10

10-2. SQL

데이터베이스

SQL 개요

- 데이터베이스 필수 용어
  - 데이터
    - 하나하나의 단편적인 정보
    - 정보는 있으나 아직 체계화 되지 못한 상태
  - 테이블
    - 데이터를 입력하기 위해, 표 형태로 표현한 것
    - Ex) 회원 정보 테이블, 제품 정보 테이블
  - 데이터베이스(DB)
    - 테이블이 저장되는 저장소
    - 각 데이터베이스는 서로 다른 고유한 이름을 가지고 있음
  - DBMS (DataBase Management System)
    - 데이터베이스를 관리하는 시스템 또는 소프트웨어

SQL 개요

- 데이터베이스 필수 용어
  - 열(=컬럼=필드)
    - 각 테이블은 열로 구성
    - 회원 테이블의 경우에는 아이디, 회원 이름, 주소 등 3개의 열로 구성
  - 열 이름
    - 각 열을 구분하기 위한 이름
    - 열 이름은 각 테이블 내에서는 중복되지 않고, 고유해야 함
  - 데이터 형식
    - 열의 데이터 형식
    - 테이블을 생성할 때 열 이름과 함께 지정

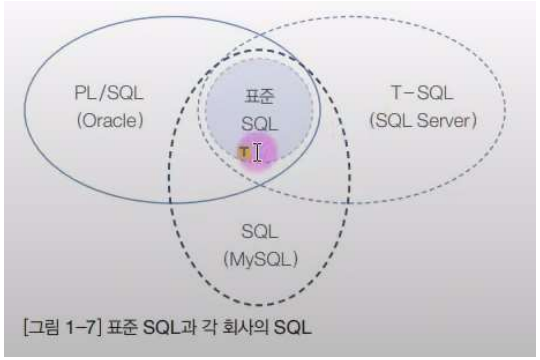
SQL 개요

- 데이터베이스 모델링과 필수 용어
  - 기본 키 (Primary Key) 열
    - 기본 키(또는 주 키) 열은 각 행을 구분하는 유일한 열
    - 중복되어서는 안되며, 비어 있어서도 안 됨
    - 각 테이블에는 기본 키가 하나만 지정
  - SQL (Structured Query Language)
    - 구조화된 질의 언어
    - 사람과 DBMS가 소통하기 위한 말(언어)

SQL 개요

SQL(Structured Query Language)

- 관계형 데이터베이스에서 사용되는 언어, '에스큐엘' 또는 '시퀄'
- DBMS 제작 회사와 독립적
- 다른 시스템으로 이식성이 좋음
- 표준이 계속 발전 중
- 대화식 언어
- 분산형 클라이언트/서버 구조



[그림 1-7] 표준 SQL과 각 회사의 SQL

SQL 개요

- 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, DELETE**가 이 구문에 해당

SQL 개요

- SQL의 분류
  - DDL (Data Definition Language)
    - 데이터 정의 언어
    - 데이터베이스, 테이블, 뷰, 인덱스 등의 데이터베이스 개체를 생성/삭제/변경하는 역할
    - CREATE, DROP, ALTER 구문

CREATE

- 테이블 생성

  - PRIMARY KEY 제약 조건
    - '기본 키Primary Key' 의 개념
      - 테이블에 존재하는 많은 행의 데이터를 구분할 수 있는 식별자
      - 중복되어서도 안되며 비어서도 안됨
      - Ex) 회원 테이블의 회원 아이디, 학생 테이블의 학번
    - 기본 키로 생성한 것은 자동으로 클러스터형 인덱스 생성

```
CREATE TABLE userTbl
( userID  char(8) NOT NULL PRIMARY KEY,
  name    nvarchar(10) NOT NULL,
   --- 중간 생략 ---
```

DROP

- 테이블 삭제
  - DROP TABLE 테이블이름;

ALTER

- 테이블 수정
  - ALTER TABLE문 사용
    - 열의 추가
      - ALTER TABLE userTbl

        ADD 열이름 데이터타입;

      - 기본적으로 가장 뒤에 추가
    - 열의 삭제
      - ALTER TABLE userTbl

        DROP COLUMN 열 이름;

SELECT

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

SELECT

- **SELECT**의 구문 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}]
```

```
SELECT 열이름
FROM 테이블이름
WHERE 조건
```

SELECT

- **SELECT**와 **FROM**
  - **SELECT ***

```
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
    - SELECT 컬럼명1, 컬럼명2, 컬럼명3

      FROM 테이블명;

SELECT

- 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

      - SELECT 컬럼명1, 컬럼명2, 컬럼명3 FROM 테이블명 WHERE 조건;

  - 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

      - SELECT 컬럼명1, 컬럼명2, 컬럼명3 FROM 테이블명 WHERE 조건1 AND 조건2;

SELECT

- 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 생략

SELECT

- 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

  - SELECT 컬럼명1, 컬럼명2, 컬럼명3 FROM 테이블명 WHERE 조건 LIMIT 숫자;

SELECT

- GROUP BY 및 HAVING 그리고 집계 함수
  - **GROUP BY**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}]
```

```
USE sqldb;
SELECT userID AS '사용자', SUM(price*amount) AS '총구매액'
   FROM buytbl
   GROUP BY userID ;
```

SELECT

- GROUP BY 및 HAVING
  - **Having**절
    - WHERE와 비슷한 개념으로 조건 제한
    - 집계 함수에 대해서 조건 제한하는 편리한 개념
    - **HAVING절은 꼭 GROUP BY절 다음에 나와야 함 !!!**

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

INSERT

- 데이터의 삽입 : INSERT
  - **INSERT**문의 기본
    - 테이블 이름 다음에 나오는 열 생략 가능
      - 생략할 경우에 VALUE 다음에 나오는 값들의 순서 및 개수가 테이블이 정의된 열 순서 및 개수와 동일해야 함

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

```
INSERT INTO testTbl1(id, userName) VALUES (2, '설현');
```

INSERT

- 데이터의 삽입 : INSERT
  - 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 에만 사용가능
    - 데이터 형은 숫자 형식만 사용 가능

```
USE  sqldb;
CREATE TABLE testTbl2
  (id  int AUTO_INCREMENT PRIMARY KEY,
   userName char(3),
   age int );
INSERT INTO testTbl2 VALUES (NULL, '지민', 25);
INSERT INTO testTbl2 VALUES (NULL, '유나', 22);
INSERT INTO testTbl2 VALUES (NULL, '유경', 21);
SELECT * FROM testTbl2;
```

UPDATE

- 데이터의 수정 : UPDATE
  - 기존에 입력되어 있는 값 변경하는 구문

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

  - WHERE절 생략 가능하나 테이블의 전체 행의 내용 변경

```
UPDATE testTbl4
    SET Lname = '없음'
    WHERE Fname = 'Kyoichi';
```

DELETE

- 데이터의 삭제 : DELETE FROM
  - 행 단위로 데이터 삭제하는 구문
  - DELETE FROM 테이블이름 WHERE 조건;
  - 테이블을 삭제하는 경우의 속도 비교
    - DML문인 DELETE는 트랜잭션 로그 기록 작업 때문에 삭제 느림
    - DDL문인 DROP과 TRUNCATE문은 트랜잭션 없어 빠름

```
DELETE FROM 테이블이름 WHERE 조건;
```

```
USE sqldb;
DELETE FROM testTbl4 WHERE Fname = 'Aamer';
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare