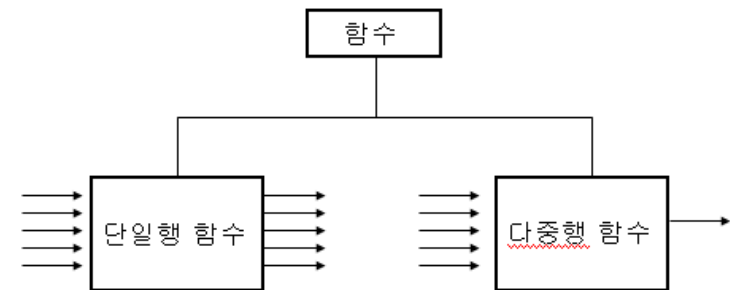
chapter 03.

단일 함수

▪ 제공되는 함수들은 기본적인 query문을 더욱 강력하게 해주고 데이터 값을 조작하는데 사용할 수 있다.

▪ SQL 함수의 특징 및 이점

- 데이터에 계산을 수행할 수 있다.
- 개별적인 데이터 항목을 수정할 수 있다.
- 행의 그룹에 대해 결과를 조작할 수 있다.
- 출력을 위한 날짜와 숫자 형식을 조작할 수 있다.
- 열의 자료형을 변환할 수 있다.



▪ 함수의 종류

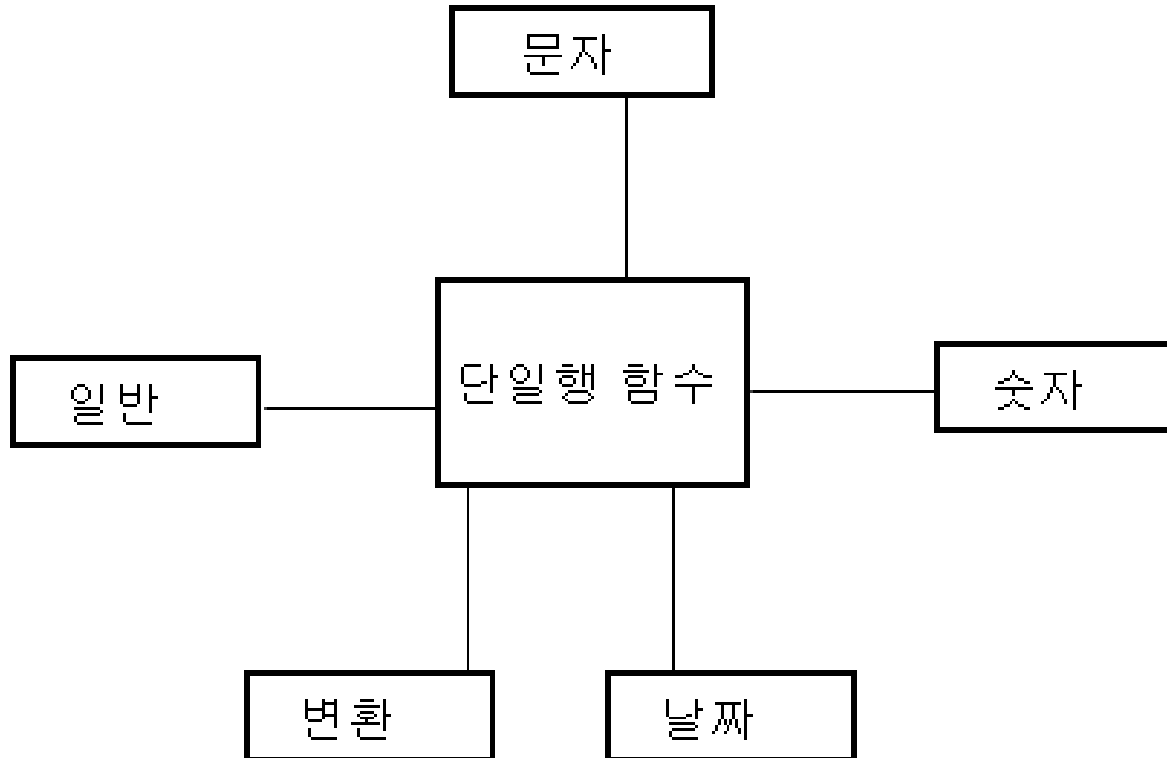
▪ 단일 행 함수

- : 모든 행에 대해서 각각 적용되어 행의 개수와 동일한 개수의 결과를 리턴.
- : SELECT, WHERE, ORDER BY절에 사용 가능하다.

▪ 다중 행 함수 (그룹 함수)

- : 검색되는 모든 행에 대해서 한번만 적용되고 한 건의 결과만을 리턴.

■ 단일 행 함수의 종류



```
# https://docs.oracle.com/cd/E17952\_01/mysql-8.0-en/string-functions.html
```

■ LOWER 함수

- 대소문자가 혼합되어 있거나 대문자인 문자열을 소문자로 변경.

문법: LOWER(str)

```
SELECT empno, lower(ename), LOWER('HeLLo')  
FROM emp;
```

■ UPPER 함수

- 대소문자가 혼합되어 있거나 소문자인 문자열을 대문자로 변경.

문법: UPPER(str)

```
SELECT empno, UPPER(ename), UPPER('HeLLo')  
FROM emp;
```


▪ CONCAT 함수

- 여러 개의 문자열을 연결한다.

문법: CONCAT(str1,str2,...)

```
SELECT empno, CONCAT(ename, ' ', sal)
FROM emp;
```

▪ LPAD 함수

– 왼쪽에 문자열을 끼워 넣는 역할을 한다. len은 반환되는 문자열의 전체 길이를 나타낸다.

문법: LPAD(str,len,padstr)

```
SELECT empno, LPAD(ename,10, '*'), LPAD(sal, 10, '*')  
FROM emp;
```

emp (14r x 3c)		
empno	LPAD(ename,10, '*')	LPAD(sal, 10, '*')
7,369	*****SMITH	*****800
7,499	*****ALLEN	*****1600
7,521	*****WARD	*****1250

■ RPAD 함수

– LPAD와 반대로 오른쪽에 문자열을 끼워 넣는 역할을 한다.

문법: RPAD(str,len,padstr)

```
SELECT empno, RPAD(ename,10, '*'), RPAD(sal, 10, '*')  
FROM emp;
```

emp (14r x 3c)		
empno	RPAD(ename,10, '*')	RPAD(sal, 10, '*')
7,369	SMITH*****	800*****
7,499	ALLEN*****	1600*****
7,521	WARD*****	1250*****

■ SUBSTR 함수

- pos 번째 자리부터 길이가 len개인 문자열을 반환 한다.
- pos는 순방향은 1부터 시작하고 역방향은 -1부터 시작한다.

문법: SUBSTR(str,pos [,len])

```
SELECT empno, ename, SUBSTR(ename,1,2), SUBSTR(ename,3),  
        SUBSTR(ename, -1)  
FROM emp;
```

emp (14r x 5c)				
empno	ename	SUBSTR(ename,1,2)	SUBSTR(ename,3)	SUBSTR(ename, -1)
7,369	SMITH	SM	ITH	H
7,499	ALLEN	AL	LEN	N
7,521	WARD	WA	RD	D

■ LENGTH 함수

– 문자열의 길이를 반환한다.

문법: LENGTH(str)

```
SELECT empno, ename, LENGTH(ename)
FROM emp;
```

emp (14r x 3c)		
empno	ename	LENGTH(ename)
7,369	SMITH	5
7,499	ALLEN	5
7,521	WARD	4

■ REPLACE 함수

– 특정 문자열을 치환하는 함수 .

문법: REPLACE(str, from, to)

```
SELECT empno, ename, sal, REPLACE(sal, '0', 'o')  
FROM emp;
```

emp (14r x 4c)			
empno	ename	sal	REPLACE(sal, '0', 'o')
7,369	SMITH	800	8oo
7,499	ALLEN	1,600	16oo
7,521	WARD	1,250	125o

■ INSTR 함수

- 문자열이 포함되어 있는지를 조사하여 문자열의 위치를 반환 한다.
- 지정한 문자열이 발견되지 않으면 0이 반환 됩니다.

문법: INSTR(str, substr)

```
SELECT INSTR('foobarbar', 'bar'), INSTR('foobarbar', 'xbar');
```

■ LTRIM함수

-문자열의 첫 문자부터 확인해서 공백 문자가 나타나는 동안 해당 문자를 제거.

문법: LTRIM(str)

```
SELECT LTRIM(' bar '), LENGTH(LTRIM(' bar '));
```

결과 #1 (1r x 2c)	
LTRIM(' bar ')	LENGTH(LTRIM(' bar '))
bar	8

■ RTRIM함수

-문자열의 끝 문자부터 확인해서 공백 문자가 나타나는 동안 해당 문자를 제거.

문법: RTRIM(str)

```
SELECT RTRIM(' bar '), LENGTH(RTRIM(' bar '));
```

결과 #1 (1r x 2c)	
RTRIM(' bar ')	LENGTH(RTRIM(' bar '))
bar	8

■ TRIM함수

- 문자열에서 머리말,꼬리말 또는 양쪽에 있는 공백 문자를 제거.
(leading | trailing | both 가 생략되면 both가 기본)

문법: TRIM([BOTH|LEADING|TRAILING from] str)

```
SELECT TRIM('      bar      '),  
       TRIM(BOTH FROM '      bar      '),  
       TRIM(LEADING FROM '      bar      '),  
       TRIM(TRAILING FROM '      bar      ');
```

▪ REVERSE함수

-문자열 값을 거꾸로 반환한다.

문법: REVERSE(str)

```
SELECT ename, REVERSE(ename)
FROM emp;
```

emp (14r x 2c)	
ename	REVERSE(ename)
SMITH	HTIMS
ALLEN	NELLA
WARD	DRAW
JONES	SENOJ

■ FORMAT 함수

– 지정된 숫자값을 ‘#,###,###.##’ 같은 포맷을 적용하여 문자열로 반환.

문법: `FORMAT(정수, 소수점자릿수)`

```
SELECT FORMAT (9876543.2145, 2)
```

결과 #1 (1r × 1c)

FORMAT(9876543.2145, 2)

9,876,543.21

```
# https://docs.oracle.com/cd/E17952\_01/mysql-8.0-en/numeric-functions.html
```

▪ ROUND 함수

- 숫자를 소수점 n자리에서 반올림한다. 생략하면 기본값은 0.
- n이 양수이면 소수 자리를, 음수이면 정수 자리를 반올림한다.

문법: ROUND(정수 [, n])

```
SELECT ROUND (45.678) , ROUND (45.678, 2) , ROUND (45.678, -1)  
FROM DUAL;
```

DUAL (1r x 3c)		
ROUND(45.678)	ROUND(45.678, 2)	ROUND(45.678, -1)
46	45.68	50

▪ TRUNCATE 함수

- 숫자를 소수 n자리에서 절삭한다. 생략불가
- n이 양수이면 소수 자리를, 음수이면 정수 자리를 절삭.

문법: TRUNCATE(정수 , n)

```
SELECT TRUNCATE (45.678, 0) , TRUNCATE (45.678, 2) ,  
       TRUNCATE (45.678, -1)  
FROM DUAL;
```

DUAL (1r x 3c)		
TRUNCATE(45.678, 0)	TRUNCATE(45.678, 2)	TRUNCATE(45.678, -1)
45	45.67	40

- CEIL(CEILING) 함수 (올림값)
 - 주어진 숫자보다 크거나 같은 최소 정수 리턴.

문법: CEIL(실수값)

```
SELECT CEIL(45.178), CEIL(-45.178);
```

결과 #1 (1r x 2c)	
CEIL(45.178)	CEIL(-45.178)
46	-45

- FLOOR 함수 (버림값)
 - 주어진 숫자보다 작거나 같은 최대 정수 리턴.

문법: FLOOR(실수값)

```
SELECT FLOOR(45.178), FLOOR(-45.178);
```

결과 #1 (1r x 2c)	
FLOOR(45.178)	FLOOR(-45.178)
45	-46

■ MOD 함수

- 숫자의 나머지를 구하는 함수로서 % 연산자와 동일
- n으로 나누어 남은 값을 반환한다. n이 0일 경우 그 자체를 반환.

문법: MOD(m, n)

```
SELECT MOD(10, 3), 10%3, 10 MOD 3 FROM DUAL;
```

■ SIGN 함수

- 숫자의 부호를 식별하는 함수로서 음수(-1), 양수(1), 0(0) 값 반환

문법: SIGN(값)

```
SELECT SIGN(-1.200), SIGN(34.3), SIGN(0) FROM DUAL;
```

DUAL (1r x 3c)		
SIGN(-1.200)	SIGN(34.3)	SIGN(0)
-1	1	0

```
# https://docs.oracle.com/cd/E17952\_01/mysql-8.0-en/date-and-time-functions.html
```

■ CURDATE 함수

- 데이터베이스 서버에 설정되어 있는 날짜를 리턴. ('YYYY-MM-DD')
- current_date() 와 current_date 는 curdate()의 synonym이다.

문법: CURDATE()

```
SELECT CURDATE ( ) ,    CURRENT_DATE ( ) ,    CURRENT_DATE FROM DUAL;
```

DUAL (1r x 3c)		
CURDATE()	CURRENT_DATE()	CURRENT_DATE
2023-01-09	2023-01-09	2023-01-09

■ CURTIME 함수

- 데이터베이스 서버에 설정되어 있는 시간을 리턴. ('hh:mm:ss')
- current_time() 와 current_time 는 curtime()의 synonym이다.

문법: CURTIME()

```
SELECT CURTIME ( ) ,    CURRENT_TIME ( ) ,    CURRENT_TIME FROM DUAL;
```

DUAL (1r × 3c)		
CURTIME()	CURRENT_TIME()	CURRENT_TIME
12:26:50	12:26:50	12:26:50

■ SYSDATE 함수

- 데이터베이스 서버에 설정되어 있는 날짜 및 시간을 리턴.
('YYYY-MM-DD hh:mm:ss')

문법: SYSDATE()

```
SELECT SYSDATE ( ) , NOW ( ) FROM DUAL;
```

DUAL (1r × 2c)	
SYSDATE()	NOW()
2023-01-09 12:32:38	2023-01-09 12:32:38

■ 날짜 함수

- 날짜 함수는 MariaDB 날짜에 대해 연산을 한다.

함수명	설명
ADDDATE	날짜에 일(day) 더하기 및 빼기
DATE_ADD	날짜에 입력된 기간(interval)만큼 더한다.
DATE_SUB	날짜에 입력된 기간(interval)만큼 뺀다
DATEDIFF	날짜간 일(day) 차이를 반환한다
TIMESTAMPDIFF	날짜에 지정된 단위(unit)만큼 차이를 반환한다.
LAST_DAY	월의 마지막 날짜를 반환한다.
EXTRACT	날짜에서 지정된 단위(unit)에 해당하는 날짜 정보 추출
DATE_FORMAT	지정된 날짜를 format 적용하여 문자열로 반환한다.
STR_TO_DATE	지정된 문자를 format 적용하여 날짜로 반환한다.

■ ADDDATE 함수

-날짜에 일(day)을 더하거나 뺀다.

문법: ADDDATE(date, days)

```
SELECT ADDDATE ('2008-01-02' , 30) ;
```

결과 #1 (1r x 1c)
ADDDATE('2008-01-02', 30)
2008-02-01

■ DATE_ADD 함수

-날짜에 입력된 기간(interval)만큼 더한다.

문법: DATE_ADD(date, INTERVAL expr unit)

```
SELECT DATE_ADD('2008-01-02', INTERVAL 1 DAY) as A1,  
       DATE_ADD('2008-01-02', INTERVAL 1 MONTH) as A2,  
       DATE_ADD('2008-01-02', INTERVAL 1 YEAR) as A3,  
       NOW() as A4,  
       DATE_ADD(NOW(), INTERVAL 10 MINUTE) as A5,  
       DATE_ADD(NOW(), INTERVAL 2 HOUR) as A6;
```

결과 #1 (1r x 6c)					
A1	A2	A3	A4	A5	A6
2008-01-03	2008-02-02	2009-01-02	2023-01-09 14:25:03	2023-01-09 14:35:03	2023-01-09 16:25:03

■ DATE_SUB 함수

-날짜에 입력된 기간(interval)만큼 뺀다.

문법: DATE_SUB(date, INTERVAL expr unit)

```
SELECT DATE_SUB('2008-01-02', INTERVAL 1 DAY) as B1,  
       DATE_SUB('2008-01-02', INTERVAL 1 MONTH) as B2,  
       DATE_SUB('2008-01-02', INTERVAL 1 YEAR) as B3,  
       NOW() as B4,  
       DATE_SUB(NOW(), INTERVAL 10 MINUTE) as B5,  
       DATE_SUB(NOW(), INTERVAL 2 HOUR) as B6;
```

결과 #1 (1r x 6c)					
B1	B2	B3	B4	B5	B6
2008-01-01	2007-12-02	2007-01-02	2023-01-09 14:27:34	2023-01-09 14:17:34	2023-01-09 12:27:34

- DATEDIFF함수
- 날짜간 일(day) 차이를 반환한다.

문법: DATEDIFF(date1, date2)

```
SELECT DATEDIFF('2023-01-04', '2022-01-04');
```

결과 #1 (1r x 1c)	
DATEDIFF('2023-01-04', '2022-01-04')	365

- TIMESTAMPDIFF함수
- 날짜에 지정된 단위(unit)만큼 차이를 반환한다.

문법: TIMESTAMPDIFF(unit, date1, date2)

```
SELECT TIMESTAMPDIFF(HOUR, '2020-3-1', '2020-3-3') AS C1,  
       TIMESTAMPDIFF(DAY, '2020-3-1', '2020-3-3') AS C2,  
       TIMESTAMPDIFF(MONTH, '2020-2-1', '2020-6-3') AS C3,  
       TIMESTAMPDIFF(YEAR, '2020-3-1', '2022-3-3') AS C4;
```

■ LAST_DAY 함수

- 월의 마지막 날짜를 계산
- 윤년, 평년은 자동 계산

문법: LAST_DAY(date)

```
SELECT LAST_DAY('2003-02-05'), LAST_DAY(NOW());
```

결과 #1 (1r x 2c)	
LAST_DAY('2003-02-05')	LAST_DAY(NOW())
2003-02-28	2023-01-31

▪ EXTRACT 함수

– 날짜에서 지정된 단위(unit)에 해당하는 날짜 정보 추출.

문법: EXTRACT(unit FROM date)

```
SELECT NOW() ,  
       EXTRACT (SECOND FROM NOW() ) ,  
       EXTRACT (MINUTE FROM NOW() ) ,  
       EXTRACT (HOUR FROM NOW() ) ,  
       EXTRACT (DAY FROM NOW() ) ,  
       EXTRACT (MONTH FROM NOW() ) ,  
       EXTRACT (YEAR FROM NOW() ) ,  
       EXTRACT (YEAR_MONTH FROM NOW() ) ;
```

▪ DATE_FORMAT 함수

– 지정된 날짜를 format 적용하여 문자열로 반환.

문법: DATE_FORMAT(date, format)

Specifier	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th..31st)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)

```
SELECT NOW ( ) ,
       DATE_FORMAT (NOW ( ) , '%Y%m%d' ) ,
       DATE_FORMAT (NOW ( ) , '%Y/%m/%d' ) ,
       DATE_FORMAT (NOW ( ) ,
                    '%Y년%m월 %d일' ) ,
       DATE_FORMAT (NOW ( ) , '%H:%i:%S' ) ,
       DATE_FORMAT (NOW ( ) , '%Y' ) ;
```

▪ STR_TO_DATE 함수

- 지정된 문자를 format 적용하여 날짜로 반환.

문법: STR_DATE_FORMAT(date, format)

```
SELECT STR_TO_DATE('2020-03-04', '%Y-%m-%d'),  
       STR_TO_DATE('01,5,2013', '%d,%m,%Y'),  
       STR_TO_DATE('2020년03월05일', '%Y년%m월%d일');
```

■ CASE 함수

문법:

```
CASE value WHEN compare_value THEN result  
          [WHEN compare_value THEN result ...]  
          [ELSE result] END
```

```
CASE WHEN condition THEN result  
      [WHEN condition THEN result ...]  
      [ELSE result] END
```

```
SELECT empno, ename, sal, job,  
       CASE job WHEN 'ANALYST' THEN sal * 1.1  
               WHEN 'CLERK' THEN sal * 1.2  
               WHEN 'MANAGER' THEN sal * 1.3  
               WHEN 'PRESIDENT' THEN sal * 1.4  
               WHEN 'SALESMAN' THEN sal * 1.5  
               ELSE sal END AS 급여  
FROM emp;
```

```
SELECT empno, ename, sal,  
       CASE      WHEN sal > 3500 THEN '이사급'  
                 WHEN sal > 2500 THEN '과장급'  
                 ELSE '사원급' END AS 직급연봉  
FROM emp;
```

■ IF 함수

문법: IF (조건식, 참, 거짓)

```
SELECT sal, IF(sal > 2500 , '과장이상급', '사원급')  
FROM emp;
```

chapter 04.

그룹 함수

https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/aggregate-functions.html

- 단일 행 함수와는 달리 그룹 함수는 여러 행 또는 테이블 전체에 대해 함수가 적용되어 하나의 결과를 가져오는 함수를 말한다.
- 그룹 당 하나의 결과가 주어지도록 행의 집합에 대해 연산할 경우 GROUP BY절을 이용하여 그룹화 할 수 있고 HAVING 절을 이용하여 그룹에 대한 조건을 제한 할 수 있다.

■ 그룹함수 종류

함수명	설명
SUM([distinct] exp)	총합
AVG([distinct] exp)	평균
MIN([distinct] exp) MAX([distinct] exp)	최소값, 최대값
COUNT([distinct] exp *)	행 갯수

count(*) : null값을 포함한 행의 수
count(표현식): null값을 제외한 행의 수

EMP 테이블에서 모든 SALESMAN에 대한 급여의 평균, 최고액, 최저액, 합계를 출력.

```
SELECT AVG(sal), MAX(sal), MIN(sal), SUM(sal)
FROM emp
WHERE job = 'SALESMAN';
```

EMP 테이블에 등록되어 있는 인원수, 보너스에 null이 아닌 인원수, 보너스의 평균, 등록 되어 있는 부서의 수를 구하여 출력.

```
SELECT COUNT(*), COUNT(comm),
       AVG(comm), AVG(NVL(comm,0)),
       COUNT(deptno), COUNT(DISTINCT deptno)
FROM emp;
```

emp (1r x 6c)					
COUNT(*)	COUNT(comm)	AVG(comm)	AVG(NVL(comm,0))	COUNT(deptno)	COUNT(DISTINCT deptno)
14	4	550.0000	157.1429	14	3

- GROUP BY
 - 전체 테이블이 아닌 특정 그룹으로 묶을 때 사용. (예: 부서별, 직급별)

```
SELECT [ column ,] 그룹함수( column), ...  
FROM   table  
[WHERE 조건식]  
[GROUP BY column|별칭|컬럼순서]  
[ORDER BY column|별칭|컬럼순서];
```

GROUP BY 작성 지침

- SELECT 절 뒤에 사용할 수 있는 컬럼은 GROUP BY 뒤에 기술된 컬럼 또는 그룹함수가 적용된 컬럼 이어야 한다. (상수인 리터럴은 제외)
- WHERE 절을 사용하여 행을 그룹으로 분류하기 전에 제외 시킬 수 있다.
- HAVING절을 이용하여 GROUP BY 소그룹을 제외 시킬 수 있다.
- ORDER By와 동일하게 GROUP BY 절 뒤에도 컬럼 별칭 및 컬럼순서 정수값을 사용할 수 있다.
- WHERE 절에 그룹함수를 사용 할 수 없다. (그룹함수를 사용할 수 있는 GROUP By절보다 WHERE절이 먼저 수행된다.)

샘플) EMP 테이블에서 부서별로 인원수, 평균 급여, 최저급여, 최고 급여, 급여의 합을 구하여 출력?

```
SELECT deptno, COUNT(*), ROUND(AVG(sal)), MIN(sal), MAX(sal),  
SUM(sal)  
FROM emp  
GROUP BY deptno;
```

샘플) 각 부서별로 인원수, 급여의 평균, 최저 급여, 최고 급여, 급여의 합을 구하여 급여의 합이 많은 순으로 출력?

```
SELECT deptno, COUNT(*), ROUND(AVG(sal)), MIN(sal), MAX(sal),  
SUM(sal)  
FROM emp  
GROUP BY deptno  
ORDER BY SUM(sal) DESC;
```

샘플) 부서별, 업무별 그룹화하여 결과를 부서번호, 업무, 인원수, 급여의 평균, 급여의 합을 구하여 출력?

```
SELECT deptno, job, COUNT(*), ROUND(AVG(sal)), SUM(sal)
FROM emp
GROUP BY deptno, job;
```

■ HAVING 절

: GROUP BY에 의해 분류된 그룹들을 제한하기 위한 방법.

5. SELECT [column ,] group_function(column), ...
1. FROM table
2. [WHERE 조건식]
3. [GROUP BY column]
4. [HAVING group_조건식]
6. [ORDER BY column];

```
SELECT deptno, SUM(sal)
FROM emp
GROUP BY deptno
HAVING SUM(sal) > 7000;
```

emp (3r × 2c)		
deptno		SUM(sal)
10		8,750
20		10,875
30		9,400

```
SELECT deptno, SUM(sal)
FROM emp
WHERE sal > 1500
GROUP BY deptno
HAVING SUM(sal) > 7000;
```

emp (2r × 2c)		
deptno		SUM(sal)
10		7,450
20		8,975

샘플) EMP 테이블에서 급여가 최대 2900 이상인 부서에 대해서 부서번호, 평균 급여, 급여의 합을 구하여 출력?

```
SELECT deptno, ROUND(AVG(sal)), SUM(sal)
FROM emp
GROUP BY deptno
HAVING MAX(sal) > 2900;
```

샘플) EMP 테이블에서 업무별 급여의 평균이 2000 이상인 업무에 대해서 업무명, 평균 급여, 급여의 합을 구하여 출력?

```
SELECT job, ROUND(AVG(sal)), SUM(sal)
FROM emp
GROUP BY job
HAVING AVG(sal) > 2000;
```

샘플) EMP 테이블에서 전체 월급이 5000을 초과하는 각 업무에 대해서 업무와 월급 합계를 출력하여라.

단 판매원은 제외하고 월 급여 합계로 내림차순 정렬한다.

```
SELECT job, SUM(sal)
FROM emp
WHERE job != 'SALESMAN'
GROUP BY job
HAVING SUM(sal) > 5000
ORDER BY SUM(sal) DESC;
```



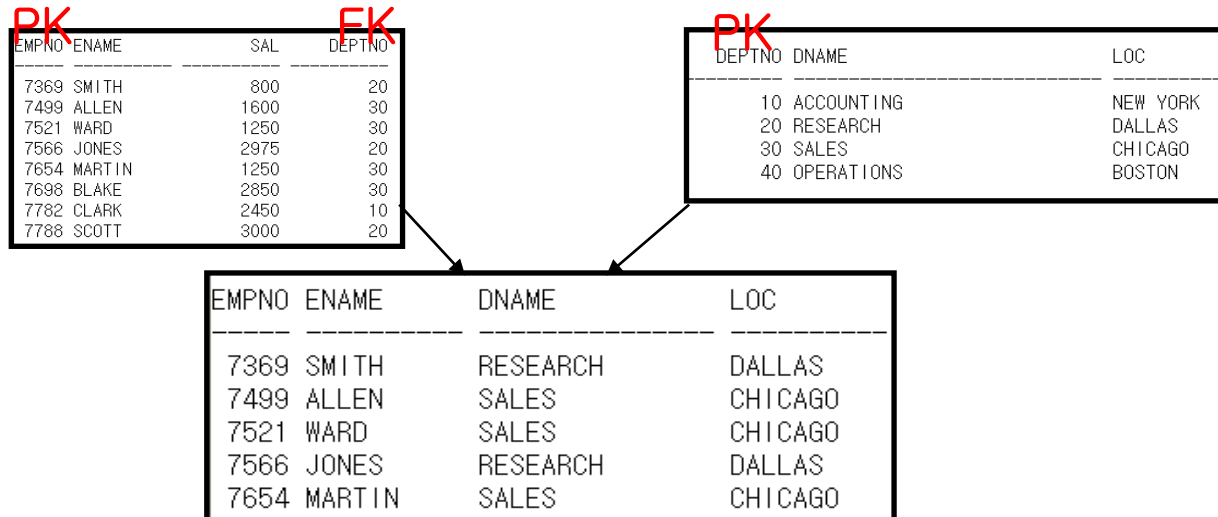
chapter 05.

조인

https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/join.html

JOIN 정의

- 검색하고자 하는 컬럼이 한 개의 테이블이 아닌 여러 개의 테이블에 존재하는 경우에 사용되는 기술.
- 일반적으로 Primary Key(PK 로 사용)와 Foreign Key(FK 로 사용)를 사용하여 Join 하는 경우가 대부분이지만 때로는 논리적인 값들의 연관으로 Join 하는 경우도 있다.



■ JOIN 종류

ANSI 조인

- Cross 조인
- Natural 조인
- Using 절 이용한 조인
- On 절 이용한 조인
- Outer 조인 (left|right) # MariaDB에서는 full 지원안됨
- self 조인

■ ANSI 조인 특징

- Join의 형식이 FROM 절에서 지정된다.
- Join 조건이 WHERE 절이 아닌 ON절에서 명시된다.
- USING절에서는 공통 컬럼명에 alias 사용 못함.

ANSI 조인 기본 문법

```
SELECT table1.column, table2.column
FROM table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)];
```

- Cross Join
 - 두 개의 테이블의 레코드를 각각 cross해서 조인한다.
 - 실제 데이터로서 사용은 불가

```
SELECT empno, ename, dname  
FROM emp CROSS JOIN dept;
```

결과 #1 (56r, 3c)			
empno	ename	dname	
7,369	SMITH	ACCOUNTING	
7,369	SMITH	RESEARCH	
7,369	SMITH	SALES	
7,369	SMITH	OPERATIONS	
7,499	ALLEN	ACCOUNTING	

- Natural Join
 - Natural Join을 사용할 때의 전제조건은 양쪽 Table에 반드시 1개의 공통 컬럼이 있어야 한다.
 - 2 개 이상의 공통 컬럼이 있다면 오류는 아니지만 엉뚱한 결과를 가져온다. 즉, 2개의 공통 컬럼 값이 서로 같은 것만 추출해 온다.

```
SELECT empno, ename, dname  
FROM emp NATURAL JOIN dept;
```

```
SELECT empno, ename, dname, deptno  
FROM emp e NATURAL JOIN dept d;
```

```
SELECT e.empno, e.ename, d.dname, e.deptno  
FROM emp e NATURAL JOIN dept d;
```

■ USING(컬럼)

- 명시적으로 조인으로 사용될 컬럼을 지정하는 방식이다.
여기서 컬럼명은 조회할 자료가 있는 테이블에 공통된 컬럼명이다.
- USING 절에서는 컬럼명에 Table Alias 명을 사용하지 못한다.

```
SELECT empno, ename, dname  
FROM emp JOIN dept USING(deptno);
```

```
SELECT e.empno, e.ename, d.dname, e.deptno  
FROM emp e JOIN dept d USING(deptno);
```

예러

```
SELECT e.empno, e.ename, d.dname, e.deptno  
FROM emp e JOIN dept d USING(e.deptno);
```

- join~on 이용한 Join
 - 임의의 조건으로 조인시 사용한다.
(서브쿼리 , AND/OR 연산자 , EXIST , IN 연산자)
 - 검색 조건 목적인 경우는 WHERE절을 사용한다.

```
SELECT empno, ename, dname, loc , e.deptno
FROM emp e JOIN dept d ON e.deptno = d.deptno;
```

```
SELECT empno, ename, dname, loc , sal, e.deptno
FROM emp e JOIN dept d ON e.deptno = d.deptno
WHERE sal > 2500;
```

```
SELECT empno, ename, dname, sal, grade
FROM emp e JOIN dept d ON e.deptno = d.deptno
      JOIN salgrade s ON e.sal BETWEEN s.LOSAL AND s.HISAL;
```

```
SELECT empno, ename, dname, sal, grade
FROM emp e JOIN (dept d, salgrade s) ON (e.deptno = d.deptno AND e.sal
BETWEEN s.LOSAL AND s.HISAL);
```

```
SELECT e.ename 사원, m.ename 관리자
FROM emp e JOIN emp m ON e.mgr = m.empno;
```

- LEFT OUTER JOIN ~ ON | USING

- 좌측에 기술한 테이블의 모든 행들이 우측에 기술한 테이블내 행들과 일치 여부에 상관없이 모두 출력된다.

- RIGHT OUTER JOIN ~ ON | USING

- 우측에 기술한 테이블의 모든 행들이 좌측에 기술한 테이블내 행들과 일치 여부에 상관없이 모두 출력된다.

누락된 40 부서번호 포함하여 출력

```
SELECT empno, ename, dname, d.deptno  
FROM emp e RIGHT OUTER JOIN dept d ON e.deptno = d.deptno;
```

누락된 신입사원 포함하여 출력

```
# 신입사원  
INSERT INTO emp VALUES ( 9000, 'TEST', 'SALESMAN', 7499,  
'90/01/01', 600, NULL, NULL );  
COMMIT;
```

```
SELECT empno, ename, dname, d.deptno  
FROM emp e LEFT OUTER JOIN dept d ON e.deptno = d.deptno;
```


■ 실습 문제

1. 부서 테이블과 사원테이블에서 사번, 사원명, 부서코드, 부서명을 검색 하시오.
(사원명 오름차순 정렬할 것)
2. 부서 테이블과 사원테이블에서 사번, 사원명 , 급여 , 부서명을 검색 하시오. 단, 급여가 2000 이상인 사원에 대하여 급여기준으로 내림차순 정렬할 것.
3. 부서 테이블과 사원 테이블에서 사번, 사원명, 업무, 급여 , 부서명을 검색 하시오. 단, 업무가 Manager이며 급여가 2500 이상인 사원에 대하여 사번을 기준으로 오름차순 정렬할 것.
4. 사원 테이블과 급여 등급 테이블에서 사번, 사원명, 급여, 등급을 검색 하시오. 단, 등급은 급여가 하한값과 상한값 범위에 포함되고 등급이 4이며 급여를 기준으로 내림차순 정렬할 것.
5. 부서 테이블, 사원 테이블, 급여등급 테이블에서 사번, 사원명, 부서명, 급여 , 등급을 검색 하시오. 단, 등급은 급여가 하한값과 상한값 범위에 포함되며 등급을 기준으로 내림차순 정렬할 것.
6. 사원 테이블에서 사원명과 해당 사원의 관리자명을 검색 하시오.
7. 사원 테이블에서 사원명, 해당 사원의 관리자명, 해당 사원의 관리자의 관리자명을 검색 하시오
8. 7번 결과에서 상위 관리자가 없는 모든 사원의 이름도 사원명에 출력되도록 수정 하시오.
9. 20번 부서의 이름과 그 부서에 근무하는 사원의 이름을 출력 하시오.

10. 커미션을 받는 사원의 이름, 커미션, 부서이름을 출력 하시오.
11. 이름에 'A' 가 들어가는 사원들의 이름과 부서명을 출력 하시오.
12. DALLAS에 근무하는 사원 중 급여 1500 이상인 사원의 이름, 급여, 입사일을 출력 하시오.
13. 자신의 관리자 보다 연봉(sal)을 많이 받는 사원의 이름과 연봉을 출력 하시오.
14. 직원 중 현재시간 기준으로 근무 개월 수가 30년보다 많은 사람의 이름, 급여, 입사일, 부서명을 출력 하시오.
15. 각 부서별로 1982년 이전에 입사한 직원들의 부서명, 인원수를 출력 하시오.



chapter 06.

서브쿼리

https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/subqueries.html

■ 서브쿼리 정의

- SELECT 문 (Mainquery) 에 포함되어 있는 별도 SELECT 문 (Subquery)이다.
- 여러 번의 SELECT문을 수행해야 얻을 수 있는 결과를 하나의 중첩된 SELECT 문으로 쉽게 얻을 수 있도록 해준다.

샘플) emp 테이블에서 SCOTT 사원보다 많은 급여를 받는 사원의 이름을 검색 하시오.

```
SQL> SELECT SAL
  2  FROM EMP
  3  WHERE ENAME = 'SCOTT';

      SAL
-----
      3000

SQL> SELECT ENAME
  2  FROM EMP
  3  WHERE SAL > 3000;

ENAME
-----
KING
```



```
SQL> SELECT ENAME
  2  FROM EMP
  3  WHERE SAL > ( SELECT SAL
  4                  FROM EMP
  5                  WHERE ENAME = 'SCOTT' );

ENAME
-----
KING
```

■ 서브쿼리 사용방법

```
SELECT select_list
FROM table
WHERE expr operator
      (SELECT select_list
       FROM table);
```

- 바깥 쪽 쿼리를 Main query , 안쪽 쿼리를 Subquery 라고 한다.
- 서브쿼리가 먼저 실행되고, 그 결과가 메인 쿼리에 전달되어 실행된다.
따라서 메인 쿼리에서 서브 쿼리 컬럼을 사용할 수 없다.
- 서브쿼리는 SELECT, FROM, WHERE, HAVING, ORDER BY, UPDATE, INSERT INTO절
에도 사용될 수 있다.
- 연산자는 단일 행 연산자 (> , = , >= , < , <= , !=)와 복수 행 연산자
(IN , ANY , ALL, EXISTS)를 사용 할 수 있다.
- SQL 문장의 서브쿼리가 단일 행 서브쿼리인 경우는 단일 행 연산자를 사용하고 복수행
서브쿼리인 경우에는 복수 행 연산자를 사용해야 한다.
- 서브쿼리에는 반드시 괄호를 사용한다.
- 서브쿼리에는 ORDER BY 절을 사용불가

■ 단일행 서브쿼리

- 서브쿼리가 한 개의 행을 리턴.
- 반드시 단일행 연산자를 사용해야 한다. (= , > , < , >= , <= , !=)

샘플) EMP 테이블에서 SCOTT의 급여보다 많은 사원의 사원번호, 이름, 담당업무, 급여를 출력 ?

```
SELECT empno, ename, job, sal
FROM emp
WHERE sal > ( SELECT sal
               FROM emp
               WHERE ename = 'scott' );
```

emp (1r x 4c)			
empno	ename	job	sal
7,839	KING	PRESIDENT	5,000

샘플) EMP 테이블에서 사원번호가 7521의 업무와 같고 급여가 7934보다 많은 사원의 사원번호, 이름, 담당업무,입사일자, 급여를 출력 ?

```
SELECT empno, ename, job, sal
FROM emp
WHERE job = ( SELECT job
              FROM emp
              WHERE empno = 7521 )
AND
      sal > ( SELECT sal
              FROM emp
              WHERE empno = 7934 );
```

emp (2r x 4c)			
empno	ename	job	sal
7,499	ALLEN	SALESMAN	1,600
7,844	TURNER	SALESMAN	1,500

샘플) EMP 테이블에서 급여의 평균보다 적은 사원의 사원번호, 이름, 담당업무, 급여, 부서번호를 출력 ?

```
SELECT empno, ename, job, sal, deptno
FROM emp
WHERE sal < (SELECT AVG(sal)
             FROM emp );
```

empno	ename	job	sal	deptno
7,369	SMITH	CLERK	800	20
7,499	ALLEN	SALESMAN	1,600	30
7,521	WARD	SALESMAN	1,250	30
7,654	MARTIN	SALESMAN	1,250	30
7,844	TURNER	SALESMAN	1,500	30

샘플) EMP 테이블에서 20번 부서의 최소 급여보다 많은 모든 부서를 출력 ?

```
SELECT deptno, MIN(sal)
FROM emp
GROUP BY deptno
HAVING MIN(sal) > ( SELECT MIN(sal)
                   FROM emp
                   WHERE deptno = 20 );
```

deptno	MIN(sal)
10	1,300
30	950

■ 복수행 서브쿼리

- 서브쿼리가 여러 개의 행을 리턴.
- 반드시 복수행 연산자를 사용해야 한다. (IN , ANY ,ALL ,EXISTS)

■ IN 연산자

- WHERE 절에서 사용하는 일반 비교연산자와 동일하다.
- 메인쿼리의 비교 조건이 서브쿼리의 결과 중에서 하나라도 일치하면 검색가능.

샘플) EMP 테이블에서 업무별로 최소 급여를 받는 사원의 사원번호, 이름, 업무, 입사일자, 급여, 부서번호를 출력 ?

```
SELECT empno, ename, job, hiredate, sal, deptno
FROM emp
WHERE sal IN ( SELECT MIN(sal)
                FROM emp
                GROUP BY job );
```

emp (6r x 6c)					
empno	ename	job	hiredate	sal	deptno
7,369	SMITH	CLERK	1980-12-17	800	20
7,782	CLARK	MANAGER	1981-06-09	2,450	10
7,788	SCOTT	ANALYST	1987-06-28	3,000	20
7,839	KING	PRESIDENT	1981-11-17	5,000	10
7,902	FORD	ANALYST	1981-12-03	3,000	20
9,000	TEST	SALESMAN	1990-01-01	600	(NULL)

▪ ALL 연산자

- 복수행 서브쿼리 결과가 메인 쿼리의 WHERE 절에서 부등호 조건으로 비교할 때 사용된다. (원래 부등호 조건은 단일행에서 사용됨)
- 서브쿼리에서 반환되는 행들 전체에 대해 모두 조건을 만족해야 된다.

샘플) 사원 테이블에서 업무가 MANAGER인 사원의 최소급여보다 적은 급여를 받는 사원들의 이름 검색?

```
SQL> SELECT SAL  
2 FROM EMP  
3 WHERE JOB = 'MANAGER';
```

SAL
2975
2850
2450

최소값보다 작은

```
SQL> SELECT EMPNO, ENAME, SAL  
2 FROM EMP  
3 WHERE SAL < 2450;
```

EMPNO	ENAME	SAL
7369	SMITH	800
7499	ALLEN	1600
7521	WARD	1250
7654	MARTIN	1250
7844	TURNER	1500
7876	ADAMS	1100
7900	JAMES	950
7934	MILLER	1300

8 개의 행이 선택되었습니다.

```
SQL> SELECT EMPNO, ENAME, SAL  
2 FROM EMP  
3 WHERE SAL < ALL (SELECT SAL  
4 FROM EMP  
5 WHERE JOB = 'MANAGER' );
```

EMPNO	ENAME	SAL
7369	SMITH	800
7499	ALLEN	1600
7521	WARD	1250
7654	MARTIN	1250
7844	TURNER	1500
7876	ADAMS	1100
7900	JAMES	950
7934	MILLER	1300

8 개의 행이 선택되었습니다.

■ ANY 연산자

- 복수행 서브쿼리 결과가 메인 쿼리의 WHERE 절에서 부등호 조건으로 비교할 때 사용된다.
- 서브쿼리에서 반환되는 행들 중에서 어느 하나만 만족해도 된다.

샘플)사원 테이블에서 업무가 MANAGER인 사원의 최소급여보다 많은 급여를 받는 사원들의 이름 검색?

최소값보다 큰

```
SQL> SELECT SAL
2 FROM EMP
3 WHERE JOB = 'MANAGER';
```

SAL
2975
2850
2450

```
SQL> SELECT EMPNO, ENAME, SAL
2 FROM EMP
3 WHERE SAL > 2450;
```

EMPNO	ENAME	SAL
7566	JONES	2975
7698	BLAKE	2850
7788	SCOTT	3000
7839	KING	5000
7902	FORD	3000



```
SQL> SELECT EMPNO, ENAME, SAL
2 FROM EMP
3 WHERE SAL > ANY ( SELECT SAL
4 FROM EMP
5 WHERE JOB = 'MANAGER');
```

EMPNO	ENAME	SAL
7839	KING	5000
7902	FORD	3000
7788	SCOTT	3000
7566	JONES	2975
7698	BLAKE	2850

▪ EXISTS 연산자

서브쿼리에서 검색된 결과가 하나라도 존재하는지 여부를 확인할 때 사용된다.
만일 서브쿼리에서 검색된 결과가 하나도 없으면 메인 쿼리의 조건절이
거짓이 메인 쿼리가 실행되지 않고 결과가 하나라도 있으면 메인 쿼리가 실행된다.

샘플) 만일 사원중에서 comm을 받는 사원이 한 명이라도 있으면 모든 사원출력?

```
SELECT empno, ename, job, hiredate, sal, deptno
FROM emp
WHERE EXISTS ( SELECT empno
                FROM emp
                WHERE comm IS NOT NULL );
```

▪ 다중 컬럼 서브쿼리

- 서브쿼리에서 여러개의 컬럼값을 검색하여 메인쿼리의 조건절과 비교하는 서브쿼리이다. 메인쿼리의 조건절에서도 서브쿼리의 컬럼수만큼 지정해야 된다.
- 컬럼을 쌍으로 묶어서 동시에 비교한다.

1	2
1	3
4	2
3	2

1	2
4	3

샘플) 부서별로 가장 많은 sal을 받는 사원 정보 출력 ?

```
SELECT deptno, empno, ename, sal
FROM emp
WHERE (deptno, sal) IN ( SELECT deptno, MAX(sal)
                        FROM emp
                        GROUP BY deptno);
```

emp (4r x 4c)				
deptno	empno	ename	sal	
30	7,698	BLAKE	2,850	
20	7,788	SCOTT	3,000	
10	7,839	KING	5,000	
20	7,902	FORD	3,000	

from절 뒤에 테이블 명이 나와야 되지만, 서브 쿼리가 하나의 가상 테이블을 반환하는 형태로 사용되는 경우를 의미한다.

```
SELECT column_list
FROM (subquery) alias
WHERE condition;
```

샘플> emp 와 dept 테이블에서 부서별 sal 총합과 평균을 출력?

```
SELECT e.deptno, total_sum, total_avg, cnt
FROM ( SELECT deptno, SUM(sal) total_sum, ROUND(AVG(sal)) total_avg,
           COUNT(*) cnt
       FROM emp
       GROUP BY deptno) e JOIN dept d ON e.deptno = d.deptno;
```

emp (3r × 4c)			
deptno	total_sum	total_avg	cnt
10	8,750	2,917	3
20	10,875	2,175	5
30	9,400	1,567	6

■ 실습 문제

1. 사원 테이블에서 BLAKE 보다 급여가 많은 사원들의 사번, 이름, 급여를 검색 하시오.
2. 사원 테이블에서 MILLER 보다 늦게 입사한 사원의 사번, 이름, 입사일을 검색 하시오.
3. 사원 테이블에서 사원 전체 평균 급여보다 급여가 많은 사원들의 사번, 이름, 급여를 검색 하시오.
4. 사원 테이블에서 부서별 최대 급여를 받는 사원들의 사번, 이름, 부서코드, 급여를 검색 하시오.
5. Salgrade가 2등급인 사원들의 평균 급여보다 적게 받는 사원 정보를 검색 하시오.



chapter **07.**

DML

https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/sql-data-manipulation-statements.html

■ DML 용도 및 종류

13.2 Data Manipulation Statements

[13.2.1 CALL Statement](#)

[13.2.2 DELETE Statement](#)

[13.2.3 DO Statement](#)

[13.2.4 EXCEPT Clause](#)

[13.2.5 HANDLER Statement](#)

[13.2.6 IMPORT TABLE Statement](#)

[13.2.7 INSERT Statement](#)

[13.2.8 INTERSECT Clause](#)

[13.2.9 LOAD DATA Statement](#)

[13.2.10 LOAD XML Statement](#)

[13.2.11 Parenthesized Query Expressions](#)

[13.2.12 REPLACE Statement](#)

[13.2.13 SELECT Statement](#)

[13.2.14 Set Operations with UNION, INTERSECT, and EXCEPT](#)

[13.2.15 Subqueries](#)

[13.2.16 TABLE Statement](#)

[13.2.17 UPDATE Statement](#)

[13.2.18 UNION Clause](#)

[13.2.19 VALUES Statement](#)

[13.2.20 WITH \(Common Table Expressions\)](#)

[Previous](#)

[Home](#)

[Up](#)

[Next](#)

[MySQL Documentation Library](#)

■ INSERT 문

- 테이블에 데이터를 입력하기 위한 데이터 조작용어 이다.
- 한번에 하나의 행을 테이블에 입력하는 방법과 서브쿼리를 이용하여 한번에 여러 행을 동시에 입력하는 방법이 있다.

```
INSERT INTO table [(column [, column ...])]  
VALUES (value [, value ...])
```

- INTO 절에 명시한 컬럼에 VALUES 절에서 지정한 컬럼값을 입력한다.
(일대일 대응)
- INTO 절에 컬럼명을 지정하지 않으면 테이블 생성시 정의한 컬럼순서와 동일한 순서로 입력된다.
- 입력되는 데이터의 타입은 컬럼의 데이터 타입과 같아야 되며 입력되는 데이터의 크기는 컬럼의 크기보다 작아야 된다.

```
DESC dept;
```

COLUMNS (3r x 6c)					
Field	Type	Null	Key	Default	Extra
DEPTNO	int(11)	NO	PRI	(NULL)	
DNAME	varchar(14)	YES		(NULL)	
LOC	varchar(13)	YES		(NULL)	

컬럼명 지정한 경우

```
INSERT INTO dept (deptno, dname, loc )  
VALUES (90, '인사과', '서울');
```

```
INSERT INTO dept (deptno, dname ) # loc 컬럼에 null 저장  
VALUES (91, '인사과');
```

```
INSERT INTO dept (loc, dname, deptno )  
VALUES ('서울', '인사과', 80);
```

컬럼명 지정하지 않은 경우

```
INSERT INTO dept  
VALUES (70, '인사과', '서울');
```

- 다중 행 입력

- 서브쿼리 절을 이용하여 하나의 INSERT 명령문에 여러 행을 동시에 저장 가능하다.

```
# CTAS 이용한 테이블 생성
CREATE TABLE copy_emp
AS
SELECT empno, ename, sal FROM emp
WHERE 1=2;
```

```
# 서브쿼리 이용한 다중 데이터 저장
INSERT INTO copy_emp (empno, ename, sal)
SELECT empno, ename, sal
FROM emp;
```

▪ UPDATE 용도

- 테이블에 저장된 행들을 변경하는 문장이다.
- 한 번에 여러 개의 행들을 변경할 수 있다.

```
UPDATE table  
SET column = value [, column = value, ... ]  
[WHERE condition];
```

```
UPDATE dept  
SET dname = '경리과', loc='부산'  
WHERE deptno = 90;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
70	인사과	서울
80	인사과	서울
90	인사과	서울
91	인사과	(NULL)



DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
70	인사과	서울
80	인사과	서울
90	경리과	부산
91	인사과	(NULL)

- 서브쿼리를 이용한 복수 컬럼 변경

```
UPDATE dept
SET  dname = (SELECT dname
               FROM dept
               WHERE deptno = 20 ),
     loc   = (SELECT loc
               FROM dept
               WHERE deptno = 30 )
WHERE deptno = 90;
```

- DELETE 용도
 - 테이블에 저장된 행들을 삭제한다.
 - 한 번에 여러 개의 행들을 삭제할 수 있다.

```
DELETE [FROM] table  
[WHERE condition];
```

```
DELETE FROM dept  
WHERE deptno = 90;
```

- 서브쿼리를 이용한 삭제

```
DELETE FROM dept  
WHERE deptno IN ( SELECT deptno  
                  FROM dept  
                  WHERE dname = '인사과' );
```

▪ MERGE 기능

- 대상 테이블에 해당 행이 이미 존재하면 UPDATE 가 실행되고 존재하지 않으면 INSERT가 실행된다.

13.2.7.2 INSERT ... ON DUPLICATE KEY UPDATE Statement

If you specify an `ON DUPLICATE KEY UPDATE` clause and a row to be inserted would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row occurs. For example, if column `a` is declared as `UNIQUE` and contains the value `1`, the following two statements have similar effect:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE c=c+1;
```

```
UPDATE t1 SET c=c+1 WHERE a=1;
```

```
INSERT INTO dept (deptno, dname, loc )
VALUES (90, '인사과', '서울');
```

```
INSERT INTO dept (deptno, dname, loc )
VALUES (90, '인사과', '서울') ON DUPLICATE KEY UPDATE loc='제주';
```


https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/sql-transactional-statements.htm

■ Transaction 정의

- 트랜잭션은 데이터베이스의 논리적인 단위이다.
- 트랜잭션은 밀접히 관련되어 분리될 수 없는 한 개 이상의 데이터베이스 조작을 가리킨다.
- 하나의 트랜잭션에는 하나 이상의 SQL문장이 포함되며 분할 할 수 없는 최소의 단위이다. 그렇기 때문에 전부 적용하거나 전부 취소된다.
즉, 트랜잭션은 'All or Nothing'이다.
- 트랜잭션의 대상이 되는 SQL문은 DML문이다.

■ COMMIT

- 모든 데이터 변경사항을 데이터베이스에 영구히 반영시키는 명령어.
- 변경전의 데이터는 모든 잃게 된다.
- 모든 사용자들이 트랜잭션 종료 후의 결과를 확인 할 수 있다.
- 트랜잭션이 진행 중이었던 행들에 대한 잠금이 모두 해소되며, 다른 사용자에게 의해서 변경이 가능해진다.
- MariaDB는 기본적으로 autocommit 으로 설정되어 있다.
set autocommit = true|false 지정 가능

```
# autocommit 설정값 확인
show variables like 'autocommit%';
#autocommit 설정 또는 해제
SET AUTOCOMMIT = TRUE;      -- 설정
SET AUTOCOMMIT = FALSE;    -- 해제
```

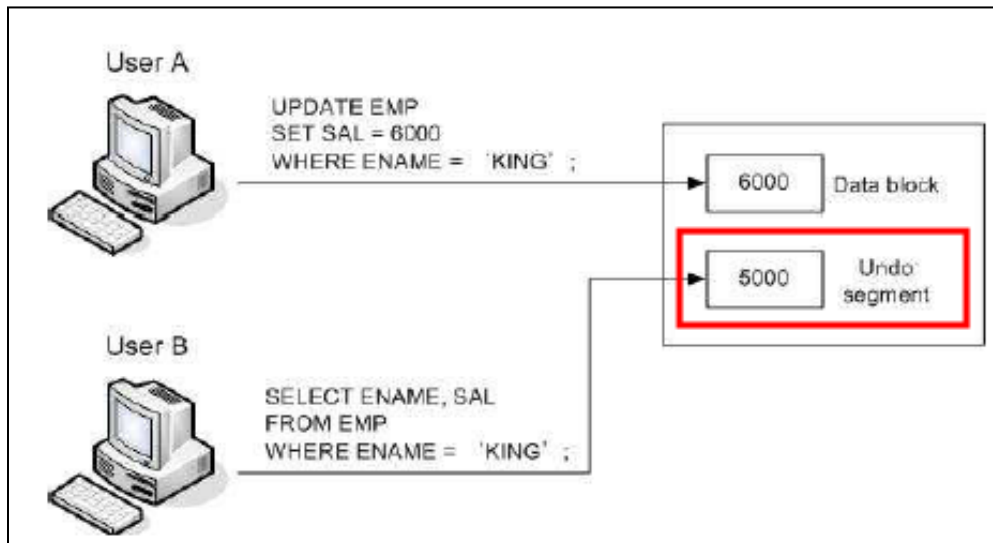
■ ROLLBACK

- 모든 데이터 변경사항을 취소하는 명령어.
- 변경전의 데이터가 복원된다.
- 모든 사용자들이 트랜잭션 종료 후의 결과를 확인 할 수 있다.
- 트랜잭션이 진행 중이었던 행들에 대한 잠금이 모두 해소되며, 다른 사용자에게 의해서 변경이 가능해진다.

■ 읽기 일관성 (Read Consistency)

- 사용자들에게 가장 최근에 commit 된 데이터를 보여주는 것.
- 데이터를 검색하는 사용자와 변경하는 사용자들 사이에 일관적인 관점을 제공한다.
즉, 다른 사용자들이 변경중인 데이터를 볼 수 없게 한다.
- 일관적인 데이터베이스 변경방법을 제공함으로써 동일한 데이터를 동시에 변경할 때 발생할 수 있는 혼란을 방지한다.

■ 읽기 일관성 구현 원리



LOCK 경합

① SQL> SELECT * FROM DEPT;

DEPTNO	DNAME	LOC
90	경리과	부산
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

② SQL> UPDATE DEPT
2 SET DNAME = '인사과'
3 WHERE DEPTNO = 90;

1 행이 갱신되었습니다.

③ SQL> SELECT * FROM DEPT;

DEPTNO	DNAME	LOC
90	인사과	부산
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

⑥ SQL> ROLLBACK;

```
C:\WINDOWS\System32>mysql -u root -h 127.0.0.1 -p
Enter password: ****
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 1316
Server version: 10.11.1-MariaDB mariadb.org binary distribution
```

```
MariaDB [(none)]> show databases
-> ;
+-----+
| Database |
+-----+
| information_schema |
| inky      |
| ktds     |
| mysql    |
| performance_schema |
| sys      |
+-----+
6 rows in set (0.001 sec)

MariaDB [(none)]> use inky;
Database changed
MariaDB [inky]> select * from emp;
```

④ SQL> SELECT * FROM DEPT;

DEPTNO	DNAME	LOC
90	경리과	부산
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

⑤ SQL> UPDATE DEPT
2 SET LOC = '서울'
3 WHERE DEPTNO = 90;

Wait!!

chapter 08.

DDL

https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/sql-data-definition-statements.html

■ 테이블 생성

```
CREATE TABLE [IF NOT EXISTS] table명  
( column명 datatype [DEFAULT 값] [AUTO_INCREMENT] [제약조건], ... )
```

```
CREATE TABLE IF NOT EXISTS dept_2  
( deptno INT,  
  dname VARCHAR(10),  
  loc VARCHAR(10) );
```

■ DEFAULT 옵션

- 해당 테이블에 행을 입력할 때 해당 컬럼에 값을 지정하지 않은 경우 자동으로 기본값이 입력되어 null 값이 저장되는 것을 방지할 수 있다.

```
CREATE TABLE IF NOT EXISTS dept_3
( deptno INT,
  dname VARCHAR(10),
  loc VARCHAR(10) DEFAULT '서울' );
```

```
CREATE TABLE IF NOT EXISTS board
( num INT PRIMARY KEY AUTO_INCREMENT,
  author VARCHAR(10) NOT NULL,
  title VARCHAR(20) NOT NULL,
  content VARCHAR(100) NOT NULL,
  writeday DATE DEFAULT NOW(),
  readcnt INT DEFAULT 0 );
```

■ 제약조건 정의 (Constraints)

- 부적절한 자료가 입력되는 것을 방지하기 위하여 constraint을 사용한다.
- 제약 조건은 종속성이 존재할 경우 테이블 삭제를 방지 한다.
- 테이블에서 행이 삽입, 갱신, 삭제될 때마다 테이블에서 규칙을 적용한다.
- 제약 조건은 테이블 레벨 및 컬럼 레벨 방법으로 규칙을 적용한다.

■ 제약조건 종류

제약조건 타입	설명
PRIMARY KEY	테이블의 행(레코드) 식별용. NOT NULL + UNIQUE
UNIQUE (KEY)	유일한 값 저장. null 허용.
NOT NULL	반드시 값을 저장. null 허용 안함.
CHECK	특정 조건 지정. 예> age > 20
FOREIGN KEY	컬럼과 참조된 테이블의 컬럼 사이의 외래키 관계설정

■ 제약조건 정의 방법

1. 컬럼 LEVEL 지정 방법

- 한 개의 컬럼에 한 개의 제약조건만 정의 가능하다.
- 모든 제약조건에 대해서 정의 가능하다. (NOT NULL 제약조건 필수)

```
CREATE TABLE [IF NOT EXISTS] table명  
( column명 datatype,  
  column명 datatype, ...  
  CONSTRAINT constraint_type, ... )
```

2. 테이블 LEVEL 지정 방법

- 테이블의 컬럼 정의와는 별개로 정의한다.
- 한 개 이상의 컬럼에 한 개의 제약 조건을 정의할 수 있다.
- NOT NULL 제약조건을 제외한 모든 제약조건 정의 가능하다.

```
CREATE TABLE [IF NOT EXISTS] table명  
( column명 datatype constraint_type,  
  column명 datatype constraint_type, ... )
```

■ PRIMARY KEY

- 하나의 기본 키만이 각 테이블에 대해 존재할 수 있다.
- PRIMARY KEY 제약 조건은 테이블에서 각행을 유일하게 식별하는 열 또는 열의 집합(복합컬럼)이다.(UNIQUE와 NOT NULL조건을 만족)
- 이 제약 조건은 열 또는 열의 집합의 유일성을 요구하고 NULL값을 포함할 수 없음을 보증 한다.

■ UNIQUE

- 해당 컬럼에 중복된 값이 저장되지 않도록 제한한다.
- 한 개 이상의 컬럼(복합컬럼)으로 구성 할 수 있다.
- NULL 값 저장 가능하다.

■ NOT NULL

- 해당 컬럼에 NULL 값이 입력되는 것을 방지하는 제약조건이다.
- NOT NULL 제약조건은 컬럼 레벨에서만 지정 가능하다.

■ CHECK

- 해당 컬럼에 반드시 만족해야 될 조건을 지정하는 제약 조건이다. (회사의 업무규칙등)

컬럼 레벨 방식

```
CREATE TABLE student
( no INT PRIMARY KEY,
  name VARCHAR(10) UNIQUE ,
  address VARCHAR(10) NOT NULL, # 컬럼레벨만 가능
  age INT CHECK( age IN ( 10,20,30 ) ) );
```

테이블 레벨 방식

```
CREATE TABLE student2
( no INT,
  name VARCHAR(10),
  address VARCHAR(10) NOT NULL,
  age INT ,
  CONSTRAINT PRIMARY KEY (no) ,
  CONSTRAINT UNIQUE (name) ,
  CONSTRAINT CHECK ( age IN ( 10,20,30 ) ) );
```

■ FOREIGN KEY

- FOREIGN KEY는 child 테이블에서 정의한다.
- master 테이블의 PRIMARY KEY, UNIQUE KEY로 정의된 열을 참조 할 수 있으며 참조된 열의 값 및 NULL값만 저장할 수 있다.
- FOREIGN KEY는 열 또는 열의 집합을 지정할 수 있으며 동일 테이블 또는 다른 테이블 간의 관계를 지정할 수 있다.

```
CREATE TABLE MASTER1
( NO INT PRIMARY KEY ,
  NAME VARCHAR(10) NOT NULL );
```

컬럼 레벨 방식

```
CREATE TABLE SLAVE1
( num INT PRIMARY KEY,
  NO INT REFERENCES master1(NO) );
```

테이블 레벨 방식

```
CREATE TABLE SLAVE2
( num INT PRIMARY KEY,
  NO INT,
  CONSTRAINT FOREIGN KEY (NO) REFERENCES master1(NO) );
```

- FOREIGN KEY 추가 옵션
 - 부모 테이블의 행 삭제 시 문제될 수 있는 자식 테이블 행 설정법

ON DELETE CASCADE

- FK 제약조건에 의해 참조되는 테이블(부모 테이블)의 행이 삭제되면, 해당 행을 참조하는 테이블(자식 테이블)의 행도 같이 삭제 되도록 한다.

ON DELETE SET NULL

- FK 제약조건에 의해 참조되는 테이블(부모 테이블)의 행이 삭제되면, 해당 행을 참조하는 테이블(자식 테이블)의 컬럼을 NULL 로 설정한다.

테이블 레벨 방식

```
CREATE TABLE SLAVE2
( num INT PRIMARY KEY,
  NO INT,
  CONSTRAINT FOREIGN KEY(NO) REFERENCES master1(NO) ON DELETE CASCADE );
```

■ 테이블 삭제

- 데이터베이스에서 해당 테이블을 제거하는 것이다.
- 테이블에 저장된 모든 데이터와 관련 INDEX 및 제약조건이 삭제된다.
- 만약 foreign key 제약조건이 있으면 테이블 삭제 안됨. CASCADE 지원 안됨.

```
DROP TABLE [IF EXISTS] table명, ...
```

■ 테이블 잘라내기

- 테이블의 모든 행들을 삭제 할 수 있다.
- TRUNCATE은 저장 공간을 해제하고 DELETE 명령은 해제하지 않는다.
- ROLLBACK 정보를 발생시키지 않아서 DELETE 보다 수행속도가 빠르다.
단, DELETE와 달리 ROLLBACK 은 불가능하다.

```
TRUNCATE TABLE table명
```

■ 테이블 변경 (ALTER TABLE 문)

- 새로운 컬럼 추가
- 기존 컬럼 변경
- 컬럼 이름 변경
- 컬럼 삭제
- 제약조건 추가, 삭제
- 제약조건 활성화, 비활성화
- 테이블 읽기 모드(read only)

■ 컬럼 추가

```
ALTER TABLE 테이블명  
ADD [COLUMN] [IF NOT EXISTS] (column datatype , ... );
```

```
CREATE TABLE scott_t  
( num INT,  
  NAME VARCHAR(10) );  
  
DESC scott_t;  
  
ALTER TABLE scott_t  
ADD ( address VARCHAR(30));
```

■ 기존 컬럼 변경

- 숫자 및 문자 컬럼의 전체 길이를 증가 또는 축소 시킬 수 있다.
단, 모든 행의 컬럼이 NULL 또는 행이 없는 경우
- 모든 행의 해당 컬럼 값이 NULL인 경우에만 데이터 타입을 변경 할 수 있다.
- 디폴트 값을 변경하면 변경 이후부터 입력되는 행에 대해서만 적용된다.

```
ALTER TABLE 테이블명  
MODIFY [COLUMN] [IF EXISTS] column datatype;
```

```
ALTER TABLE scott_t  
MODIFY address VARCHAR(5);
```

```
ALTER TABLE scott_t  
MODIFY address INT;
```

■ 컬럼 이름 변경

```
ALTER TABLE 테이블명  
RENAME COLUMN old_column TO new_column;
```

```
ALTER TABLE scott_t  
RENAME COLUMN address TO addr;
```

■ 컬럼 삭제

```
ALTER TABLE 테이블명  
DROP [COLUMN] [IF EXISTS] column ;
```

```
ALTER TABLE scott_t  
DROP address;
```


■ 제약조건 추가

- NOT NULL 제약조건은 ALTER TABLE ~ MODIFY 명령을 사용한다.
- 나머지 제약조건은 ALTER TABLE ~ ADD 명령을 사용한다.

ALTER TABLE 테이블명
ADD CONSTRAINT 제약조건타입(컬럼);

```
CREATE TABLE scott_t2
( num INT,
  NAME VARCHAR(10),
  age INT,
  address VARCHAR(20) );
```

```
ALTER TABLE scott_t2
ADD CONSTRAINT PRIMARY KEY(num);

ALTER TABLE scott_t2
ADD CONSTRAINT UNIQUE(NAME);

ALTER TABLE scott_t2
ADD CONSTRAINT CHECK(age>20);

ALTER TABLE scott_t2
MODIFY address VARCHAR(20) NOT NULL;
```

```
CREATE TABLE IF NOT EXISTS scott_t2_child
( no INT PRIMARY KEY,
  num INT );
```

```
ALTER TABLE scott_t2_child
ADD CONSTRAINT FOREIGN KEY(num)
REFERENCES scott_t2(num);
```



Thank you
