



# Ch 06. SQL 기본

이것이 MariaDB다

# Contents

## ❖ 핵심 개념

- **SELECT문의 형식과 사용법**
- **책 전체에서 사용할 sqIDB 생성**
- **특정 조건을 조회하는 WHERE절**
- **ORDER BY절 및 LIMIT절**
- **GROUP BY 및 HAVING 그리고 집계 함수**
- **INSERT/UPDATE/DELETE문의 형식**



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT...FROM>

- **SELECT의 구문 형식**
  - P. 171 의 구문 전체 형식 참고

```
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 열이름  
FROM 테이블이름  
WHERE 조건



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT...FROM>

### ■ USE 구문

- 사용할 DB를 지정하는 구문
- 다른 DB를 사용하겠다 명시하지 않으면 같은 DB 사용
- HeidiSQL 에서는 클릭으로 해결

USE 데이터베이스\_이름;

The screenshot shows the HeidiSQL interface with the title bar "localhost#employees - HeidiSQL 9.4.0.5125". The left sidebar lists databases: localhost (selected), employees (197.4 MiB), information\_schema (176.0 KiB), modeldb, mysql, performance\_schema, shopdb, and test. The main pane displays the "employees" database structure with tables: departments, dept\_emp, dept\_manager, employees, salaries, and titles. Each table has columns: 이름 (Name), 행 (Rows), 크기 (Size), 생성됨 (Created), 업데이트 (Updated), 엔진 (Engine), 코멘트 (Comment), and 유형 (Type). The employees table has 298,936 rows and 14.5 MiB size.

이름	행	크기	생성됨	업데이트	엔진	코멘트	유형
departments	9	32.0 KiB	2018-12-21 16:2...		InnoDB		Table
dept_emp	331,143	22.5 MiB	2018-12-21 16:2...		InnoDB		Table
dept_manager	24	48.0 KiB	2018-12-21 16:2...		InnoDB		Table
employees	298,936	14.5 MiB	2018-12-21 16:2...		InnoDB		Table
salaries	2,838,426	130.2 MiB	2018-12-21 16:2...		InnoDB		Table
titles	443,378	30.1 MiB	2018-12-21 16:2...		InnoDB		Table



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT...FROM>

### ■ SELECT와 FROM

The screenshot shows the MySQL Workbench interface. On the left, the database browser displays the 'employees' schema selected under the 'localhost' connection. In the center, a query editor window shows the SQL command '1 SELECT \* FROM titles;'. Below it, the results pane displays the 'titles' table with 443,308 rows. The table has columns: emp\_no, title, from\_date, and to\_date. A dotted arrow points from the 'titles' table back up to the 'titles' command in the query editor. At the bottom, the status bar shows the command was run at 00:00 h on Jan 5, 2019, and the UTC time is 2019-01-02 04:55:00.

emp_no	title	from_date	to_date
10,001	Senior Engineer	1986-06-26	9999-01-01
10,002	Staff	1996-09-03	9999-01-01
10,003	Senior Engineer	1995-12-03	9999-01-01
10,004	Engineer	1986-12-01	1995-12-01
10,004	Senior Engineer	1995-12-01	9999-01-01
10,005	Senior Staff	1996-09-12	9999-01-01
10,005	Staff	1989-09-12	1996-09-12
10,006	Senior Engineer	1990-08-05	9999-01-01

```
24 SELECT * FROM titles;
25 /* Affected rows: 0  총 읽은 행: 443,308  경고: 0  지속 시간: 1 쿼리: 0.000 sec. (+ 0.390 sec. network) */
```

1 :: 연결됨: 00:00 h | MariaDB 10.3.11 | 가동 시간: 5 일, 00:17 h | UTC: 2019-01-02 오전 4:55:00 유 휴



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT...FROM>

- **SELECT와 FROM**

- **SQL 수행 결과**

- 25: MariaDB에서 처리된 순번. 내부적으로 수행 한 쿼리를 포함해 번호 부여
    - **Affected rows**: 변경된 행의 개수를 나타낸다. SELECT문이므로 변경된 행이 별도로 없다.
    - **찾은 행**: SELECT문은 조회된 행의 개수
    - **경고**: 경고가 발생될 경우 경고의 개수
    - **지속시간 1 쿼리**: 실행된 쿼리의 개수와 쿼리가 실행된 시간
    - (+ 시간. network) : 네트워크 전송시간



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT…FROM>

### ▪ SELECT와 FROM

- FROM 뒤에 오는 DB이름.테이블 이름
- 보통 USE DB 를 먼저 사용하므로 아래의 두 구문은 같음

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



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT…FROM>

### ■ SELECT와 FROM

- 해당 테이블에서 원하는 열 가져오기
  - 여러 개의 열을 가져올 때는 콤마로 구분
  - 순서는 사용자가 원하는 대로

employees (3×300,024)		
first_name	last_name	gender
Georgi	Facello	M
Bezalel	Simmel	F
Parto	Bamford	M
Chirstian	Koblick	M
Kyoichi	Maliniak	M
Anneke	Preusig	F
Tzvetan	Zielinski	F
Saniya	Kalloufi	M



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT...FROM>

### ■ SELECT와 FROM 실습

- 조회 내용의 이름이 명확하지 않을 때 조회하는 방법
- DB 조회
  - SHOW DATABASES;
  - 찾던 DB가 있으면 USE DB이름

SCHEMATA (1×7)	
Database	
employees	
information_schema	
modeldb	
mysql	
performance_schema	
shopdb	
test	



# 6.1 SELECT 문

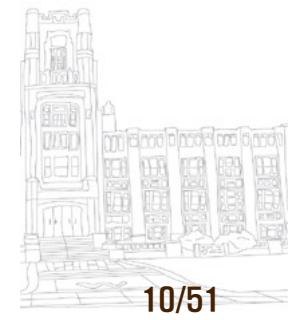
## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT…FROM>

- SELECT와 FROM 실습

- TABLE 조회

- SHOW TABLE STATUS;

TABLES (20x6)						
Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length
departments	InnoDB	10	Dynamic	9	1,820	16,384
dept_emp	InnoDB	10	Dynamic	331,143	36	12,075,008
dept_manager	InnoDB	10	Dynamic	24	682	16,384
employees	InnoDB	10	Dynamic	298,936	50	15,220,736
salaries	InnoDB	10	Dynamic	2,838,426	35	100,270,080
titles	InnoDB	10	Dynamic	443,378	46	20,512,768



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT…FROM>

- **SELECT와 FROM 실습**

- **열 이름 조회**

- **DESCRIBE employees; 또는 DESC employees;**

COLUMNS (6x6)					
Field	Type	Null	Key	Default	Extra
emp_no	int(11)	NO	PRI	(NULL)	
birth_date	date	NO		(NULL)	
first_name	varchar(14)	NO		(NULL)	
last_name	varchar(16)	NO		(NULL)	
gender	enum('M','F')	NO		(NULL)	
hire_date	date	NO		(NULL)	

- **최종적으로 데이터 조회**

- **SELECT first\_name, gender FROM employees;**



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT...FROM>

### ■ SELECT와 FROM 실습

#### • 별칭 AS

- 열 이름을 다르게 볼 수 있어 보기 좋음
- 중간에 공백이 있다면 이름 전체에 “ ” 사용
- 복잡한 열 이름 대신 한글로 정리 가능

employees (3x300,024)		
이름	성별	회사 입사일
Georgi	M	1986-06-26
Bezalel	F	1985-11-21
Parto	M	1986-08-28
Chirstian	M	1986-12-01
Kyoichi	M	1989-09-12
Anneke	F	1989-06-02
Rzvet	F	1989-09-10



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT...FROM>

### ■ SELECT와 FROM 실습

- 샘플 DB 작성 – SQL 저장해 놓는 것 추천

sqlDB

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

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

PK FK

13/51

# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT...FROM>

### ▪ SELECT와 FROM 실습

- 샘플 DB 작성

- DB와 테이블 만든 뒤 데이터 INSERT 하면 완성
- 데이터 확인

usertbl (8x10)							buytbl (6x12)						
userID	name	birthYear	addr	mobile1	mobile2	height	mDate	num	userID	prodName	groupName	price	amount
BBK	바비킴	1,973	서울	010	00000000			1	KBS	운동화	(NULL)	30	2
EJW	은지원	1,972	경북	011	88888888			2	KBS	노트북	전자	1,000	1
JKW	조관우	1,965	경기	018	99999999			3	JYP	모니터	전자	200	1
JYP	조용필	1,950	경기	011	44444444			4	BBK	모니터	전자	200	5
KBS	김범수	1,979	경남	011	22222222			5	KBS	청바지	의류	50	3
KKH	김경호	1,971	전남	019	33333333			6	BBK	메모리	전자	80	10
LJB	임재범	1,963	서울	016	66666666			7	SSK	책	서적	15	5
LSG	이승기	1,987	서울	011	11111111			8	EJW	책	서적	15	2
SSK	성시경	1,979	서울	(NULL)	(NULL)			9	EJW	청바지	의류	50	1
YJS	윤종신	1,969	경남	(NULL)	(NULL)			10	BBK	운동화	(NULL)	30	2



# 6.1 SELECT 문

## 6.1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT...FROM>

### ■ SELECT와 FROM 실습

- 문제 생긴 DB 초기화

- 초기화 해주는 SQL 파일 불러와 실행

The screenshot shows the HeidiSQL interface. The left sidebar lists databases: employees, information\_schema, modeldb, mysql, performance\_schema, shopdb, sqlDB, and test. The main pane displays the following SQL code:

```
1 DROP DATABASE IF EXISTS sqlDB; -- 만약 sqlDB가 존재하면 우선 삭제
2 CREATE DATABASE sqlDB;
3
4 USE sqlDB;
5 CREATE TABLE userTbl -- 회원 테이블
6 ( userID      CHAR(8) NOT NULL PRIMARY KEY, -- 사용자 아이디(PK)
7   name        VARCHAR(10) NOT NULL, -- 이름
8   birthYear    INT NOT NULL, -- 출생년도
9   addr         CHAR(2) NOT NULL, -- 지역(경기, 서울, 경남 식으로 2글자)
10  mobile1     CHAR(3), -- 휴대폰의 국번(011, 016, 017, 018, 019, 010)
11  mobile2     CHAR(8), -- 휴대폰의 나머지 전화번호(하이픈 제외)
           SMALLINT )
```

The toolbar at the top has a play button icon highlighted with a red box. The right sidebar contains links to various features: 열 (Columns), SQL 함수 (SQL Functions), SQL 키워드 (SQL Keywords), 스니펫 (Snippets), 쿼리 내역 (Query History), 쿼리 프로필 (Query Profile), and 매개변수 (Parameters).

# 6.1 SELECT 문

## 6.1.2 특정한 조건의 데이터만 조회하는 <SELECT… FROM … WHERE>

- 기본적인 WHERE 절
  - 특정한 조건의 데이터만 보고 싶을 때 사용

```
SELECT 필드이름 FROM 테이블이름 WHERE 조건식;
```

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

usertbl (8x1)							
userID	name	birthYear	addr	mobile1	mobile2	height	mDate
KKH	김경호	1,971	전남	019	3333333	177	2007-07-07

[그림 6-16] 쿼리 실행 결과



## 6.1 SELECT 문

### 6.1.2 특정한 조건의 데이터만 조회하는 <SELECT… FROM … WHERE>

- 관계 연산자의 사용

- ‘…했거나’ , ‘… 또는’ – OR 연산자
- ‘…하고’ , ‘…면서’ , ‘… 그리고’ – AND 연산자
- 조건 연산자(=, <, >, <=, >=, <>, != 등)와 관계 연산자(NOT, AND, OR 등)를 잘 조합해 쿼리 생성



## 6.1 SELECT 문

### 6.1.2 특정한 조건의 데이터만 조회하는 <SELECT… FROM … WHERE>

- BETWEEN… AND와 IN() 그리고 LIKE
  - BETWEEN… AND
    - 숫자로 구성되고 연속적인 값을 가질 경우
    - Ex) 180, 183
  - IN()
    - 이산적인 (Discrete) 값을 가진 데이터
    - Ex) 경남, 전북, 전남
  - LIKE
    - 문자열의 내용 검색
    - DB 성능 저하의 원인 – 큰 DB에는 피할 것
    - ‘\_’ 앞에 한 글자 ‘%’ 뒤에 여러 글자



## 6.1 SELECT 문

---

### 6.1.2 특정한 조건의 데이터만 조회하는 <SELECT… FROM … WHERE>

- ANY/ALL/SOME , 서브쿼리(SubQuery, 하위쿼리)
  - 서브쿼리란?
    - 쿼리문 안에 또 쿼리문이 들어 있는 것
  - ANY
    - 서브쿼리의 여러 개의 결과 중 한 가지만 만족해도 OK
    - SOME은 ANY와 동일한 의미
  - ALL
    - 서브쿼리의 여러 개의 결과를 모두 만족시켜야 함



## 6.1 SELECT 문

---

### 6.1.2 특정한 조건의 데이터만 조회하는 <SELECT… FROM … WHERE>

- 원하는 순서대로 정렬하여 출력: ORDER BY

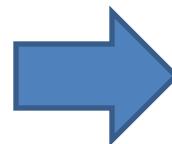
- 결과물에 대해 영향을 미치지는 않음
- 결과가 출력되는 순서 조절하는 구문
- 기본적으로 오름차순 정리
- 내림차순은 열 이름 뒤에 DESC 라 적어줄 것
- ORDER BY 절은 SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY 중에서 제일 뒤



# 6.1 SELECT 문

## 6.1.2 특정한 조건의 데이터만 조회하는 <SELECT… FROM … WHERE>

- 중복된 것은 하나만 남기는 DISTINCT
- **SELECT DISTINCT addr FROM userTbl;**



The diagram illustrates the effect of the DISTINCT keyword on a table. On the left, a table titled "usertbl (1x10)" contains 10 rows of address data, with some entries being duplicates. On the right, a second table titled "usertbl (1x5)" shows the result after applying DISTINCT, where only unique address values are retained.

addr
서울
경북
경기
경기
경남
전남
서울
서울
서울
경남

addr
서울
경북
경기
경남
전남



## 6.1 SELECT 문

---

### 6.1.2 특정한 조건의 데이터만 조회하는 <SELECT… FROM … WHERE>

- 출력하는 개수를 제한하는 LIMIT
  - 30만 명 가운데 입사일이 오래된 다섯 명 출력
    - ORDER BY 는 30만 건 모두를 검색
    - ORDER BY 상위 N개 출력 (LIMIT N) 의 경우
      - » 개수가 차면 검색 멈춤
      - » DB의 부하를 줄여 악성 쿼리문 피함



# 6.1 SELECT 문

## 6.1.2 특정한 조건의 데이터만 조회하는 <SELECT… FROM … WHERE>

- 테이블을 복사하는 CREATE TABLE … SELECT
  - PK나 FK 같은 제약 조건은 복사되지 않음
  - 테이블을 테이블로 복사
  - 열 정보를 테이블로 복사

형식:

CREATE TABLE 새로운테이블 (SELECT 복사할열 FROM 기존테이블)

```
CREATE TABLE buyTb13 (SELECT userID, prodName FROM buyTb1);
SELECT * FROM buyTb13;
```



# 6.1 SELECT 문

## 6.1.3 GROUP BY 및 HAVING 그리고 집계 함수

### ▪ GROUP BY절

- 그룹으로 묶어주는 절

형식 :

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



# 6.1 SELECT 문

## 6.1.3 GROUP BY 및 HAVING 그리고 집계 함수

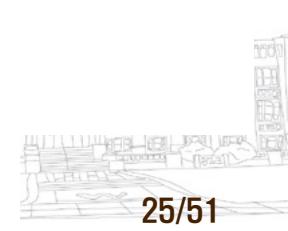
### ▪ GROUP BY절

- 집계 함수(Aggregate Function)

- GROUP BY절과 함께 쓰이며 데이터를 그룹화 Grouping 해주는 기능
  - Ex) 개인 별 물건 구매 총량

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

buytbl (2x5)	
userID	SUM(amount)
BBK	19
EJW	4
JYP	1
KBS	6
SSK	5



# 6.1 SELECT 문

## 6.1.3 GROUP BY 및 HAVING 그리고 집계 함수

### ▪ GROUP BY절

- 집계 함수(Aggregate Function)

- 결과를 별칭 사용해 보기 좋게 만들기
  - 구매액의 총합도 ID 별 계산 가능

```
SELECT userID AS '사용자 아이디', SUM(price*amount) AS '총 구매액'  
FROM buyTbl GROUP BY userID;
```

buytbl (2×5)	
사용자 아이디	총 구매액
BBK	1,920
EJW	95
JYP	200
KBS	1,210
SSK	75



# 6.1 SELECT 문

## 6.1.3 GROUP BY 및 HAVING 그리고 집계 함수

### ▪ GROUP BY절

- Sum이외 집계 함수(Aggregate Function)의 종류

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



# 6.1 SELECT 문

## 6.1.3 GROUP BY 및 HAVING 그리고 집계 함수

### ▪ GROUP BY절

- 집계 함수 AVG를 userID에 따라 출력

```
SELECT userID, AVG(amount) AS '평균 구매 개수' FROM buyTbl GROUP BY userID;
```

buytbl (2×5)	
userID	평균 구매 개수
BBK	4.7500
EJW	1.3333
JYP	1.0000
KBS	2.0000
SSK	5.0000



# 6.1 SELECT 문

## 6.1.3 GROUP BY 및 HAVING 그리고 집계 함수

### ▪ GROUP BY절

- 가장 큰 키와 작은 키의 회원 이름과 키 출력하는 SQL
  - 서브 쿼리와 조합해 구함

```
SELECT Name, height  
      FROM userTbl  
 WHERE height = (SELECT MAX(height)FROM userTbl)  
       OR height = (SELECT MIN(height)FROM userTbl) ;
```

usertbl (2×2)	
Name	height
조용필	166
성시경	186



# 6.1 SELECT 문

## 6.1.3 GROUP BY 및 HAVING 그리고 집계 함수

- GROUP BY절

- 휴대폰이 있는 사용자의 수 카운트
  - NULL 값인 것은 제외하고 카운트

```
SELECT COUNT(mobile1) AS '휴대폰이 있는 사용자' FROM userTbl;
```

결과 #1 (1x1)
휴대폰이 있는 사용자 8



# 6.1 SELECT 문

## 6.1.3 GROUP BY 및 HAVING 그리고 집계 함수

- GROUP BY 절

- HAVING 절 사용

- 집계 함수는 WHERE 절에 나타날 수 없음
    - GROUP BY 절 다음에 꼭 위치해야 함
    - 총 구매액이 적은 사용자부터 나타내려면 ORDER BY 절을 맨 끝에 사용할 것

```
SELECT userID AS '사용자', SUM(price*amount) AS '총구매액'  
FROM buyTbl  
GROUP BY userID  
HAVING SUM(price*amount) > 1000  
ORDER BY SUM(price*amount) ;
```



# 6.1 SELECT 문

## 6.1.3 GROUP BY 및 HAVING 그리고 집계 함수

- GROUP BY절

- ROLLUP

- 총합 또는 중간 합계가 필요하면

- » GROUP BY절과 WITH ROLLUP문 사용

- 분류(groupName)별로 합계 및 그 총합

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

buytbl (2×5)	
groupName	비용
(NULL)	180
서적	120
의류	200
전자	3,000
(NULL)	3,500

총합계



# 6.1 SELECT 문

## 6.1.3 SQL의 분류

- DML (데이터 조작 언어)

- Data Manipulation Language
- 데이터 조작(선택, 삽입, 수정, 삭제)에 사용되는 언어
- DML 구문이 사용되는 대상은 테이블의 행
  - DML을 사용하기 위해서는 꼭 그 이전에 테이블이 정의되어 있어야 함
- SQL문 중 SELECT, INSERT, UPDATE, DELETE
- 트랜잭션 (Transaction) 이 발생하는 SQL
  - 테이블의 데이터를 변경(입력/수정/삭제) 할 때 임시로 적용시키는 것
  - 실수가 있었을 경우에 임시적용 취소 가능



# 6.1 SELECT 문

## 6.1.3 SQL의 분류

- **DDL (데이터 정의 언어)**

- Data Definition Language
- 데이터베이스, 테이블, 뷰, 인덱스 등의 데이터베이스  
개체를 생성/삭제/변경하는 역할
- CREATE, DROP, ALTER
- DDL은 트랜잭션을 발생시키지 않음
  - 되돌림(ROLLBACK)이나 완전적용(COMMIT)  
불가
  - DDL문은 실행 즉시 MariaDB에 적용



# 6.1 SELECT 문

---

## 6.1.3 SQL의 분류

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



## 6.2 데이터의 변경을 위한 SQL문

### 6.2.1 데이터의 삽입: INSERT

- **INSERT문 기본 형식**

- **INSERT는 테이블에 데이터를 삽입하는 명령어**
- **테이블 이름 다음에 나오는 열은 열 이름을 생략할 경우 차례대로 다 써줄 것**
- **열을 생략하려면 목록을 순서에 맞춰 나열해야 함**
- **기본적인 형식**

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



## 6.2 데이터의 변경을 위한 SQL문

### 6.2.1 데이터의 삽입: INSERT

- INSERT문 기본 형식
  - 자동으로 증가하는 AUTO\_INCREMENT
    - 테이블 속성이 AUTO\_INCREMENT로 지정
      - » INSERT에서는 해당 열은 입력 X
      - » 자동으로 1부터 증가하는 값 입력
    - AUTO\_INCREMENT로 지정할 때는 꼭 PRIMARY KEY 또는 UNIQUE로 지정
    - 데이터 형은 숫자 형식만 사용 가능
    - AUTO\_INCREMENT로 지정된 열
      - » INSERT문에서 NULL값 지정하면 자동 입력



## 6.2 데이터의 변경을 위한 SQL문

### 6.2.1 데이터의 삽입: INSERT

- **INSERT문 기본 형식**
  - **SELECT LAST\_INSERT\_ID( )문**
    - 사용하면 마지막에 입력된 값 출력
  - **ALTER TABLE 테이블이름 AUTO\_INCREMENT=시작 값;**
    - 증가 시작값 변경
  - **SET @@auto\_increment\_increment=증가값 ;**
    - 서버 변수 변경으로 증가값 지정
  - **P.207~ 209의 예제 참고**



## 6.2 데이터의 변경을 위한 SQL문

### 6.2.1 데이터의 삽입: INSERT

- 대량의 샘플 데이터 생성

- INSERT INTO… SELECT문

- 기존의 대량의 데이터를 샘플 데이터로 사용할 때  
유용

형식:

```
INSERT INTO 테이블이름 (열이름1, 열이름2, …)  
SELECT문 ;
```

- CREATE TABLE… SELECT 구문

- 테이블 정의까지 생략

```
USE sqlDB;  
CREATE TABLE testTBL5  
(SELECT emp_no, first_name, last_name FROM employees.employees) ;
```

## 6.2 데이터의 변경을 위한 SQL문

### 6.2.2 데이터의 수정: UPDATE

- UPDATE 문의 형식

- WHERE절은 생략이 가능
- WHERE절을 생략하면 테이블 전체의 행 변경

```
UPDATE 테이블이름
```

```
    SET 열1=값1, 열2=값2 ...
```

```
    WHERE 조건 ;
```



## 6.2 데이터의 변경을 위한 SQL문

### ❖ 6.2.3 데이터의 삭제: DELETE FROM

- **DELETE 문의 형식**

- WHERE 절이 생략되면 테이블 전체의 데이터 삭제
- 행 단위 삭제의 개념
- 상위 몇 개 케이스의 경우 LIMIT N 구문과 함께 사용

```
DELETE FROM 테이블이름 WHERE 조건 ;
```



## 6.2 데이터의 변경을 위한 SQL문

### ❖ 6.2.3 데이터의 삭제: DELETE FROM ▪ DELETE, DROP, TRUNCATE

- 테이블 3개를 생성해 삭제하는 실습

```
USE sqlDB;
```

```
CREATE TABLE bigTBL1 (SELECT * FROM employees.employees);
```

```
CREATE TABLE bigTBL2 (SELECT * FROM employees.employees);
```

```
CREATE TABLE bigTBL3 (SELECT * FROM employees.employees);
```

```
16 DELETE FROM bigTBL1;
17 /* Affected rows: 300,024  찾은 행: 0  경고: 0  지속 시간 1 쿼리: 0.468 sec. */
18 DROP TABLE bigTBL2;
19 /* Affected rows: 0  찾은 행: 0  경고: 0  지속 시간 1 쿼리: 0.406 sec. */
20 TRUNCATE TABLE bigTBL3;
21 /* Affected rows: 0  찾은 행: 0  경고: 0  지속 시간 1 쿼리: 0.485 sec. */
```

연결됨: 00:02 h

MariaDB 10.3.11

가동 시간: 5 일, 04:20 h

UTC: 2019-01-02 오전 8:59 ◎ 유튜



## 6.2 데이터의 변경을 위한 SQL문

### 6.2.3 데이터의 삭제: DELETE FROM

- **DELETE, DROP, TRUNCATE**
  - **DELETE**
    - DML문이라 트랜잭션 로그 기록 -> 성능 나쁨
    - Affected Rows 표시
  - **DROP**
    - 테이블 자체 삭제 -> 아예 필요 없을 경우
  - **TRUNCATE**
    - 트랜잭션 로그 기록 X
    - 테이블 구조는 남기고 싶을 때



## 6.2 데이터의 변경을 위한 SQL문

### 6.2.4 조건부 데이터 입력, 변경

- 여러 건을 입력하는데 오류가 생겨도 진행하는 방법
  - 보통 한 건이 오류 생기면 그 뒤로 입력 안됨
  - **INSERT IGNORE**
    - PK중복처럼 오류가 생겨도 오류를 발생시키지 않아 오류 이후의 데이터는 입력됨
  - **ON DUPLICATE UPDATE**
    - PK가 중복되지 않으면 일반 INSERT
    - PK가 중복되면 그 뒤의 UPDATE문 수행



## 6.3 WITH절과 CTE

### 6.3.1 WITH절과 CTE 개요

- **WITH절**

- **CTE (Common Table Expression) 표현 구문**
  - 기존의 뷰, 파생 테이블, 임시 테이블 등으로 사용 되던 것을 대신
  - 더 간결한 식으로 보여지는 장점
  - ANSI-SQL99 표준
    - » 기존의 SQL은 ANSI-SQL92 기준
    - » 최근의 DBMS는 ANSI-SQL99와 호환
  - 재귀적/비재귀적 CTE



# 6.3 WITH절과 CTE

## 6.3.2 비재귀적 CTE

- 재귀적이지 않은 CTE
- 기본 형식

```
WITH CTE_테이블이름(열이름)
AS
(
    <쿼리문>
)
SELECT 열이름 FROM CTE_테이블이름 ;
```

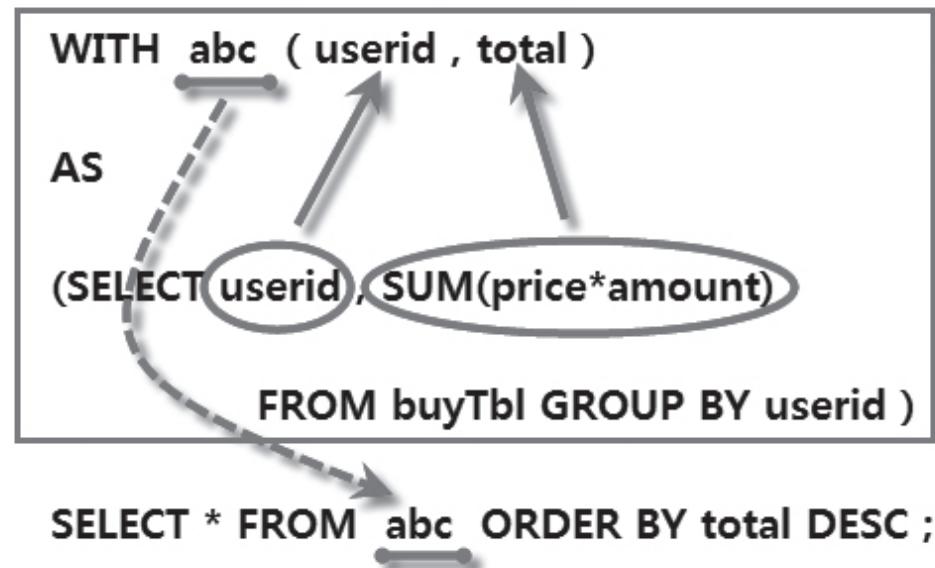
```
WITH abc(userid, total)
AS
(SELECT userid, SUM(price*amount)
     FROM buyTBL GROUP BY userid )
SELECT * FROM abc ORDER BY total DESC ;
```

# 6.3 WITH절과 CTE

## 6.3.2 비재귀적 CTE

- CTE 쿼리의 결과와 작동 원리

buytbl (2×5)	
userid	total
BBK	1,920
KBS	1,210
JYP	200
EJW	95
SSK	75



## 6.3 WITH절과 CTE

### 6.3.2 비재귀적 CTE

- 회원 테이블에서 각 지역 별 가장 큰 키 뽑아 평균 구하기
  - CTE를 사용하되 각 단계별로 작업을 정리
  - 1단계 → “각 지역별로 가장 큰 키” 를 뽑는 쿼리

```
SELECT addr, MAX(height) FROM userTBL GROUP BY addr
```

- 2단계 → 위 쿼리를 WITH 구문으로 묶기

```
WITH cte_userTBL(addr, maxHeight)
AS
( SELECT addr, MAX(height) FROM userTBL GROUP BY addr)
```



# 6.3 WITH절과 CTE

## 6.3.2 비재귀적 CTE

- 회원 테이블에서 각 지역 별 가장 큰 키 뽑아 평균 구하기
  - 3단계 → 키의 평균을 구하는 쿼리 작성

```
SELECT AVG(키) FROM CTE_테이블이름
```

- 4단계 → 2단계와 3단계의 쿼리를 합치기
  - 키의 평균을 실수로 만들기 위해서 키에 1.0을 곱해서 실수로 변환

결과 #1 (1x1)	
각 지역별 최고키의 평균	
176.40000	

```
WITH cte_userTBL(addr, maxHeight)
AS
( SELECT addr, MAX(height) FROM userTBL GROUP BY addr)
SELECT AVG(maxHeight*1.0) AS '각 지역별 최고키의 평균' FROM cte_userTBL;
```

# 6.3 WITH절과 CTE

## 6.3.2 비재귀적 CTE

- 뷰는 구문이 끝나도 계속 존재
  - 다른 구문에서도 사용 가능
- CTE와 파생 테이블
  - 뷰와 그 용도는 비슷하지만 개선된 점 있음
  - 구문이 끝나면 같이 소멸
- 중복 CTE에서는 정의되지 않은 CTE 미리 참조 불가

WITH

AAA (컬럼들)

AS ( AAA의 쿼리문 ),

BBB (컬럼들)

AS ( BBB의 쿼리문 ),

CCC (컬럼들)

AS ( CCC의 쿼리문 )

SELECT \* FROM [AAA 또는 BBB 또는 CCC]





# Thank You !

이것이 MariaDB다