DAY 1

DATABASE

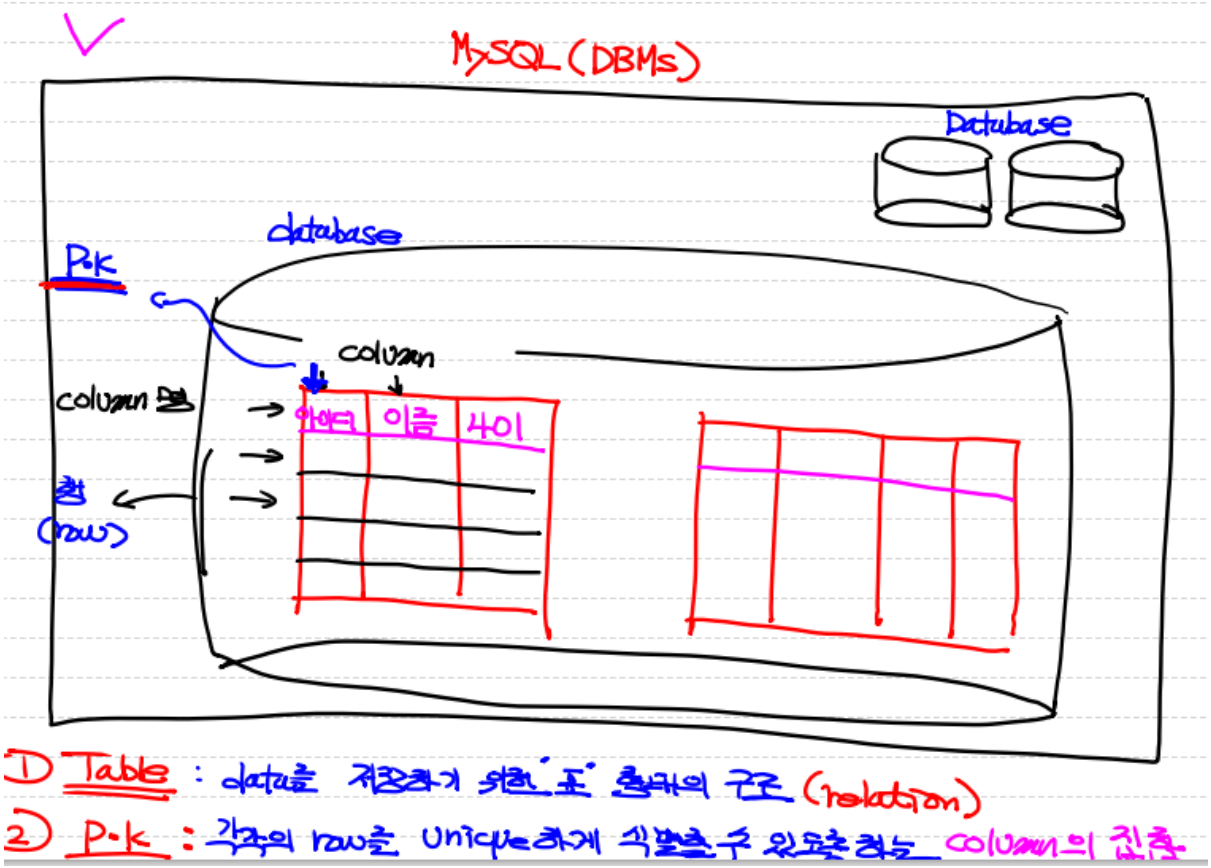
관련있는 대용량의 데이터 집합을 체계적으로 구성해놓은것.

- 여러 명이 사용하기 때문에 데이터 처리에 용이해야하고, 여러 개의 **Database** 관리가 필요하다.
  - 데이터를 **처리하고 관리**하기 위해서 **Software**가 필요한데 이것을 **DMBS**라고 한다.
- MySQL
- MariaDB
- **Oracle** 사용시장 점유율 1등
- IBM DB2: 메인프레임시장 점유율 1등
- MS SQL Server
- Postgres

DBMS 특징

- **Integrity(무결성)**
  - 데이터의 오류가 있어서는 안된다.
    - 데이터가 한번 오염되면 더이상 그 안에 있는 데이터를 신뢰하지 못하기 때문.
- **데이터의 독립성 (Independence)**
  - Database의 크기, 위치등을 변경해도 기존에 사용하고 있는 software는 영향을 받지 않아야한다.
- **보안(Security)**
- **중복**
  - 중복은 허용하나 가능한 최소화시켜야 한다. (중복된 데이터가 많으면 많을수록 무결성의 문제가 있기 때문에, 쓸데없는 operation의 낭비로 인한 성능의 저하 등)
- **안정성 (bacup/restore)**

DBMS 종류



- **계층형 Database**
  - 폴더 안의 폴더를 두듯이 데이터를 보관하는 방식
- **네트워크 Database**
  - 이론상 가능했는데 구현이 안되서 실패!
- **IBM (관계형 Database)**
  - Relational Database
- **객체지향 Database**
  - IBM이 너무 편해서 망함,,
- **객체-관계형 Database**
  - **Oracle**
  - Table구조로 되어있다.

MYSQL (DBMS)

- Table: data를 저장하기 위한 “표” 형태의 구조(relation)
- Primary Key: 각각의 row를 유니크하게 식별살수 있도록하는 컬럼의 집합
- 스키마라는 용어를 database와 동일시한 개념으로 표현한다.

- **MySQL에서만!!!**

Schema

> 💡 **database안에서 data의 구조**

- data의 표현방법
- data의 type
- data의 관계

이것들을 형식언어를 이용해서 정의한 구조이다.

Schema의 종류

- **external schema**
  - Table 형태로 바라보는 제일 익숙한 Schema
- **conceptual schema**
  - 데이터베이스안에 데이터의 논리적인 구조를 명시해 놓은 Schema
- **Internal schema**
  - 물리적인 어떻게 저장이 되는지 기술한 Schema
  - file system

index

primary key를 설정하면 해당 column에 index가 설정

MySQL Query구문

VARCHAR,CHAR의 차이

- `VARCHAR()`
  - 가변문자열이다.
  - 글자의 크기만큼만 메모리의 공간을 차지한다.
- `CHAR()`
  - 고정문자열이다.
  - 무조건 지정된 크기만큼 메모리의 공간을 차지한다.

Q. 그렇다면 글자의 크기만큼만 메모리의 공간을 차지하는 `VARCHAR()`를 쓰지 왜 `CHAR()`을 쓸까??

✅ **CHAR**는 **VARCHAR**에 비해 고정된 메모리를 가지므로 일일히 메모리의 크기를 연산하지 않아도 되서 문자열을 비교하거나 연산을 수행할때 빠르다는 장점이있다.

table생성

```
CREATE TABLE indexTBL(
first_name VARCHAR(14),
last_name VARCHAR(16),
hire_date DATE
);
```

데이터삽입

```
INSERT INTO indexTBL
SELECT first_name, last_name, hire_date
FROM employees.employees
LIMIT 500;
```

index 설정

```
CREATE INDEX idx_indexTBL_firstname ON indexTBL(first_name);
SELECT * FROM shopdb.indextbl WHERE first_name = 'Mary';
```

VIEW

가상의 table 인데, 실제로 view는 data를 가지고 있지 않다.

View 생성

```
CREATE VIEW v_memberTBL
AS
SELECT memberName,memberAddress FROM memberTBL;
```

VIEW는 왜 쓸까??

1. **data를 안전하게 유지하기 위해서**
    a. 실제 테이블을 가지고 작업을 하면 데이터가 유실될 위험이 있고, 무결성의 원칙에 어긋나기 때문에 개발자에게 보여주지 않기 위해서이다.
2. **보안적인 측면**
    a. 테이블 안에는 일반 개발자, 사용하는 사람들에게 보여주면 안되는 컬럼들이 있다. 그래서 table을 숨기고 view만 보여주는것이다.

3. **사용의 편리성**

stored procedure

- **함수 interface를 제공한다.**

```
DELIMITER //
CREATE PROCEDURE myFUNC()
BEGIN
  SELECT * FROM memberTBL WHERE memberName = '아이유';
    SELECT * FROM productTBL WHERE productName = '냉장고';
END //
-- DELIMITER
-- =>SQL은 ;으로 구분해서 // <- 이게 문장의 끝이라고 바꿔주는 명령어
DELIMITER ;

CALL myFUNC();
```

Trigger

💡 **Trigger 를 Table 에 부착시키면 `CREATE, INSERT, DELETE, UPDATE` 가 발생했을 때 trigger 가 자동적으로 동작한다.**

```
DELIMITER //
CREATE TRIGGER trg_deleteMemberTBL
  AFTER DELETE -- DELETE라는 SQL구문이 실행되면 그 이후에 실행하라는 의미이다.
  -- TRIGGER가 발동하는 시점을 명시.
  ON memberTBL -- memberTBL에서 삭제가 발생되면 이라는 뜻
    FOR EACH ROW -- 각각의 행들마다 트리거를 발생시켜라

BEGIN
  INSERT INTO deleteMemberTBL VALUES(
    OLD.memberID, OLD.memberName, OLD.memberAddress, CURDATE());
    -- CURDATE -> SQL이 제공하는 현재 시간을 알려주는 함수
    )
END //
DELIMITER ;
```

SQL

```
USE sqldb;

-- 1970년 이후에 출생하고 신장이 182인 사람의 아이디와 이름을 조회하세요!

SELECT userID, name FROM usertbl WHERE birthYear >= 1970 AND height >= 182;

-- 1970년 이후에 출생하거나 신장이 182인 사람의 아이디와 이름을 조회하세요!
```

```
SELECT userID, name FROM usertbl WHERE birthYear >= 1970 OR height >= 182;

-- 키가 180 ~ 183인 사람의 이름과 키를 조회하세요!
SELECT name, height FROM usertbl WHERE height >= 180 AND height <= 183;
SELECT name, height FROM usertbl WHERE height  BETWEEN 180 AND 183;

-- 지역이 경남, 전남, 경북인 사람의 이름과 지역만 조회하세요!
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');

-- 성이 김씨인 사람들의 이름과 키를 조사하세요!
SELECT name,height FROM usertbl WHERE name LIKE '김%';
-- "%"는 0개 이상의 글자를 지칭한다. "_"는 한글자를 지칭한다.

-- 김경호보다 키가 크거나 같은 사람의 이름과 키를 조회하세요!
SELECT name,height FROM usertbl WHERE height >= (SELECT height FROM usertbl WHERE name = '김경호');
-- 서브쿼리 이용하는 방법

-- 지역의 '경남'인 사람의 키보다
-- 키가 크거나 같은 사람을 조회하세요!
SELECT name,height FROM usertbl WHERE height >= (SELECT MIN(height) FROM usertbl WHERE addr = '경남');
-- 지역이 경남인 사람의 키중 최솟값 추출
SELECT name,height FROM usertbl WHERE height >= ANY(SELECT height FROM usertbl WHERE addr = '경남');
-- ANY 경남권인 사람의 키인 177이나 173보다 크거나 같은 경우의 로우들을 모두 불러오게 된다.

-- 가입한 순서대로 출력하세요!
SELECT name, mDate FROM usertbl ORDER BY mDate ASC ;

-- 회원들의 지역을 중복을 제거하고 출력하세요!
SELECT distinct addr FROM usertbl; -- distinct 중복배제

-- 먼저 가입한 순으로 4명만 출력하세요!
SELECT name, mDate FROM usertbl ORDER BY mDate ASC LIMIT 4 ;
-- LIMIT 상위 4명만 출력
SELECT name, mDate FROM usertbl ORDER BY mDate ASC LIMIT 1,2 ;
-- 0번부터 시작하는 1번째 부터 2개 들고오라는 뜻

-- 테이블을 복사하는 전형적인 방법
CREATE TABLE buytbl2(SELECT * FROM buytbl);
-- 기본키,외래키같은 제약조건들은 복사 안되고 data만 복사된다!

-- 구매테이블에서 각 사용자별로 구매한 물품의 개수를 출력하세요!
SELECT userID, SUM(amount)
FROM buytbl
GROUP BY userID;

-- 구매테이블에서 각 사용자별 구매액의 총합을 출력하세요!
SELECT userID, SUM(amount * price)
FROM buytbl
GROUP BY userID;
```