

# SQL Basic

---

# Database 개요

---

# 데이터베이스

---

- 데이터 베이스
  - 지속적으로 유지, 관리되어야 하는 데이터들의 집합
- 데이터베이스 관리 시스템
  - Database Management System (DBMS)
  - 데이터베이스를 관리해 주는 시스템을 말한다.
  - MySQL, Oracle, MS-SQL, DB2 등.

# 관계형 데이터 베이스(Relational Database)

- 관계형 데이터베이스(RDB)

- 행과 열로 이루어진 2차원 표 형식으로 Data를 관리하는 데이터베이스.

- 데이터를 관계 있는 여러 항목(열)의 집합으로 표현

- 이 데이터 집합을 관계(Relation)이라고 한다.

- 업무적 연관성 있는 테이블간의 관계를 통해 데이터를 관리하는 방식

ID		이름	나이
A120		홍길동	20
A121		이순신	24
A122		강감찬	23

열(Column, Attribute)

→ 행(ROW, Record, Tuple)

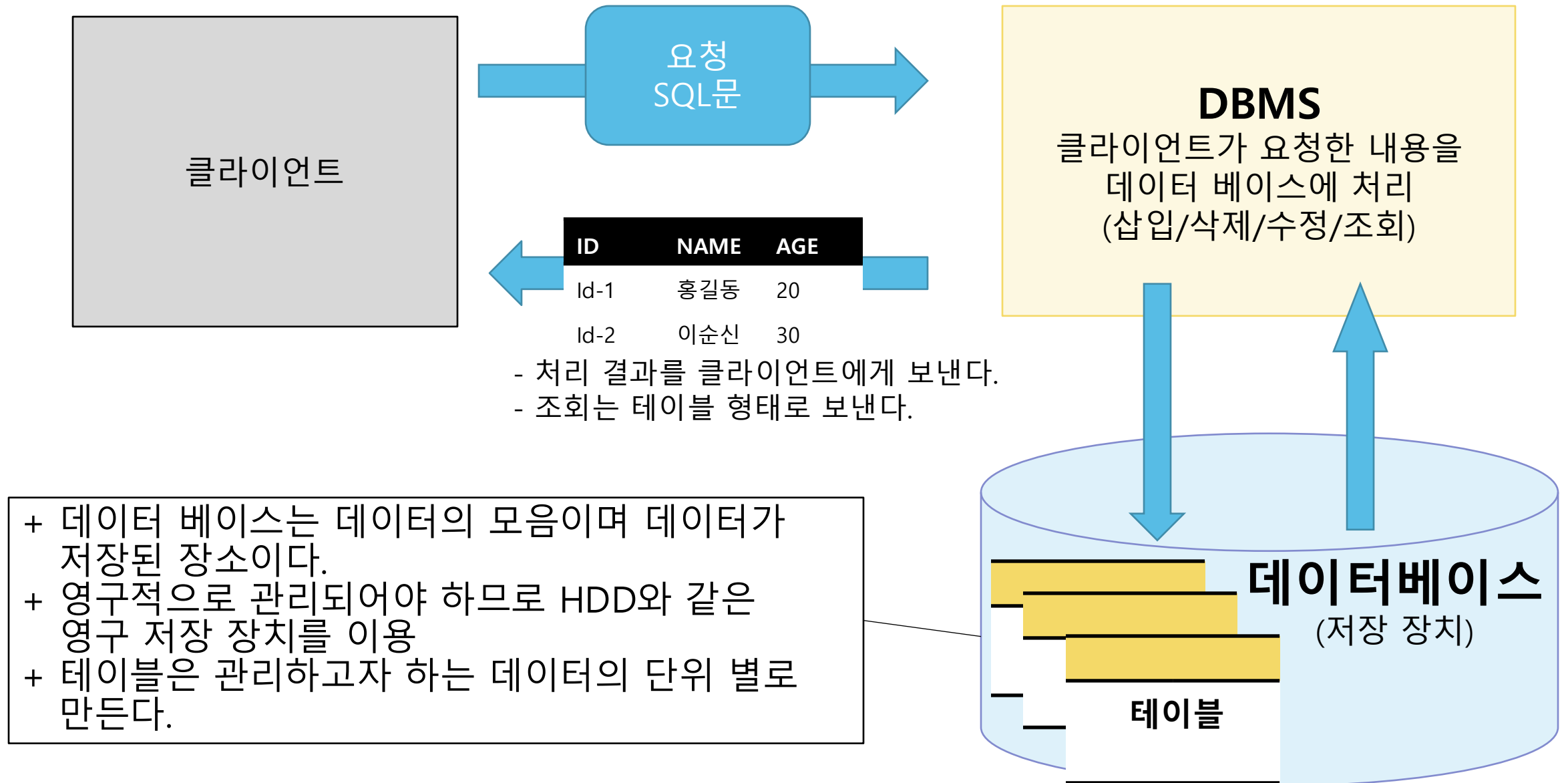
↘ 테이블(Table, Relation)

# 테이블(Table)

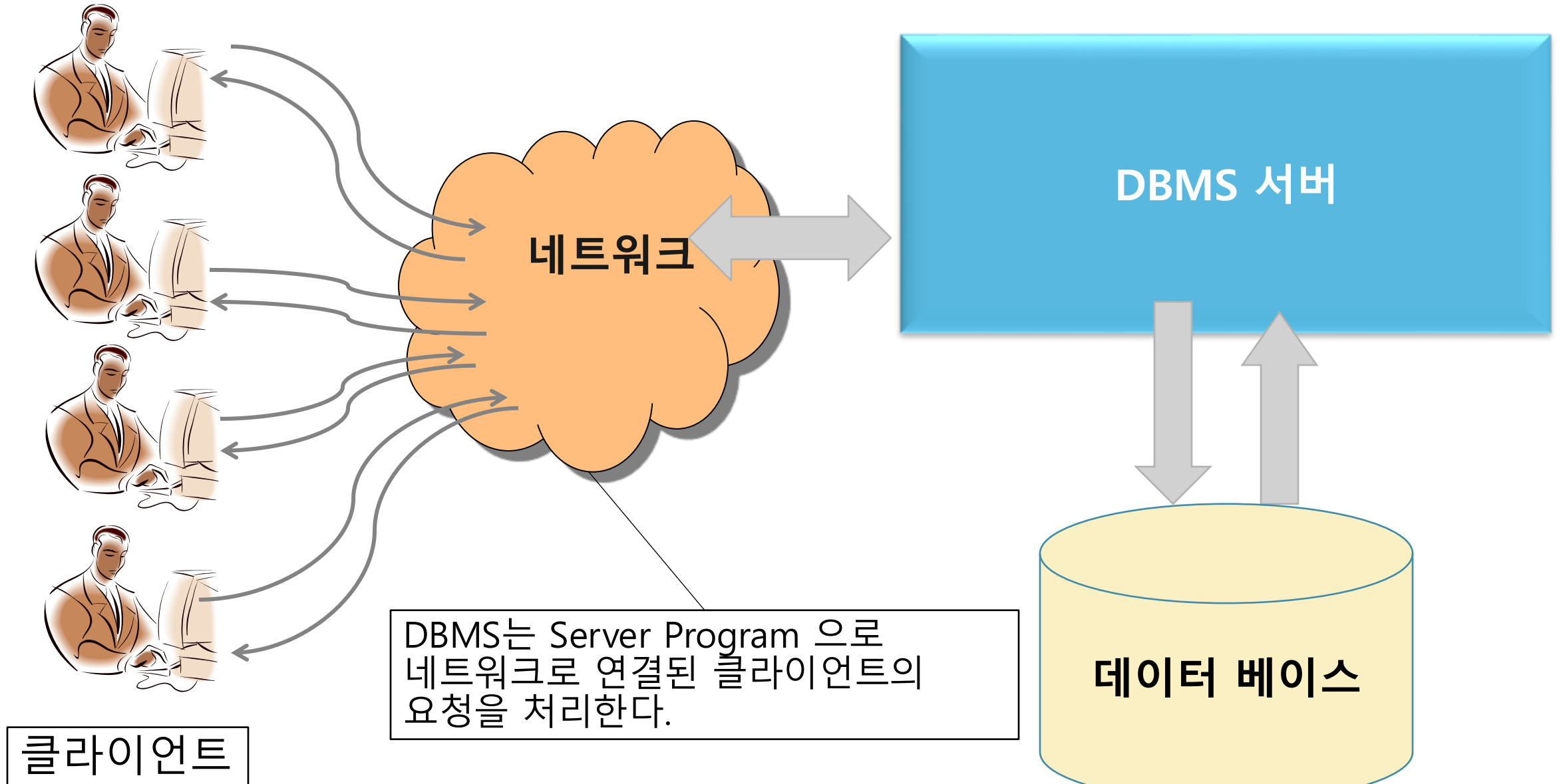
---

- 데이터 베이스에서 데이터를 저장하는 단위
  - Entity
    - 시스템이 독립적으로 관리하길 원하는 데이터
  - Table
    - Entity를 물리적 데이터베이스에 표현하는 방식
  - 열(Column)과 행(Row, Record) 의 이차원 표형식으로 관리한다.
- 열(Column, Attribute)
  - Data를 구성 하는 속성
- 행(Row, Record)
  - 하나의 Data

# DBMS 처리흐름



# DBMS 처리흐름 : Client -Server 구조



# 기본 SQL(Structured Query Language)

---



# SQL

- Structured Query Language
- 데이터베이스에 데이터를 질의, 등록, 수정, 삭제 등을 요청하기 위한 표준 언어.
  - DML(Data Manipulation Language) – INSERT, UPDATE, DELETE, SELECT
    - 데이터 조작하는데 사용하는 언어.
    - Table에 Data 삽입(추가), 수정, 삭제, 조회(선택)
    - DML은 트랜잭션이 발생하여 SQL문은 임시로 적용되며 commit을 이용해 완전 적용시키거나 Rollback을 이용해 처리를 취소할 수 있다.
  - DDL(Data Definition Language) – CREATE, ALTER, DROP, TRUNCATE
    - Database 스키마(Database, Table, View, Index등의 object)를 생성, 삭제, 수정하는 언어.
    - DDL은 트랜잭션이 발생하지 않아 명령어를 실행하면 Rollback 시킬 수 없다.
  - DCL(Data Control Language) – GRANT, REVOKE
    - 사용자에게 권한을 주거나 권한을 없애는 것과 같은 Data 접근을 제어하기 위한 언어.

DDL

---

# 사용자 계정 생성 및 권한 설정

- 사용자 계정 생성

- **CREATE USER** 계정명@HOST **IDENTIFIED BY** 'PASSWORD'

- HOST

- localhost : LOCAL 접속 계정

- '%' : 원격 접속 계정

- 예

- CREATE USER 'scott'@'localhost' IDENTIFIED BY 'tiger';

- CREATE USER 'scott'@'%' IDENTIFIED BY 'tiger';

- 초기 관리자 계정

- user name: root

- Password: 설치시 등록한 password

- 사용자 계정 삭제

- **DROP USER** '계정명'@host

- 예

- DROP USER 'scott'@'localhost';

- DROP USER 'scott'@'%';

- 권한 지정

- **GRANT** 권한 **ON** DB.TABLE **TO** 계정@host

- 예

- GRANT ALL PRIVILEGES ON \*.\* TO 'scott'@localhost;

- GRANT ALL PRIVILEGES ON \*.\* TO 'scott'@'%';

\*:table.\*:host

## 생성된 계정조회

```
SELECT host, user FROM mysql.user;
```

# Database 생성 및 조회

---

- 관리자 계정으로 들어간다.

- 생성

```
CREATE DATABASE 이름
```

- 조회

```
SHOW DATABASES
```

- 사용

```
USE 이름
```

# 테이블 생성

```
CREATE TABLE 테이블이름  
( 컬럼이름 datatype [제약조건]  
  [, 컬럼이름 datatype ...]  
  [, 제약조건]  
)
```

- SQL은 키워드나 테이블, 컬럼명의 경우 대소문자를 구분하지 않는다.
- 문자열 값인 경우는 구분

## ▪ 테이블명, 컬럼명 규칙

- 영문자, 숫자, \_ 만 가능하다.
- 첫 글자는 반드시 영문자여야 한다.

```
▪ CREATE TABLE DEPARTMENT(  
  DEPARTMENT_ID INT CONSTRAINT dept_pk_deptno PRIMARY KEY,  
  DEPARTMENT_NAME VARCHAR(30),  
  LOCATION VARCHAR(100)  
)
```

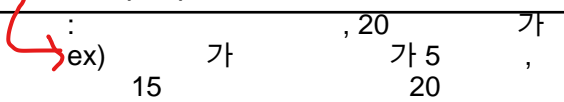
x, : x ->

## ▪ 데이터베이스에 생성된 테이블들 조회

- **show tables;**

# 테이블 생성 - 데이터 타입

## ▪ 문자열 타입

종류	Data Type	최대크기 (byte)	설명	비고
문자열 타입	CHAR(n)	0~255	고정 길이 문자열.	CHAR(크기), CHAR(20)
	VARCHAR(n)	0~65,535	가변 길이 문자열	 <b>VARCHAR</b> 는 MAX LENGTH를 지정할 수 있다. - VARCHAR(50): 최대 50글자까지 가능  TEXT 타입은 MAX LENGTH를 지정하지 않는다.
	TINYTEXT	0~255		
	TEXT	0~65,535		
	MEDIUMTEXT(n)	0 ~ 16,777,215		
	LONGTEXT(n)	0 ~ 4,294,967,295		

- **(n)** : 최대 길이(글자수)를 설정
- 길이 생략하면 1 (1글자) 로 설정 된다.
- **고정 길이**: 입력된 데이터의 글자 수가 모자라면 공백으로 채운다.
- **가변 길이**: 최초 지정된 길이는 최대 크기이며 입력된 데이터의 글자수에 따라 저장 크기가 변경되는 타입.
- 문자열(char, varchar) 의 값은 작은 따옴표로 감싼다. (큰 따옴표는 안됨)

# 테이블 생성 - 데이터타입

## ▪ Number 타입 ( )

종류	Data Type	크기	설명	비고
정수	TINYINT	1 byte	-128 ~ 127 (0 ~ 255)	( ) 는 unsigned 범위. TINYINT UNSIGNED 로 지정
	BOOLEAN	1 byte	TRUE: 1, FALSE: 0	TINYINT(1) 형식으로 설정되며 입력 시 TRUE, FALSE 키워드로 입력하나 조회시에는 1, 0이 조회된다.
	SMALLINT	2 byte	-32,768 ~ 32,767 (0 ~ 65535)	
	INT	4 byte	-2,147,483,648 ~ 2,147,483,647 (0 ~ 4,294,967,295)	
	BIGINT	8 byte	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 (0 ~ 18,446,744,073,709,551,615)	
실수	DECIMAL	8 byte	고정 소수	DECIMAL(M, N) - M: 총 자릿수, N: 소수 자릿수 - 계산 시 정확도가 요구되는 실수 데이터에 사용한다.
	FLOAT	4 byte	부동 소수	
	DOUBLE	8 byte	부동 소수	

# 테이블 생성 - 데이터타입

## ▪ 날짜형

종류	Data Type	최대크기	설명	비고
날짜 시간	DATE	3 byte	년,월,일	YYYY-MM-DD
	TIME	3 byte	시,분,초	hh:mm:ss
	DATETIME	8 byte	년,월,일,시,분,초 1000/01/01:00:00:00 ~ 9999/12/31 23:59:59	- YYYY-MM-DD hh:mm:ss - 입력된 일시는 system time zone에 상관없이 고정
	TIMESTAMP	4byte	년,월,일,시,분,초 1970/01/01 00:00:00 ~ 2038/01/19 03:14:07(UTC)	- YYYY-MM-DD hh:mm:ss - Sytem의 time zone을 변경하면 입력된 일시가 변경된다
	YEAR	1 byte		YYYY

## ▪ 결측치

- NULL
- 없는 값, 모르는 값, 수집되지 않은 값.



# 테이블 생성 - 제약조건 (Key)

- 컬럼(열)이 가질 수 있는 값에 대한 제약조건을 지정할 때 사용한다.
- 기본 구문
  - 컬럼 설정 시 지정
  - 컬럼 설정 다음에 따로 설정
    - constraint 제약조건이름 제약조건 (컬럼)
- 종류

제약조건	설명
PRIMARY KEY(PK)	하나의 행(Row)을 대표하는 열(컬럼). NOT NULL과 UNIQUE 조건을 만족한다.
FOREIGN KEY(FK)	컬럼과 참조하는 테이블의 컬럼 사이의 연결 관계를 설정.
UNIQUE KEY(UK)	테이블의 모든 행이 다른 값을 가져야 하는 컬럼. (NULL은 제외)      PK                      가
NOT NULL(NN)	NULL을 값으로 가질 수 없는 컬럼. 즉 반드시 값을 가져야 하는 열
CHECK(CK)	컬럼에 들어는 값의 조건을 지정 (대부분 업무 규칙을 설정)
AUTO_INCREMENT	자동 증가 정수 컬럼(MY SQL에서 사용). 정수컬럼으로 행이 입력되면 1씩 증가하는 값을 가진다.

# 테이블 생성 예

컬럼명(한글)	컬럼명(영문)	Data type	길이	NULL 허용	KEY
아이디	id	int		X	PK
이메일	email	varchar	10	O	UK
성별	gender	char	1	X	CK

```
CREATE TABLE member (  
  id      int      AUTO_INCREMENT PRIMARY KEY,  
  email   varchar(10) DEFAULT 'None'   UNIQUE,  
  gender  char      CHECK (gender in ('m', 'f'))  
);
```

```
CREATE TABLE member (  
  id      int      AUTO_INCREMENT,  
  email   varchar(10) DEFAULT 'None',  
  gender  char      NOT NULL,  
  CONSTRAINT pk_id    PRIMARY KEY (id),  
  CONSTRAINT uk_email  UNIQUE(email) ,  
  CONSTRAINT ck_gender CHECK (gender in ('m', 'f'))  
);
```

# 테이블 삭제

- 구문

**DROP TABLE 테이블이름**

- 삭제 하면 되돌릴 수 없다.

- 예)

- DROP TABLE MEMBER;

- DROP TABLE IF EXISTS MEMBER;

- 자식테이블에서 참조되고 있는 부모테이블을 삭제

- 자식테이블들을 먼저 삭제한다.

- Foreign key check 옵션을 끄고 해야 한다.

**set foreign\_key\_checks = 0 --옵션 끄기 (1을 주면 옵션을 켜다.)**

# 테이블의 컬럼이나 제약조건 수정 - ALTER

## ■ 컬럼 및 제약조건 추가

### - 컬럼 추가

```
ALTER TABLE 테이블이름 ADD COLUMN(컬럼명 DATA_TYPE [제약조건])
```

### - 제약조건 추가

```
ALTER TABLE 테이블이름 ADD CONSTRAINT 제약조건구문
```

- 예) ALTER TABLE employee ADD (address VARCHAR(100))

## ■ 컬럼 타입 변경

```
ALTER TABLE 테이블이름 MODIFY COLUMN 변경할컬럼명 DATA_TYPE [제약조건]
```

- 데이터가 존재하는 경우 변경할 수 없다. 단 VARCHAR, CHAR의 경우 변경하려는 크기가 더 큰 경우는 가능하다.

- 예) ALTER TABLE employee MODIFY (address VARCHAR(200))

# 테이블 수정

- 컬럼 삭제

**ALTER TABLE** 테이블이름 **DROP COLUMN** 삭제할컬럼이름

– 예) ALTER TABLE employee DROP COLUMN address

- 제약조건 삭제

**ALTER TABLE** 테이블이름 **DROP CONSTRAINT** 삭제할제약조건이름

– 예) ALTER TABLE employee DROP CONSTRAINT fk\_emp\_dept

– 예) ALTER TABLE employee DROP PRIMARY KEY

DML

---

# INSERT (데이터 삽입)

## ▪ INSERT 기본구문

**INSERT INTO** 테이블이름 (컬럼명, 컬럼명 [...]) **VALUES** (값1, 값2 [...])

- 예) **INSERT INTO** DEPARTMENT (DEPARTMENT\_ID, DEPARTMENT\_NAME, LOCATION)  
      **VALUES** (100, '기획부', '서울')
- INSERT는 한 행(Row, 레코드) 씩 처리한다.
- 문자열의 경우 삽입할 값을 ' '로 감싸준다.
- 날짜는 형태에 맞게 문자열로 넣어준다. ( 날짜: - 나 / 로 구분, 시간 : 로 구분)
- 결측치값은 **null** 키워드를 입력한다.
- 테이블의 **모든 컬럼에 데이터를 넣을 경우 컬럼 항목은 생략**할 수 있다.

# UPDATE (데이터 수정), DELETE (데이터 삭제)

- **UPDATE 기본구문**

**UPDATE** 테이블이름  
**SET** 컬럼=변경할값 [, 컬럼=변경할값]  
**[WHERE]** 제약조건

– 예) **UPDATE** EMPLOYEE **SET** HIREDATE = SYSDATE

**UPDATE** EMPLOYEE **SET** SALARY = SALARY \* 1.1 **WHERE** EMPLOYEE\_ID = 120

**UPDATE** DEPARTMENT **SET** LOCATION = '부산' **WHERE** DEPARTMENT\_ID > 100

- **DELETE 기본구문**

**DELETE FROM** 테이블이름 **[WHERE 제약조건]**

– 예) **DELETE FROM** DEPARTMENT **WHERE** DEPARTMENT\_ID = 100

**DELETE FROM** EMPLOYEE **WHERE** SAL < 5000000

**DELETE FROM** DEPARTMENT



# SELECT (조회)

- SELECT 기본구문

```
SELECT 조회컬럼 [별칭][, 조회컬럼,...]  
FROM 테이블이름 [별칭]  
[WHERE 제약조건]  
[GROUP BY 그룹화할 기준컬럼]  
[HAVING 조건]  
[ORDER BY 정렬기준컬럼 [ASC | DESC]]
```

- 항목

- **SELECT 절** : 조회할 컬럼들 지정. 모든 컬럼 조회시 \* 사용. 별칭(alias)-조회 결과의 컬럼명(별칭)
- **FROM 절** : 조회대상 테이블이름. 별칭 - 테이블이름 대신 쿼리 내에서 사용할 별칭
- **WHERE 절** : 조회할 행에 대한 선택 조건.
- **GROUP BY 절** : 집계결과 조회 시 어떤 컬럼의 값이 같은 것 끼리 묶어서 조회할지 지정
- **HAVING 절** : 집계결과 조회 시 그 결과에 대한 조회조건을 넣는다.
- **ORDER BY** : 조회결과 정렬. (ASC:오름차순(Default), DESC내림차순)

# SELECT (조회)

- 예제

```
SELECT * FROM employee
```

```
SELECT    department_name 부서명, location 위치  
FROM      department  
WHERE     location= '서울'  
ORDER BY  department_no DESC
```

```
SELECT    department_no, avg(salary)  
FROM      employee  
GROUP BY  department_no  
HAVING    avg(salary) > 30000000
```

# 연산자

- 컬럼이나 상수값에 사칙연산을 이용할 수 있다.
  - SELECT 조회 컬럼에 사용시 연산은 **행 단위로 이루어진다.**
  - 연산자
    - $+$ ,  $-$ ,  $*$ ,  $/$
    - $\%$ , MOD : 나머지 연산자
    - DIV: 몫 연산자
    - 연산자 우선순위지정은 ( ) 로 묶는다.
- 예)
  - **SELECT tall + 20 FROM user;**
  - **SELECT tall - weight FROM user;**
  - **SELECT salary \* bonus\_rate FROM employee;**
  - **SELECT tall / weight FROM user;**
  - **SELECT tall DIV weight FROM user;**
  - **SELECT tall MOD weight FROM user; or SELECT tall % weight FROM user;**
  - **SELECT \* FROM user WHERE tall/100 > 1.7;**

# WHERE 절에서 사용하는 검색 조건의 주요 연산자

연산자	설명
AND, OR	논리 연산자로 조건이 하나 이상일 경우 연결 연산자. AND : 두 조건을 모두 만족하는 것. OR : 둘 중 하나만 만족하는 것
=, <> (!=), >, <, >=, <=	= : 같은 것 조회, != 같지 않은 것 조회 > : 큰 값들 조회, < : 작은 값들 조회
BETWEEN a AND b	a와b사이의 데이터를 조회(a, b값 포함)
IN (list)	list(를 구분자로 나열)의 값 중 어느 하나와 일치하는 데이터를 조회
LIKE	문자 형태로 부분일치하는 데이터를 조회 (% , _사용)
IS NULL	NULL값을 가진 데이터를 조회
NOT BETWEEN a AND b	a와b사이에 있지 않은 데이터를 조회(a, b값 포함하지 않음)
NOT IN (list)	list의 값과 일치하지 않는 데이터를 조회
NOT LIKE	문자 형태와 일치하지 않는 데이터를 조회
IS NOT NULL	NULL값을 갖지 않는 데이터를 조회

- where 절은 update, delete, select 에서 행을 선택할 때 사용한다.
- 조건이 여러 개인 경우 and 나 or 로 연결한다.
- 연산의 우선순위를 바꿀 경우 ( ) 를 사용한다.

함수

---

# 함수 종류

---

## ▪ 단일 행 함수

- 행 단위로 값을 처리하는 함수.
- 단일행은 select, where 절에서 사용 가능.
- 함수에 함수를 넣어 여러 처리를 한번에 할 수 있다.
  - **CHAR\_LENGTH(CONCAT('A','B'))**

## ▪ 다중 행 함수

- 여러 행의 값들을 묶어서 한번에 처리하는 함수
- 집계함수, 그룹함수라고 한다.
- 다중행은 select, having 절에서 사용 가능.
  - where절에는 사용할 수 없다. (subquery 이용)

# 함수 – 조건 처리 함수

---

- if (조건수식, 참, 거짓)
  - 조건수식이 True이면 참을 False이면 거짓을 출력한다.
- ifnull (기준컬럼(값), 기본값)
  - 기준컬럼(값)이 NULL값이면 기본값을 출력하고 NULL이 아니면 기준컬럼 값을 출력
- nullif(컬럼1, 컬럼2)
  - 컬럼1과 컬럼2가 같으면 NULL을 반환, 다르면 컬럼1을 반환

# 함수 – 조건 처리 함수

- CASE 구문(함수가 아닌 연산자임)
  - 동등 조건 비교
    - **CASE** 수식 **WHEN** 비교값1 **THEN** 출력값1 ... **ELSE** 출력값2 **END**
      - 수식의 값이 비교값1과 같으면 출력값2을 출력한다.
        - 조건이 여러 개일 경우 **WHEN** 비교값 **THEN** 출력값 을 반복한다.
      - 모든 비교값과 일치하지 않으면 ELSE의 출력값2을 출력한다.
        - ELSE는 생략가능 하며 생략시 NULL 출력한다.
    - 조건 비교
      - **CASE WHEN** 수식1 **THEN** 출력값1 ... **ELSE** 출력값2 **END**
        - 수식1의 조건이 True이면 출력값 1을 출력한다.
          - 조건이 여러 개일 경우 **WHEN** 비교값 **THEN** 출력값 을 반복한다.
        - 모든 조건이 False일 경우 출력값2를 출력한다.
          - ELSE는 생략가능 하며 생략시 NULL 출력한다.



# 함수 – 문자열 처리 함수

---

- 함수에 전달하는 값이 문자열이거나 처리결과가 문자열인 함수들.
- `char_length` (문자열)
  - 글자수 반환
- `concat`(값1, 값2, ...), `concat_ws`(구분자, 값1, 값2, ...)
  - 전달된 문자열들을 합친다.
  - `CONCAT_WS()`는 첫번째로 전달한 구분자를 이용해 합친다.
- `format`(숫자, 소수부자리수)
  - **정수부분**: 단위 구분자 **십표 구분자(,)**를 표시
  - **소수부분**: 지정한 자리까지 출력한다. 적을 경우 0으로 채우고 더 많을 경우 반올림한다.
- `upper`(문자열), `lower`(문자열)
  - **문자열**의 소문자를 대문자로(`UPPER`), 대문자를 소문자로(`LOWER`)로 변환.

# 함수 – 문자열 처리 함수

- insert(기준문자열, 위치, 길이, 삽입문자열)
  - **기준문자열**의 위치(**1부터 시작**)에서부터 **길이**까지 지우고 **삽입문자열**을 넣는다.
- replace(기준문자열, 원래문자열, 바꿀문자열)
  - **기준문자열**의 **원래문자열**을 **바꿀문자열**로 바꾼다.
- left(기준문자열, 길이), right(기준문자열, 길이)
  - **기준문자열**에서 왼쪽(left), 오른쪽(right)의 **길이**(개수)만큼의 문자열을 잘라서 반환한다.
- substring(기준문자열, 시작위치, 길이)
  - **기준문자열**에서 **시작위치**부터 **길이 개수**의 글자 만큼 잘라서 반환한다.
  - 길이를 생략하면 마지막까지 잘라낸다.
- substring\_index(기준문자열, 구분자, 개수)
  - **기준문자열**을 **구분자**를 기준으로 나눈 뒤 **개수**만큼 반환.
  - **개수**: 양수 – 앞에서 부터 개수, 음수 – 뒤에서 부터 개수만큼 반환

# 함수 – 문자열 처리 함수

- ltrim(문자열), rtrim(문자열), trim(문자열)
  - 문자열에서 왼쪽(ltrim), 오른쪽(rtrim), 양쪽(trim)의 공백을 제거한다. 중간공백은 유지
- trim(방향 제거할문자열 from 기준문자열)
  - 기준문자열에서 방향에 있는 제거할문자열을 제거한다.
  - 방향: **both** (앞,뒤), **leading** (앞), **trailing** (뒤)
  - 예)
    - SELECT trim(both 'a' from 'aaa안녕aaa'); => '안녕'
- lpad(기준문자열, 길이, 채울문자열), rpad(기준문자열, 길이, 채울문자열)
  - 기준문자열을 길이만큼 늘린 뒤 남은 길이만큼 채울문자열로 왼쪽(lpad), 오른쪽(rpad)에 채운다.
  - 기준문자열 글자수가 길이보다 많을 경우 나머지는 자른다.

# 함수 – 숫자(수학) 처리 함수

- ceil(실수), floor(실수)
  - 올림, 내림, 반올림 한 정수를 반환
- round(실수, 자릿수), truncate(기준숫자, 자릿수)
  - **기준숫자**에서 지정한 **자릿수** 이하에서 반올림(round)/버린다(truncate).
  - 자릿수가 양수이면 소수부분 자릿수를 음수이면 정수부분 자릿수가 된다.
  - truncate(12345.12345, 2) -> 12345.12,
  - truncate(12345.12345, -2) -> 12300
- mod(n1, n2)
  - modular 연산(나머지 연산).  **$n1 \% n2$**
- abs(n)
  - 숫자 **n**의 절대값을 반환
- sign(n)
  - 숫자 **n**의 부호를 정수로 반환.
  - 반환 값: 1(양수), 0, -1(음수)

# 함수 – 날짜, 시간 처리 함수

---

- curdate()
  - 실행 시점 날짜(년-월-일) 반환
- curtime()
  - 실행 시점의 시간(시:분:초) 반환
- now()
  - 실행 시점의 일시(년-월-일 시:분:초) 반환
- year(날짜), month(날짜), day(날짜)
  - 날짜 또는 일시의 **년, 월, 일** 을 반환한다.
- hour(시간), minute(시간), second(시간), microsecond(시간)
  - 시간 또는 일시의 **시, 분, 초, 밀리초**를 반환한다.
- date(), time()
  - datetime 에서 날짜(date), 시간(time)만 추출한다.

# 함수 – 날짜, 시간 처리 함수

- `adddate(기준일시, 더할 구문), subdate(기준일시, 뺄 구문)`
  - **기준일시**에서 더할/뺄 구문에 맞춰 계산된 날짜를 반환한다.
  - 더할/뺄 구문
    - **interval** 날짜 단위
      - 단위: microsecond, second, minute, hour, day, week, month, quarter(분기-3개월), year
    - `SELECT adddate('2010-10-20', interval 20 day)`
- `addtime(기준일시, 더할시간), subtime(기준일시, 뺄시간)`
  - **기준일시**에서 더할시간/뺄시간 만큼 계산한 날짜시간을 반환한다.
  - 더할/뺄 시간은 '시:분:초' 형식으로 지정한다.
  - `SELECT addtime('11:20:30', '1:0:0');` => 12:20:30
- `timestampdiff(옵션, 일시1, 일시2)`
  - 일시2 - 일시1 을 **옵션**을 기준으로 계산한다.
  - 옵션: microsecond, second, minute, hour, day, week, month, quarter(분기-3개월), year
- `datediff(날짜1, 날짜2)`
  - **날짜1 - 날짜2**한 일수를 반환
- `timediff(시간1, 시간2)`
  - 시간1-시간2 한 시간을 계산해서 반환 (뺀 결과를 시:분:초 로 반환)

# 함수 - 날짜, 시간 처리 함수

---

- `dayofweek(날짜)`
  - 날짜의 요일을 정수로 반환 (1: 일요일 ~ 7: 토요일)
- `monthname(날짜)`, `dayofyear(날짜)`
  - 날짜의 월 단어(`monthname`), 일년 중 몇 번째 날짜인지 반환
- `last_day(날짜)`
  - 날짜의 마지막 날짜를 반환
  - `last_day(now())` -> 1월일 경우 **2020-01-31**
- `quarter(날짜)`
  - 날짜가 4분기 중 몇 분기인지를 정수로 반환

# 함수 – 날짜, 시간 처리 함수

- date\_format(일시, 출력형식문자열)

- 일시를 출력형식문자열에 맞는 문자열로 변환해서 반환한다.
- 출력형식(format) 명시자(specifier)

- [https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html#function\\_date-format](https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html#function_date-format)

```
SELECT date_format(now(), '%Y년 %m월 %d일')
```

명시자	설명
%Y	연도 4자리
%m	월 2자리 (01 ~ 12)
%d	일 2자리 (01 ~ 31)
%H	시간 (00 ~ 23)
%h	시간 (00 ~ 12)
%i	분 2자리 (00 ~ 59)
%S	초 2자리 (00 ~ 59)
%p	AM, PM
%W	요일(단어)
%w	요일 정수 (0: 일요일, 1: 토요일)



# 함수 - 형 변환 처리

- 형 변환이란? (Type casting)
  - 값의 타입을 다른 타입으로 변환
  - 암시적 변환
    - 구문에 맞춰 MySQL 서버가 알아서 변환한다.
    - ex)
      - select '1000' + '2000'; 문자열 '1000', '2000'을 정수로 변환 후 더한다.
      - select concat(322, '개'); 322 정수를 문자열로 변환한 뒤 붙인다.
  - 명시적 변환
    - 변환함수를 이용해 명시적으로 변환한다.
    - **cast**(값 as 변환할 타입)
    - **convert**(값, 변환할 타입)
      - 값을 변환할 타입으로 변환한다.
      - 두 함수는 구문만 차이가 있다.

변환할 타입
date
time
datetime
decimal
char
signed (부호있는 64bit 정수)
unsigned (부호 없는 64bit 정수)
binary

# 함수 - 집계함수

- 조회 결과를 묶어 집계 처리하는 함수들로 **그룹함수, 다중행 함수**라고도 한다.
  - 기본적으로 전체 행을 기준으로 계산한다.
  - select시 group by 절을 이용해 그룹으로 묶을 기준 컬럼을 지정할 수 있다.
    - 지정한 컬럼의 값이 같은 행끼리 하나의 그룹으로 묶인다.
- 집계함수
  - avg(column) : 평균 값 조회
  - sum(column) : 합계 값 조회
  - max(column) : 제일 큰 값 조회
  - min(column) : 가장 작은 값 조회
  - count(column) : 행 수 조회 (개수 조회)
  - count(distinct column): 행 수 조회(개수 조회). 중복된 값은 1개로 처리.

# 함수 - 집계함수

---

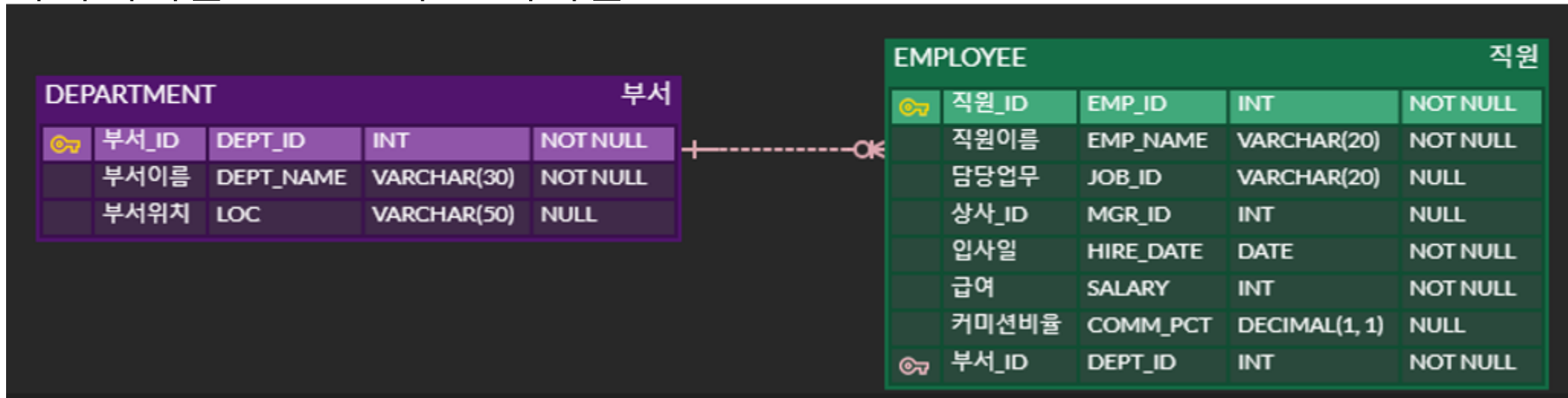
- with rollup
  - 총합, 중간 합계를 조회결과에 넣는다.
  - group by 절과 함께 사용

# 조인(JOIN)

---

# 테이블 간의 관계 : Foreign key 제약조건

- Foreign key (외래키)
  - 다른 테이블의 PK값만 가질 수 있는 컬럼
  - 테이블간의 연관관계를 표현한다.
- 부모테이블 – 참조 되는 테이블
- 자식테이블 – 참조 하는 테이블



- 부서테이블: 부모테이블, 직원테이블: 자식테이블
- 직원테이블의 데이터(ROW)는 부서테이블의 데이터를 참조한다.
- 부서테이블의 ROW는 직원테이블의 데이터 0개~N개가 참조할 수 있다.

# Foreign key 제약조건

- 기본 구문

CONSTRAINT 제약조건이름 FOREIGN KEY(컬럼) REFERECES 부모테이블(PK컬럼) [ON 설정]

- 자식 테이블로부터 참조 당하는 부모 테이블의 row(행)는 삭제 할 수 없다.

- CASCADE DELETE

- FOREIGN KEY 설정 시 ON DELETE CASCADE 설정

- 부모 테이블의 참조 ROW 삭제 시 자식 테이블의 참조 ROW 삭제

CONSTRAINT 제약조건이름 FOREIGN KEY(컬럼)  
REFERECES 부모테이블(PK컬럼) ON DELETE CASCADE

- FOREIGN KEY 설정 시 ON DELETE SET NULL설정

- 부모 테이블의 참조 ROW 삭제 시 참조 하는 자식 테이블의 컬럼 값을 NULL로 설정

CONSTRAINT 제약조건이름 FOREIGN KEY(컬럼)  
REFERECES 부모테이블(PK컬럼) ON DELETE SET NULL

# Foreign key 제약조건

## ▪ 기본 구문

CONSTRAINT 제약조건이름 FOREIGN KEY(컬럼) REFERENCES 부모테이블(참조컬럼) [ON 설정]

- 참조컬럼은 PK거나 Index 설정된 컬럼이어야 한다.

## ▪ ON 설정

- 기본적으로 자식테이블에서 참조하고 있으면 부모 테이블의 데이터를 삭제/수정하지 못한다.
  - Foreign key 컬럼 설정할 때 이것을 변경할 수 있는데 이때 on 설정을 사용한다.
- **ON DELETE|UPDATE 처리방식**
- ON DELETE: 참조하는 부모테이블의 행이 삭제 되었을 때 어떻게 처리할 것인지 설정
- ON UPDATE: 참조하는 부모테이블의 참조 컬럼값이 변경되면 어떻게 처리할 것인지 설정
- 처리방식
  - CASCADE : 부모 테이블에서 데이터를 삭제/수정하면, 그 행을 참조하는 자식테이블의 행들도 같이 삭제/수정된다.
  - SET NULL : 부모 테이블에서 데이터를 삭제/수정하면, 그 행을 참조하는 자식테이블의 foreign key 컬럼의 값을 NULL로 변경한다.

# Foreign key 제약조건

- Table drop시 foreign key 제약조건 제거
  - 자식테이블이 있는 테이블을 삭제할 때는 먼저 참조관계를 끊어줘야 삭제가 가능하다.
  - 부모 테이블 DROP전에 MySQL의 foreign key 적용이 안되도록 설정해야 한다.

```
set foreign_key_checks = 0;  -- foreign key 해제 설정
DROP table 삭제할테이블;
set foreign_key_checks = 1;  -- foreign key 적용 설정
```



# 조인 개념

- 연관관계가 있는 두개의 테이블을 합쳐서(JOIN) 조회(SELECT) 하는 것을 말한다.
- 조인할 두 테이블의 특정 컬럼의 값들이 같은 행끼리 합친다.
- 조인연산
  - 어떤 행들의 값이 같은 것끼리 합칠 것인지 설정하는 구문.
  - 일반적으로 부모테이블과 자식테이블을 연결해서 조회하는 경우가 많으며 이 경우 부모 테이블의 PK와 자식 테이블의 FK 컬럼의 값이 같은 행들을 JOIN 한다.

부서		
부서_ID	부서이름	지역
100	기획부	서울
200	총무부	서울
300	구매부	부산

직원		
직원_ID	직원이름	소속부서
E01	김영수	100
E02	박영희	200
E03	오민기	100

부서-직원 조인					
직원_ID	직원이름	소속부서	부서_ID	부서이름	지역
E01	김영수	100	100	기획부	서울
E02	박영희	200	200	총무부	서울
E03	오민기	100	100	기획부	서울

# 조인 개념

---

- 조인 종류
  - INNER JOIN
    - 조인 연산 조건을 만족하는 행들을 합친다.
  - OUTER JOIN
    - 조인 연산 조건을 만족하지 않는 행들도 포함해서 합친다.
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN

# INNER JOIN(내부조인)

- 조인 연산 조건을 만족하는 행끼리 만 합치는 JOIN

- 부서\_ID가 300인 부서는  
직원 테이블의 어떤 행 과도 관계가  
없으므로 조인 대상 포함되지 않음

부서		
부서_ID	부서이름	지역
100	기획부	서울
200	총무부	서울
300	구매부	부산

직원		
직원_ID	직원이름	소속부서
E01	김영수	100
E02	박영희	200
E03	오민기	100

- 구문

부서-직원 조인					
직원_ID	직원이름	소속부서	부서_ID	부서이름	지역
E01	김영수	100	100	기획부	서울
E02	박영희	200	200	총무부	서울
E03	오민기	100	100	기획부	서울

```
SELECT 컬럼선택
FROM 테이블_1 INNER JOIN 테이블_2 ON 조인연산
      [INNER JOIN 테이블_3 ON 조인연산 ...]
```

-- INNER 생략가능

```
SELECT b.부서_ID, b.부서이름,
       e.직원이름
FROM 부서 b INNER JOIN 직원 e ON b.소속부서=e.부서_ID
```

# 조인 – 외부조인(Outer Join)

- 조인 연산 조건을 만족하지 않는 행도 포함해서 합친다.
- Source Table
  - 조인시 조회 기준이 되는 테이블로 조회하려는 주(main) 정보를 가지고 있는 테이블
- Target Table
  - 조인시 Source table 데이터의 추가 정보를 가지고 있는 테이블로 보조(sub) 정보를 가지고 있는 테이블
- 조인시 Source 테이블의 데이터(행)은 모두 사용하고 Target table의 데이터(행)은 조인 조건을 만족하는 행만 나오도록 한다.
- Outer Join 종류
  - LEFT OUTER JOIN
    - Source 테이블이 왼쪽에 선언한 테이블인 경우의 outer join.
  - RIGHT OUTER JOIN
    - Source 테이블이 오른쪽에 선언한 테이블인 경우의 outer join.
  - FULL OUTER JOIN
    - 양쪽에 선언한 테이블이 모두 Source table인 경우의 outer join
    - MySQL은 지원하지 않는다.

# 조인 – 외부조인(Outer Join)

## ▪ LEFT OUTER JOIN

부서		
부서_ID	부서이름	지역
100	기획부	서울
200	총무부	서울
300	구매부	부산

직원		
직원_ID	직원이름	소속부서
E01	김영수	100
E02	박영희	200
E03	오민기	100
E04	조진수	NULL



부서-직원 조인 (Source: 부서, Target: 직원)					
부서_ID	부서이름	지역	직원_ID	직원이름	소속부서
100	기획부	서울	E01	김영수	100
100	기획부	서울	E03	오민기	100
200	총무부	서울	E02	박영희	200
300	구매부	부산	NULL	NULL	NULL

**SELECT** 컬럼선택

**FROM** 테이블\_1 **LEFT [OUTER] JOIN** 테이블\_2 **ON** 조인연산

-- **OUTER** 생략가능

- Source table: 테이블\_1, Target table: 테이블\_2

SELECT b.부서\_ID, b.부서이름,  
e.직원이름

FROM 부서 b **LEFT OUTER JOIN** 직원 e **ON** b.소속부서=e.부서\_ID

# 조인 – 외부조인(Outer Join)

## ▪ RIGHT OUTER JOIN

부서		
부서_ID	부서이름	지역
100	기획부	서울
200	총무부	서울
300	구매부	부산

직원		
직원_ID	직원이름	소속부서
E01	김영수	100
E02	박영희	200
E03	오민기	100
E04	조진수	NULL



직원-부서 조인 (Source: 직원, Target: 부서)					
직원_ID	직원이름	소속부서	직원_ID	직원이름	소속부서
E01	김영수	100	100	기획부	서울
E02	박영희	200	200	총무부	서울
E03	오민기	100	100	기획부	서울
E04	조진수	NULL	NULL	NULL	NULL

**SELECT** 컬럼선택

**FROM** 테이블\_1 **RIGHT OUTER JOIN** 테이블\_2 **ON** 조인연산

-- **OUTER** 생략가능

- Source table: 테이블\_2, Target table: 테이블\_1

SELECT b.부서\_ID, b.부서이름,  
e.직원이름

FROM 부서 b **RIGHT OUTER JOIN** 직원 e **ON** b.소속부서=e.부서\_ID

# 조인 – 외부조인(Outer Join)

## ▪ FULL OUTER JOIN

- MySQL은 지원하지 않는다.
- left join, right join select 문을 union 하여 구현한다.

부서		
부서_ID	부서이름	지역
100	기획부	서울
200	총무부	서울
300	구매부	부산

직원		
직원_ID	직원이름	소속부서
E01	김영수	100
E02	박영희	200
E03	오민기	100
E04	조진수	NULL



부서-직원 조인 (Source: 부서, 직원)					
부서_ID	부서이름	지역	직원_ID	직원이름	소속부서
100	기획부	서울	E01	김영수	100
100	기획부	서울	E03	오민기	100
200	총무부	서울	E02	박영희	200
300	구매부	부산	NULL	NULL	NULL
NULL	NULL	NULL	E04	조진수	NULL

```
SELECT b.부서_ID, b.부서이름, e.직원이름
FROM 부서 b LEFT OUTER JOIN 직원 e ON b.소속부서=e.부서_ID
UNION
SELECT b.부서_ID, b.부서이름, e.직원이름
FROM 부서 b RIGHT OUTER JOIN 직원 e ON b.소속부서=e.부서_ID;
```

서브쿼리(Subquery)

---



# 서브쿼리(Subquery)

---

- 개요 : SQL 문 내 SELECT문을 삽입하는 것.
  - SQL문에서 SELECT문으로 조회한 결과를 이용하는 것.
  - 주로 FROM절이나 WHERE 절에서 사용
  - 메인 쿼리
    - 실제 조회하고자 하는 쿼리
  - 서브 쿼리
    - 메인 쿼리에서 사용할 데이터를 조회하기 위한 쿼리
  - 종류
    - 단일 행 서브 쿼리 - 서브쿼리의 조회결과가 한 행.
    - 다중 행 서브 쿼리 - 서브쿼리의 조회결과가 여러 행.
    - Inline View - 서브쿼리를 FROM절에 TABLE 대신 사용

# 서브쿼리(Subquery)

## ▪ 단일 행 서브쿼리

- 서브 쿼리의 조회결과가 0개 또는 1개인 쿼리

```
SELECT    department_name
FROM      department
WHERE     department_id = (SELECT department_id
                           FROM employee
                           WHERE employee_name='홍길동')
```

## ▪ 다중 행 서브쿼리

- 서브 쿼리의 조회결과 행이 0개 이상의 경우의 쿼리
- IN, ALL, ANY 연산자등과 사용된다.

```
DELETE FROM employee
WHERE salary IN (SELECT salary
                 FROM employee
                 WHERE department_id=30)
```

# 서브쿼리(Subquery)

- **인 라인 뷰(Inline view)**

- FROM절 상에 오는 서브쿼리
- SELECT결과를 논리적 테이블로 처리

```
SELECT  e.employee_name,  
        d.department_name  
FROM    employee e,  
        (select department_id from department where location='서울') d  
WHERE   e.department_id = d.department_id
```