제2장
SQL 기초-1

1. SELECT문

- <SELECT... FROM>
  - 원하는 데이터를 가져와 주는 기본적인 구문
  - 가장 많이 사용되는 구문
  - 데이터베이스 내 테이블에서 원하는 정보 추출하는 기능

1. SELECT문

❖ SELECT의 구문 형식

- 복잡한 형식을 실제 사용되는 형태로 요약

```
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
```

1. SELECT문

❖ 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;
```

- **Workbench 에서 직접 선택해서 사용도 가능**
  - [Navigator]의 [Schemas] 탭
    - **employees 데이터베이스를 더블 클릭하면 진한 글자 전환**
    - **왼쪽 아래 'Active schema changed to employees' 메시지**

1. 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
- 여러 개의 열을 가져오고 싶을 때는 콤마로 구분
- 열 이름의 순서는 출력하고 싶은 순서대로 배열 가능

1. SELECT문

- **DB, TABLE, 열의 이름이 확실하지 않을 때 조회하는 법**
  - **현재 서버에 어떤 DB가 있는지 보기**
    - SHOW DATABASES;
  - **현재 서버에 어떤 TABLE이 있는지 보기**
    - 데이터베이스 employees 에 있는 테이블 정보 조회
      - **SHOW TABLE STATUS;**
    - 테이블 이름만 간단히 보기
      - **SHOW TABLES;**
  - **employees 테이블의 열이 무엇이 있는지 확인**
    - DESCRIBE employees;
    - DESC employees;

1. SELECT문

❖ 문제가 생긴 DB 초기화하기

▪ DB가 존재한다면 지우고 다시 만들기

```
DROP DATABASE IF EXISTES sqlDB; -- 만약 sqlDB가 존재하면 우선 삭제한다.
CREATE DATABASE sqlDB;
```

- 계속 사용할 쿼리는 SQL 파일로 저장해서 재사용 가능하게 만들기
- 파일 내용을 불러다 쓰기 전에 모든 쿼리 창을 닫도록 함

▪ DB 내용을 입력하는 예제(실습)

1. SELECT문

❖ 특정 조건의 데이터만 조회 – <SELECT FROM WHERE>

▪ 기본적인 WHERE절

- 조회하는 결과에 특정한 조건 줘서 원하는 데이터만 보고 싶을 때 사용
- SELECT 필드이름 FROM 테이블이름 WHERE 조건식;
- 조건이 없을 경우 테이블의 크기가 클수록 찾는 시간과 노력이 증가

▪ 관계 연산자의 사용

- '…했거나', '… 또는' - OR 연산자
- '…하고', '…면서', '… 그리고' - AND 연산자
- 조건 연산자(=, <, >, <=, >=, < >, != 등)와 관계 연산자(NOT, AND, OR 등)의 조합으로 알맞은 데이터를 효율적으로 추출

1. SELECT문

❖ 특정 조건의 데이터만 조회 – <SELECT FROM WHERE>

▪ BETWEEN… AND와 IN( ) 그리고 LIKE

- 데이터가 숫자로 구성되어 있어 연속적인 값
  - **BETWEEN… AND 사용 가능**

- 이산적인 (Discrete) 값의 조건
  - **IN( )**
  - **Ex) SELECT Name, addr FROM userTbl WHERE addr= ' 경남' OR addr= '전남' OR addr= '경북';**
    - **» SELECT Name, addr FROM userTbl WHERE addr IN ('경남','전남','경북');**

- 문자열의 내용 검색하기 위해 LIKE 연산자 사용
  - **문자 뒤에 % – 무엇이든(%) 허용**
  - **한 글자와 매치하기 위해서는 '_' 사용**

1. SELECT문

❖ ANY/ALL/SOME ,서브쿼리(SubQuery, 하위쿼리)

▪ 서브쿼리

- 쿼리문 안에 또 쿼리문이 들어 있는 것
- 서브쿼리 사용하는 쿼리로 변환 예제
  - **김경호보다 키가 크거나 같은 사람의 이름과 키 출력**
    - » **WHERE 조건에 김경호의 키를 직접 써줘야 함**
  - **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(중요)**

1. SELECT문

❖ ANY/ALL/SOME ,서브쿼리(SubQuery, 하위쿼리)

▪ ANY 구문의 필요성

- ANY
  - 서브쿼리의 여러 개의 결과 중 한 가지만 만족해도 가능
  - SOME은 ANY와 동일한 의미로 사용
  - = ANY 구문은 IN과 동일한 의미

- ALL - 서브쿼리의 여러 개의 결과를 모두 만족시켜야 함

1. SELECT문

❖ 원하는 순서대로 정렬하여 출력 : ORDER BY

▪ ORDER BY절

- 결과물에 대해 영향을 미치지는 않음
- 결과가 출력되는 순서를 조절하는 구문
- 기본적으로 오름차순 (ASCENDING) 정렬
- 내림차순 (DESCENDING) 으로 정렬
  - **열 이름 뒤에 DESC 적어줄 것**
- ORDER BY 구문을 혼합해 사용하는 구문도 가능
  - **SELECT Name, height FROM userTbl ORDER BY height DESC, name ASC;**
  - **키가 큰 순서로 정렬하되 만약 키가 같을 경우 이름 순으로 정렬**
  - **ASC(오름차순)는 디폴트 값이므로 생략가능**

1. SELECT문

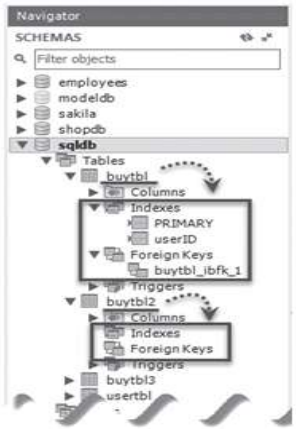
❖ 원하는 순서대로 정렬하여 출력 : ORDER BY

- 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
  - 서버의 처리량을 많이 사용해 서버의 전반적인 성능을 나쁘게 하는 악성 쿼리문 개선할 때 사용

1. SELECT문

- **원하는 순서대로 정렬하여 출력 : ORDER BY**
  - **테이블을 복사하는 CREATE TABLE … (SELECT..)**
    - 테이블을 복사해서 사용할 경우 주로 사용
    - CREATE TABLE 새로운테이블 (SELECT 복사할열 FROM 기존테이블)
    - 지정된 일부 열만 테이블로 복사하는 것도 가능
    - PK나 FK 같은 제약 조건은 복사되지 않음
      - **Workbench의 [Navigator]에서 확인 가능**



1. SELECT문

❖ GROUP BY 및 HAVING 그리고 집계 함수

▪ GROUP BY절

- 말 그대로 그룹으로 묶어주는 역할
- 집계 함수 (Aggregate Function) 함께 사용
  - **효율적인 데이터 그룹화 (Grouping)**
  - **Ex) 각 사용자 별로 구매한 개수를 합쳐 출력**
- 읽기 좋게 하기 위해 별칭 (Alias) 사용

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

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

	사용자 아이디	총 구매액
▶	BBK	1920
	EJW	95
	JYP	200
	KBS	1210
	SSK	75

1. SELECT문

❖ GROUP BY 및 HAVING 그리고 집계 함수

▪ GROUP BY와 함께 자주 사용되는 집계 함수 (집합 함수)

- 서브 쿼리와 함께 조합 가능

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.

1. SELECT문

❖ GROUP BY 및 HAVING 그리고 집계 함수

▪ Having절

- WHERE와 비슷한 개념으로 조건 제한
- 집계 함수에 대해서 조건 제한하는 편리한 개념
- HAVING절은 꼭 GROUP BY절 다음에 나와야 함 !!!

▪ WITH ROLLUP

- 총합 또는 중간합계가 필요할 경우 사용'
- GROUP BY절과 함께 WITH ROLLUP문 사용
- Ex) 분류(groupName) 별로 합계 및 그 총합 구하기

1. SELECT문

❖ 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 꼭 그 이전에 테이블이 정의되어 있어야 함
- SQL문 중 SELECT, INSERT, UPDATE, DELETE가 이 구문에 해당
- 트랜잭션 (Transaction)이 발생하는 SQL도 이 DML에 속함
  - **테이블의 데이터를 변경(입력/수정/삭제)할 때 실제 테이블에 완전히 적용하지 않고, 임시로 적용시키는 것**
  - **취소 가능!(rollback;)**

1. SELECT문

* 트랜잭션의 이해

트랜잭션을 설명할 때 자주 등장하는 예가 '입출금 처리'입니다.
예를 들어, 박진영의 계좌에서 윤종신의 계좌로 10만원을 송금한다고 했을 때, '박진영의 계좌 잔고에서 10만원을 감산'하고, '윤종신의 계좌 잔고에 10만원을 가산'하는 처리가 이루어집니다.
만일, '박진영의 계좌 잔고에서 10만 원을 감산'하는 시점에 오류가 발생했다면 어떻게 될까요?
'윤종신의 계좌 잔고에 10만원을 가산'하는 처리는 이루어 지지 않은 채 박진영의 계좌에 있던 10만원만 어디론가 사라지고 마는 것입니다. 이는 입출금 처리에서 절대로 있어서는 안 되는 사태입니다.
그래서, '10만 원의 감산'과 '10만 원의 가산'을 '분할할 수 없는 하나의 처리'로 취급해서, 만일 10만원의 가산이 실패하면 10만 원의 감산도 취소하도록 합니다. 이렇게 하면 적어도 10만 원이 사라지는 일은 피할 수 있습니다.
이처럼, 여러 단계의 처리를 하나의 처리처럼 다루는 기능을 **트랜잭션(transection)**이라고 합니다.
트랜잭션의 실행 결과를 데이터베이스에 반영하는 것을 **커밋(commit)**이라 하고, 반영하지 않고 원래 상태로 되돌리는 것을 **롤백(roll-back)**이라고 합니다.
Excel 등의 컴퓨터 처리에 익숙하다면, 무엇이든지 ctrl + z 로 되돌릴 수 있다고 생각하기 쉽습니다.
그러나 데이터베이스 세계에서는 트랜잭션을 사용하지 않으면 일단 한번 변경된 데이터는 원래 상태로 되돌릴 수 없다고 생각해야 합니다.

1. SELECT문

❖ SQL의 분류

▪ DDL (Data Definition Language)

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

▪ DCL (Data Control Language)

- 데이터 제어 언어
- 사용자에게 어떤 권한을 부여하거나 빼앗을 때 주로 사용하는 구문
- GRANT/REVOKE


감사합니다.