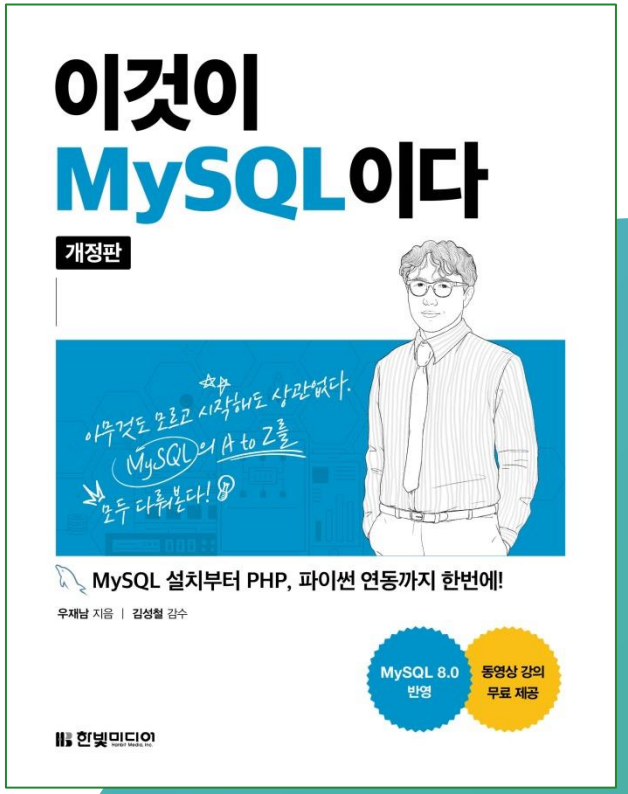
Chapter 06: SQL 기본

이것이 MySQL이다



저자 : 우재남

Contents

Contents

CHAPTER 06 SQL 기본

데이터베이스를 운영하기 위한 기본적인 SQL문에 대하여 알아본다.

SECTION 01 SELECT문

<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
```

USE 구문

- SELECT문 학습 위해 사용할 데이터베이스 지정
- 지정해 놓은 후 특별히 다시 USE문 사용하거나 다른 DB를 사용하겠다고 명시하지 않는 이상 모든 SQL문은 지정 DB에서 수행

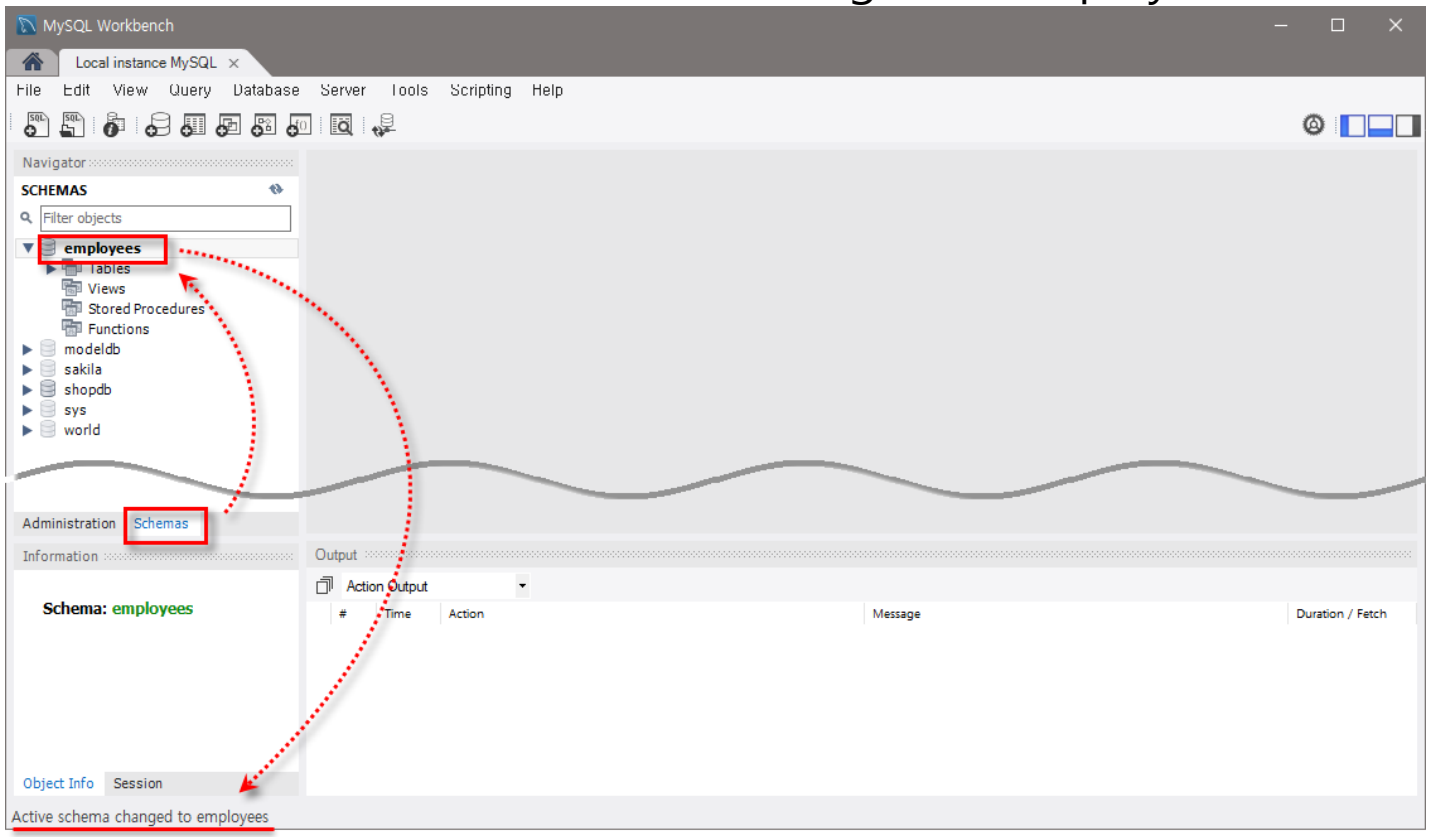
```
USE 데이터베이스_이름;
```

- employees를 사용하기 위해서는 쿼리 창에 다음과 같이 입력한다.

```
USE employees;
```

USE 구문

- Workbench 에서 직접 선택해서 사용도 가능
  - [Navigator]의 [Schemas] 탭, employees 데이터베이스를 더블 클릭하거나 마우스 오른쪽 버튼을 클릭한 후 [Set as Default Schema]를 선택
    - 진한 글자로 전환, 왼쪽 아래'Active schema changed to employees' 메시지 나옴



SECTION 01 SELECT문

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

SECTION 01 SELECT문

SELECT와 FROM

- 주석(Remark)

여기서 잠깐

주석(Remark)

MySQL은 '--' 이후부터 주석으로 처리된다. 주로 코드에 설명을 달거나 잠시 해당 부분의 실행을 막고 싶을 때 사용한다. 주의할 점은 -- 뒤에 바로 붙여서 쓰면 안되며, 공백이 하나 이상 있어야 한다.

```
-- 한 줄 주석 연습
SELECT first_name, last_name, gender -- 이름과 성별 열을 가져옴
FROM employees;
```

여러 줄 주석은 '/* */'로 묶는다.

```
/* 블록 주석 연습
SELECT first_name, last_name, gender
FROM employees;
*/
```

주석으로 묶이면 해당 글자들은 모두 회색으로 보인다.

DB, TABLE, 열의 이름이 확실하지 않을 때 조회하는 방법

- 현재 서버에 어떤 DB가 있는지 보기
  - SHOW DATABASES;
- 현재 서버에 어떤 TABLE이 있는지 보기
  - 데이터베이스에 있는 테이블 정보 조회
    - SHOW TABLE STATUS;
  - 테이블 이름만 간단히 보기
    - SHOW TABLES;
- employees 테이블의 열이 무엇이 있는지 확인
  - DESCRIBE employees; 또는 DESC employees;
- Workbench의 [Navigator]로 확인 가능 하나 명령어를 알아두면 Linux 명령어 모드에서 사용 가능

SECTION 01 SELECT문

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;
```

SECTION 01 SELECT문

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%';
```

ANY/ALL/SOME ,서브쿼리(SubQuery, 하위쿼리)

◦ 서브쿼리

- 쿼리문 안에 또 쿼리문이 들어 있는 것
- 서브쿼리 사용하는 쿼리로 변환 예제
  - ex) 김경호보다 키가 크거나 같은 사람의 이름과 키 출력
    - WHERE 조건에 김경호의 키를 직접 써주는 것을 쿼리로 해결

```
SELECT name,  height FROM usertbl WHERE height  > 177;
```

```
SELECT name,  height FROM usertbl
   WHERE height > (SELECT height FROM usertbl WHERE Name = '김경호');
```

- **서브쿼리의 결과가 둘 이상이 되면 에러 발생**

ANY/ALL/SOME ,서브쿼리(SubQuery, 하위쿼리)

- ANY
  - 서브쿼리의 여러 개의 결과 중 한 가지만 만족해도 가능
  - SOME은 ANY와 동일한 의미로 사용
  - '= ANY(서브쿼리)'는 'IN(서브쿼리)'와 동일한 의미

- ALL
  - 서브쿼리의 결과 중 여러 개의 결과를 모두 만족해야 함

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

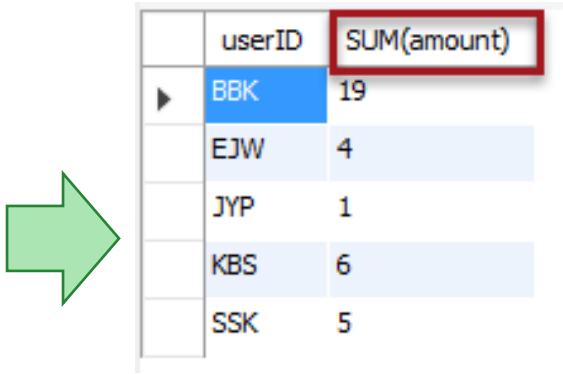
SECTION 01 SELECT문

- 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

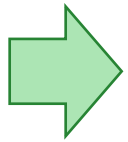
◦ GROUP BY와 함께 자주 사용되는 집계 함수

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 ;
```

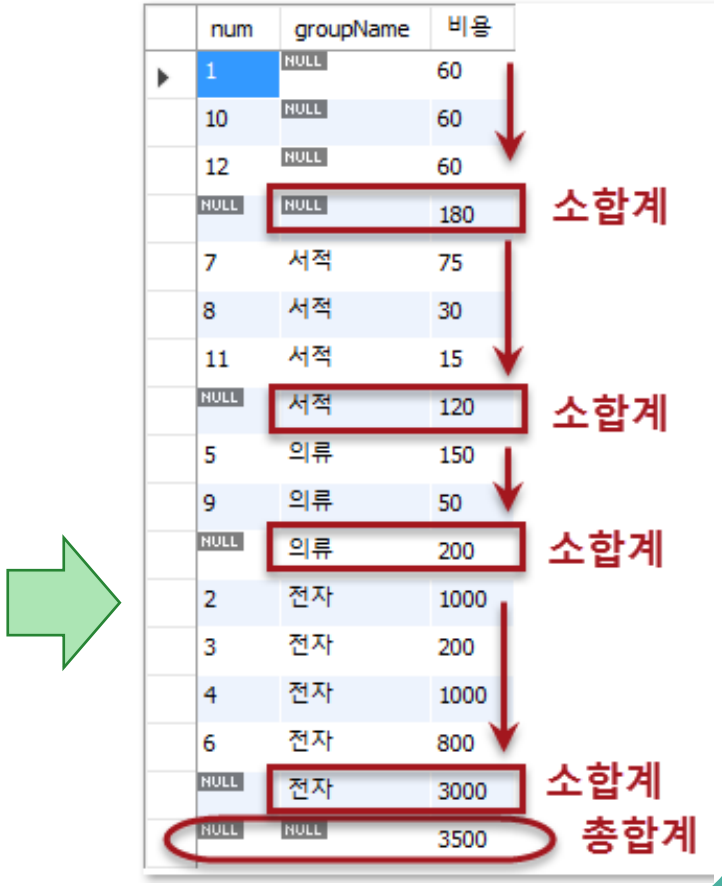


	평균 구매 개수
▶	2.9167

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;
```



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

SECTION 02 데이터의 변경을 위한 SQL문

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

SECTION 02 데이터의 변경을 위한 SQL문

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

SECTION 02 데이터의 변경을 위한 SQL문

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

SECTION 02 데이터의 변경을 위한 SQL문

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

SECTION 02 데이터의 변경을 위한 SQL문

조건부 데이터 입력, 변경

- 기본 키가 중복된 데이터를 입력한 경우
  - 오류로 입력 불가

- 대용량 데이터 처리의 경우 에러 발생하지 않은 구문 실행
  - INSERT IGNORE문
    - 에러 발생해도 다음 구문으로 넘어가게 처리
    - 에러 메시지 보면 적용되지 않은 구문이 어느 것인지 구분 가능
  - ON DUPLICATE KEY UPDATE 구문
    - 기본 키가 중복되면 데이터를 수정되도록 하는 구문도 활용 가능

SECTION 03 WITH절과 CTE

WITH절과 CTE 개요

- WITH절은 CTE(Common Table Expression)를 표현하기 위한 구문
- MySQL 8.0 이후부터 사용 가능하게 됨
- CTE는 기존의 뷰, 파생 테이블, 임시 테이블 등을 대신할 수 있으며 간결한 식으로 보여짐
- CTE는 ANSI-SQL99 표준(기존 SQL은 ANSI-SQL92 기준)
- CTE는 비재귀적 CTE와 재귀적 CTE가 있지만 주로 사용되는 것은 비재귀적 CTE

SECTION 03 WITH절과 CTE

비재귀적 CTE

◦ 단순한 형태, 복잡한 쿼리문장을 단순화하는데 적합

```
WITH CTE_테이블이름(열 이름)
AS
(
    〈쿼리문〉
)
SELECT 열 이름 FROM CTE_테이블이름 ;
```

- CTE는 뷰와 용도가 비슷하지만 개선된 점이 많음
- **뷰는 계속 존재**해서 다른 구문에서도 사용 가능하지만, **CTE와 파생 테이블은 구문이 끝나면 소멸됨**
- 중복 CTE 허용됨

이것이 MySQL 이다

Thank You!