

# MySQL

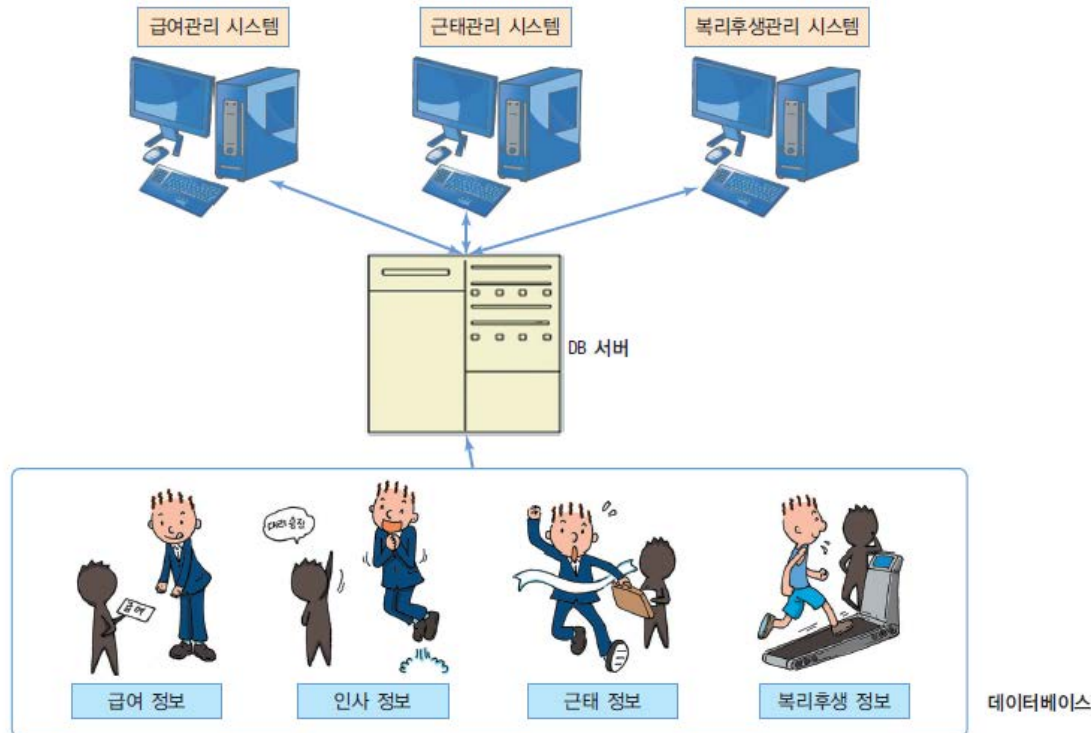
## 1. Fundamental

1. 데이터베이스 & 데이터 베이스 관리 시스템
2. 관계형 데이터베이스(RDB)
3. SQL(Structured Query Language) 개요

# 1. 데이터베이스 & 데이터베이스 관리시스템

## ❑ 데이터베이스의 기본개념 (정의)

- 데이터의 집합 ( a Set of Data )
- 여러 응용 시스템(프로그램)들의 통합된 정보들을 저장하여 운영할 수 있는 공유(share) 데이터의 집합
- 효율적으로 저장, 검색, 갱신할 수 있도록 데이터 집합들끼리 연관시키고 조직화되어야 한다.



# 1. 데이터베이스 & 데이터베이스 관리시스템

## □ 데이터베이스의 특성

- 실시간 접근성(**Real-time Accessibility**)

사용자의 요구를 즉시 처리할 수 있다.

- 지속적인 변화(**Continuous Evolution**)

정확한 값을 유지하려고 삽입·삭제·수정 작업 등을 이용해 데이터를 지속적으로 갱신할 수 있다.

- 동시 공유성(**Concurrent Sharing**)

사용자마다 서로 다른 목적으로 사용하므로 동시에 여러 사람이 동일한 데이터에 접근하고 이용할 수 있다.

- 내용 참조(**Content Reference**)

저장한 데이터 레코드의 위치나 주소가 아닌 사용자가 요구하는 데이터의 내용, 즉 데이터 값에 따라 참조할 수 있어야 한다.

# 1. 데이터베이스 & 데이터베이스 관리시스템

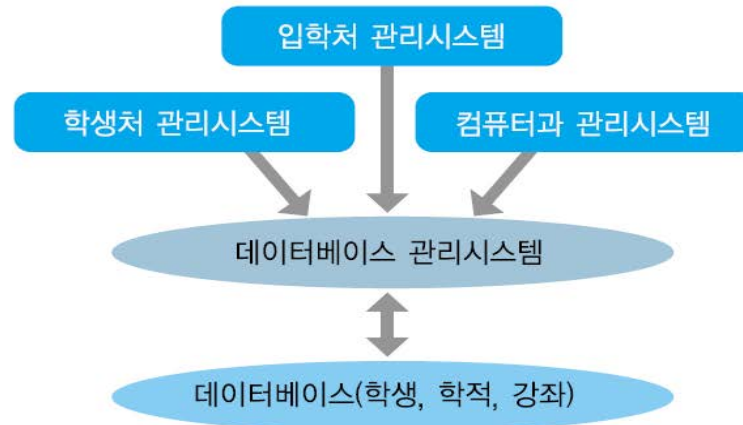
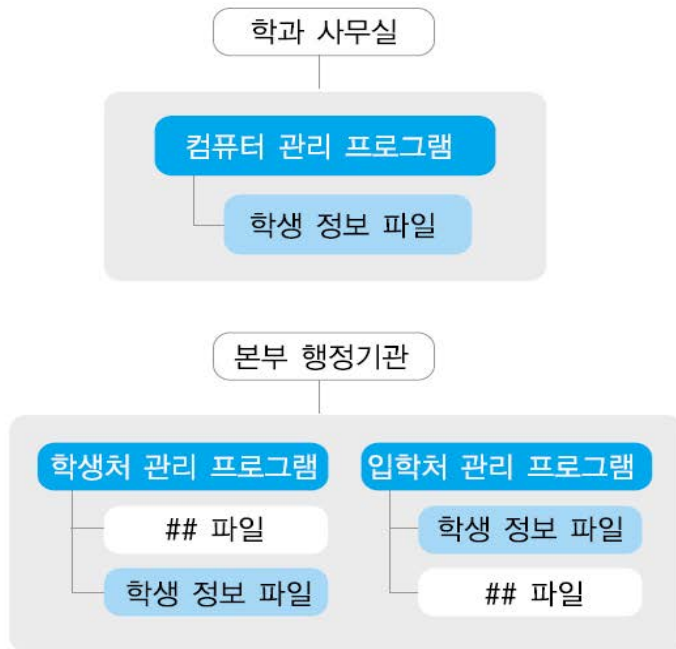
## □ 데이터 베이스 관리 시스템 ( Database Management System = DBMS )

- 데이터베이스를 관리하는 소프트웨어
- 여러 응용 소프트웨어(프로그램) 또는 시스템이 동시에 데이터베이스에 접근하여 사용할 수 있게 한다
- 필수 3기능
  - 정의기능 : 데이터 베이스의 논리적, 물리적 구조를 정의
  - 조작기능 : 데이터를 검색, 삭제, 갱신, 삽입, 삭제하는 기능
  - 제어기능 : 데이터베이스의 내용 정확성과 안전성을 유지하도록 제어하는 기능
- Oracle, SQL Server, MySQL, DB2 등의 상용 또는 공개 DBMS가 있다.

# 1. 데이터베이스 & 데이터베이스 관리시스템

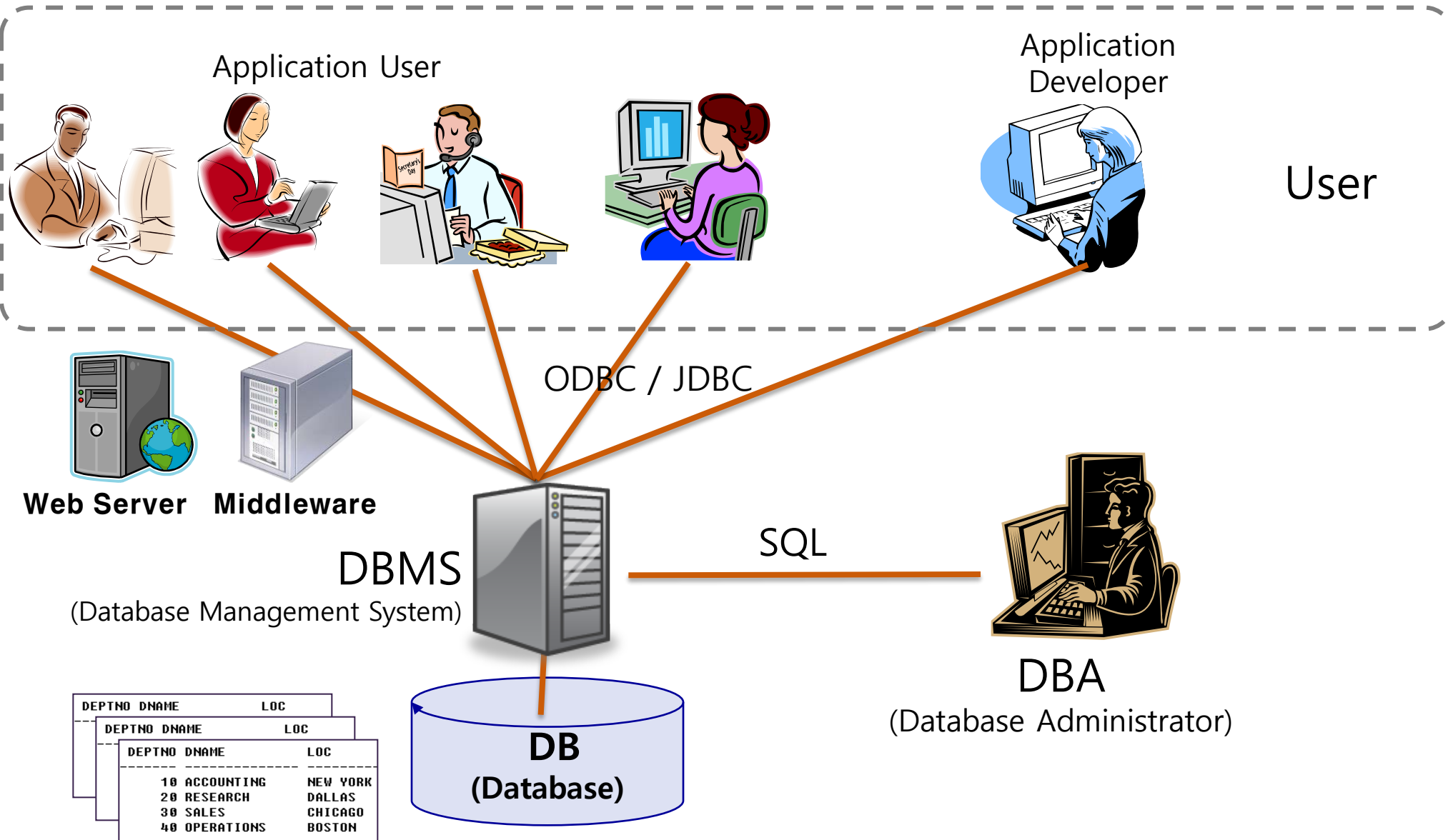
## □ 파일 시스템과의 비교

- 데이터의 종속성 보완
- 중복성 제거



파일관리시스템(좌)과 데이터베이스 관리시스템(우)

# 1. 데이터베이스 & 데이터베이스 관리시스템



# 1. 데이터베이스 & 데이터베이스 관리시스템

---

## ❑ 데이터 베이스 관리 시스템 장점

- 데이터 중복이 최소화
- 데이터의 일관성 및 무결성 유지
- 데이터 보안 보장

## ❑ 데이터 베이스 관리 시스템 단점

- 운영비가 비싸다
- 백업 및 복구에 대한 관리가 복잡
- 부분적 데이터베이스 손실이 전체 시스템을 정지

# 1. 데이터베이스 & 데이터베이스 관리시스템

## □ 데이터 베이스의 종류

### – 관계형 데이터베이스 ( Relational Database = RDB )

1970년 **IBM E. F. Codd** 에 의해 제안되어 수 십년동안 주류 데이터베이스로 성장 확대

키와 값들의 간단한 관계를 테이블화 시킨 매우 간단한 원칙의 개념의 데이터베이스

일련의 정형화된 테이블로 구성된 데이터 항목들의 집합이며 각 테이블은 데이터의 성격에 따라 여러 개의 컬럼(키)이 포함된다.

사용자는 **SQL**이라는 표준 질의어를 통해 데이터를 조작 또는 조회 할 수 있다.

### – 객체 지향 데이터베이스 ( Object Oriented DataBase = OODB )

정보를 객체의 형태로 표현하는 데이터베이스

객체 모델이 그대로 데이터베이스에도 적용되어 데이터 모델을 그대로 응용프로그램 에 적용,  
데이터 변환과 질의 작업이 필요치 않은 장점



# 1. 데이터베이스 & 데이터베이스 관리시스템

## □ 데이터 베이스의 종류

- 객체 관계형 데이터베이스 ( **Object Relation DataBase = ORDB** )

관계형 데이터베이스에서 사용하는 데이터를 확장

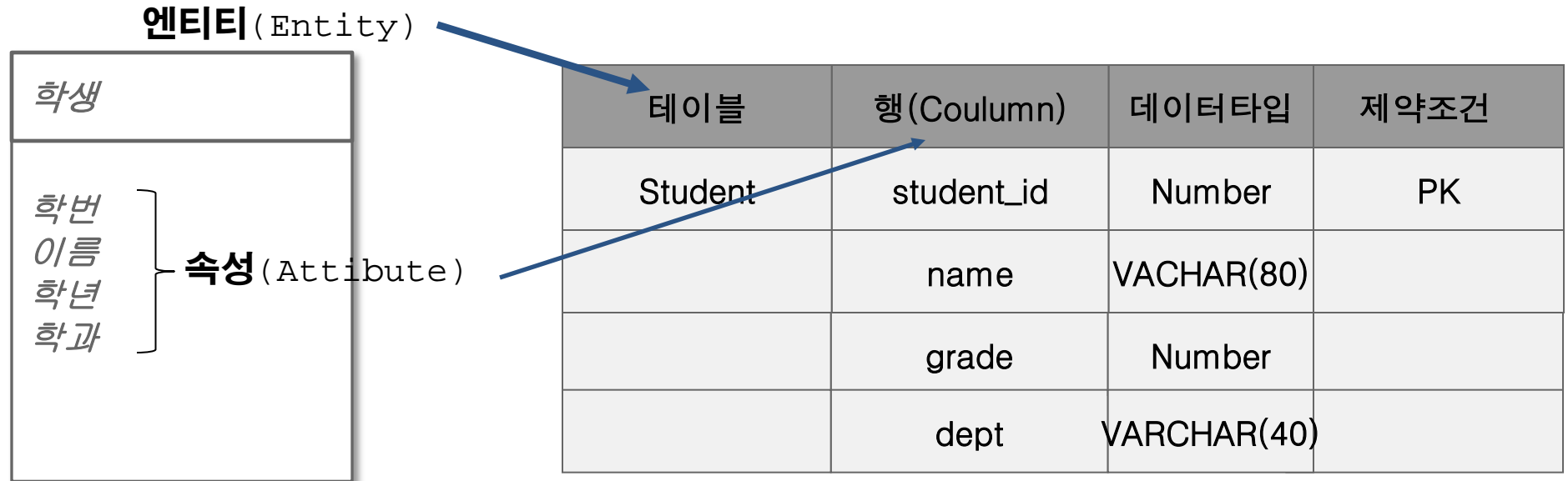
관계형 데이터베이스를 객체 지향 모델링과 데이터를 관리하는 기능을 갖도록 확장한 것

- **NoSQL**

대용량 데이터의 웹 서비스와 SNS, 클라우드 컴퓨팅의 확대 보급과 대중화로 최근 주목 받는 데이터베이스 기술

## 2. 관계형 데이터 베이스

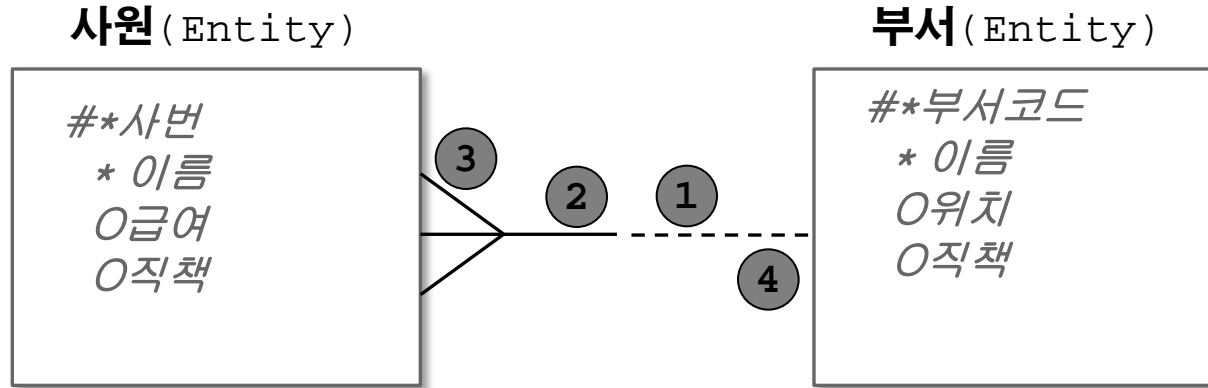
### □ 논리적(개념적) 데이터 모델링 & 물리적인 데이터베이스



### □ 관계(relation) 는 ?

## 2. 관계형 데이터 베이스

### □ ERD 예시



1. 어떤부서는 사원을 배치 받지 않을 수 있다. (점선)
2. 사원은 특정부서에 소속되어 있다. (실선)
3. 한 부서에는 여러명의 사원이 소속되어 있다. (다중선)
4. 한 사원은 하나의 부서에만 속한다.

# : 대표값 즉, PK (Primary Key)

\* : NOT NULL

O : NULL 가능

## 2. 관계형 데이터 베이스

### □ 테이블(table)

table Student		column(key)	
student_id	name	grade	dept
1	정성진	1	컴퓨터
2	박현진	2	수학
3	홍길동	4	물리
...	...	...	...

row( record )

field

field

**테이블** : RDBMS의 기본적 저장구조 한 개 이상의 column 과 0개 이상의 row로 구성

**열(Column)**: 테이블 상에서의 단일 종류의 데이터를 나타냄 특정 데이터 타입 및 크기를 가지고 있음

**행(Row)**: Column들의 값의 조합. 레코드라고 불린다.

기본키(PK)에 의해 구분된다. 기본키는 중복을 허용하지 않으며 없어서는 안 된다.

**Field** : Row 와 Column의 교차점으로 Field는 데이터를 포함할 수 있고 없을 때는 NULL 값을 가지고 있다고 한다.

### 3. SQL(Structured Query Language) 개요

- ❑ 데이터베이스 스키마 생성, 자료의 검색, 수정, 그리고 데이터베이스 객체 접근 관리 등을 위해 고안된 언어
- ❑ 다수의 데이터베이스 관련 프로그램의 표준언어
- ❑ **SQL 명령어의 종류**
  1. **DML ( Data Manipulation Language )** : 데이터 조작어로 검색 및 수정하기 위한 수단제공
    - **SELECT, INSERT, UPDATE, DELETE, MERGE**
  2. **DDL ( Data Definition Language )** : 데이터 구조를 생성, 변경, 삭제등의 기능을 제공
    - **CREATE, ALTER, DROP, RENAME**
  3. **DCL ( Data Control Language )** – 데이터에 대한 권한 관리 및 트랜잭션 제어
    - **GRANT, REVOKE**

# MySQL

## 2. MySQL 기본

1. About MySQL
2. MySQL 설치
3. Basic Queries
4. 계정/권한 관리

# 1. About MySQL

- ☐ 세계에서 가장 인기 있는 Open source DB
- ☐ 1996년 첫 공식 버전 발표
- ☐ 2001년 GNU GPL 등록
- ☐ 2008년 1월 SUN에서 인수
- ☐ 하루 50,000번 이상 다운로드
- ☐ Open source LAMP/SAMP stack으로 급성장
- ☐ 100여개 SW 및 HW 회사에 번들로 설치
- ☐ DB관리 TCO의 획기적인 감소



# 1. About MySQL

## □ MYSQL 구성

### MySQL Server

- Community Server
- Enterprise Server
- Embedded Server



### MySQL GUI Tools

- Query Browser
- Administrator
- Migration Toolkit
- Visual Studio Plug-in
- MySQL Workbench



### MySQL Drivers

- JDBC
- ODBC
- .NET
- PHP





# 1. 데이터베이스 & 데이터베이스 관리시스템

## □ 주요기능

- > 최상의 신뢰성과 보안성을 제공하는 오픈 소스 데이터베이스
- > Stored Procedure, Trigger, View 등 RDBMS로서 기본 기능에 충실
- > 사용자의 편의에 따른 Pluggable Storage Engine 기능
- > 다양한 Third पार्ट 엔진 지원
- > 마법사 툴을 이용한 손쉬운 설치 및 환경설정
- > 다양한 관리자용 GUI 툴 제공(Administration, Migration, Backup, Workbench, Query Browser 등)
- > 중앙 집중 관리(보안, 스키마 관리, Replication, 성능 모니터링 등)
- > 다양한 Platform 지원
- > 가격대비 최대 성능 효과의 TCO 절감 DBMS

## 2. MySQL 설치

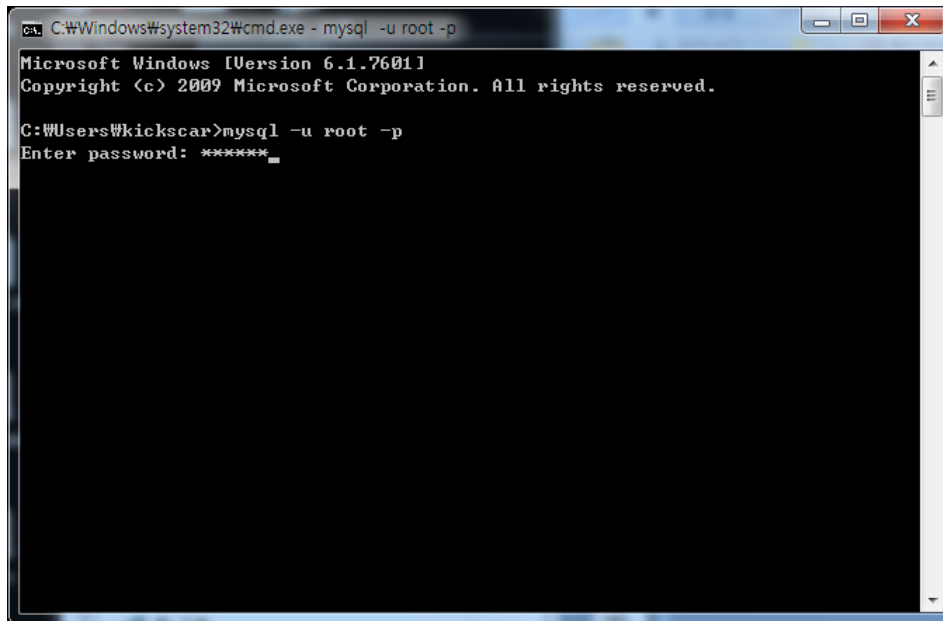
### ❑ MySQL 접속

`mysql -u (사용자 이름) -D (데이터베이스 이름) -p (패스워드)`

---

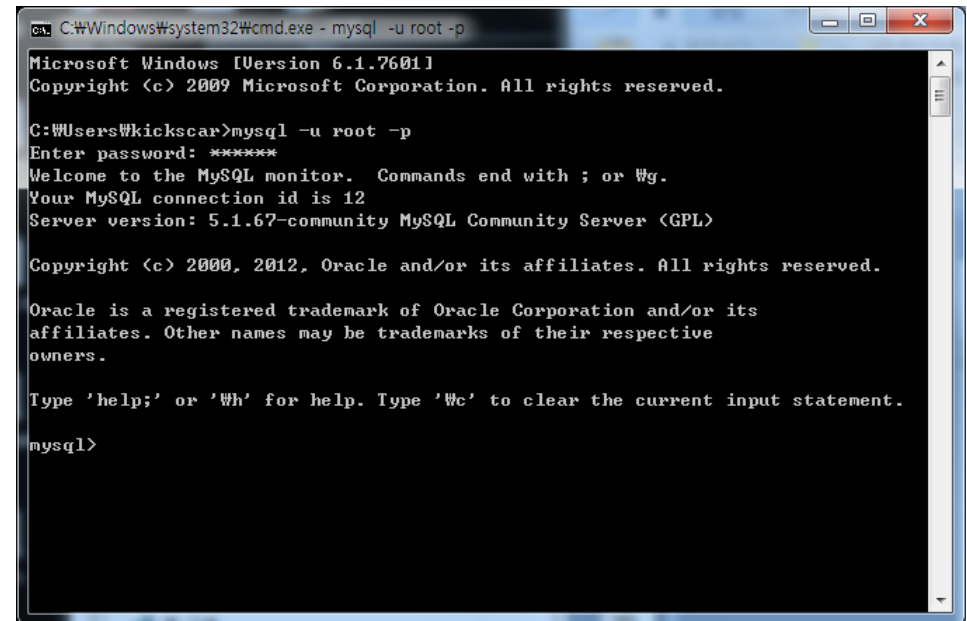
cmd창에서

`mysql -u root -p`      <- 패스워드를 비워놓고 엔터를 치면 패스워드 입력을 대기 한다.



```
C:\Windows\system32\cmd.exe - mysql -u root -p
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Wkickscar>mysql -u root -p
Enter password: *****
```



```
C:\Windows\system32\cmd.exe - mysql -u root -p
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Wkickscar>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 5.1.67-community MySQL Community Server (GPL)

Copyright (c) 2000, 2012, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

### 3. Basic Queries

---

#### □ 연결

```
Enter password: *****
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 241 to server version: 3.23.49
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql>
```

#### □ MySQL Shell 에서 빠져나오기( 연결 끊기)

```
mysql> QUIT
```

```
mysql> exit
```

### 3. Basic Queries

❑ 로그인 후, 간단한 쿼리 실행

❑ 실습:

```
mysql> SELECT VERSION(), CURRENT_DATE;
```

```
+-----+-----+
| VERSION() | CURRENT_DATE |
+-----+-----+
| 5.1.67    | 2013-01-05   |
+-----+-----+
```

```
1 row in set (0.00 sec)
```

❑ 대부분 **MySQL commands semicolon (;)**으로 끝난다.

❑ **MySQL** 은 찾은 전체 **row**를 출력하고 마지막에 전체 **row** 수와 쿼리실행에 걸린 시간을 표시한다.

### 3. Basic Queries

---

❑ 키워드는 대소문자 구별이 없다.

❑ 다음 쿼리들은 모두 같다:

```
mysql> SELECT VERSION( ), CURRENT_DATE;
```

```
mysql> select version( ), current_date;
```

```
mysql> SeLeCt vErSiOn( ), current_DATE;
```

## Database 사용

- ❑ 작업하기 위한 데이터베이스를 선택하기 위해서는 어떤 데이터베이스가 존재하는 지 알아보아야 한다.
- ❑ 현재 서버에 존재하는 데이터베이스를 찾아보기 위해서 **SHOW statement**을 사용한다.

```
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| mysql    |  
| test     |  
+-----+  
2 rows in set (0.01 sec)
```

## Database 사용

- 새 **database**를 생성하기 위해 “**create database**” command 사용:

```
mysql> create database webdb;
```

- **Database**을 선택하기 위해, “**use**” command 사용:

```
mysql> use webdb;
```

## Table 생성

❑ **Database**를 선택 후, **Database**의 전체 테이블 목록을 출력:

```
mysql> show tables;
```

```
Empty set (0.02 sec)
```

❑ “**empty set**” 은 데이터베이스에 어떤 테이블도 아직 생성되지 않았다는 것을 알려주는 것이다.




## Table 생성

❑ 애완동물 정보를 저장하기 위한 테이블 생성

❑ **Table: pets**

- name: VARCHAR(20)
- owner: VARCHAR(20)
- species: VARCHAR(20)
- gender: CHAR(1)
- birth: DATE
- death: DATE



VARCHAR 는 보통  
문자열을 저장하기  
위해 사용하는 데이터  
타입이다.

## Table 생성

❑ Table 생성을 위해, **CREATE TABLE** command 사용:

```
mysql> CREATE TABLE pet (  
    -> name VARCHAR(20),  
    -> owner VARCHAR(20),  
    -> species VARCHAR(20),  
    -> gender CHAR(1),  
    -> birth DATE,  
    -> death DATE );
```

Query OK, 0 rows affected (0.04 sec)

## Describing Table

❑ **table** 구조를 확인하기 위해, **DESCRIBE** command 사용:

```
mysql> describe pet;
```

Field	Type	Null	Key	Default	Extra
name	varchar(20)	YES		NULL	
owner	varchar(20)	YES		NULL	
species	varchar(20)	YES		NULL	
gender	char(1)	YES		NULL	
birth	date	YES		NULL	
death	date	YES		NULL	

```
6 rows in set (0.02 sec)
```

## Table 삭제

- ❑ **table** 전체를 삭제하기 위해 **DROP TABLE** command 사용:

```
mysql> drop table pet;
```

```
Query OK, 0 rows affected (0.02 sec)
```

## Loading Data

❑ **INSERT statement**를 사용해서 **table**에 데이터를 입력한다.

❑ 예제:

```
INSERT INTO pet VALUES  
    ( 'Fluffy' , 'Harold' , 'cat' , 'f' ,  
      '1999-02-04' , NULL ) ;
```

❑ 많은 데이터를 한 번에 입력하는 방법은?

### 3. Basic Queries

## 더 많은 애완동물 Data

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1998-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

## Loading Sample Pet Data

- ❑ 한 줄당 한 레코드 정보를 담고 있는 **'pet.txt'** 라는 텍스트 파일을 생성한다.
- ❑ 한 레코드의 값들은 탭(**tab**)으로 구분되어야 한다.  
그리고 순서는 테이블을 생성할 때의 **column**순서대로 되어 있어야 한다.
- ❑ 그리고 **data**를 테이블에 **load**하기 위해 **LOAD DATA Command**를 사용한다.

### 3. Basic Queries

## Sample Data 파일

Fluffy	Harold	cat	f	1993-02-04	\N	
Claws	Gwen	cat	m	1994-03-17	\N	
Buffy	Harold	dog	f	1989-05-13	\N	
Fang	Benny	dog	m	1990-08-27	\N	
Bowser	Diane	dog	m	1979-08-31	1995-07-29	
Chirpy	Gwen	bird	f	1998-09-11	\N	
Whistler		Gwen	bird	\N	1997-12-09	\N
Slim	Benny	snake	m	1996-04-29	\N	

**pet.txt** 내용 테이블에 로드하기:

```
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```



## SQL Select

❑ **Table**에서 **Data**를 가져오기 위해서 **SELECT** 구문을 사용한다.

❑ **Format:**

**SELECT** what\_to\_select

**FROM** which\_table

**WHERE** conditions\_to\_satisfy

## 전체 Data Select

- ❑ **SELECT**를 가장 간단히 사용하게 되면 **table**의 모든 데이터를 가져오게 된다.

```
mysql> select * from pet;
```

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1999-02-04	NULL
Claws	Gwen	cat	f	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL
Fang	Benny	dog	m	1999-08-27	NULL
Bowser	Diane	dog	m	1998-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	NULL
Whistler	Gwen	bird		1997-12-09	NULL
Slim	Benny	snake	m	1996-04-29	NULL

```
8 rows in set (0.00 sec)
```

## 특정 Row에 대한 Select

- 테이블에서 특정 **row**만 가져올 수 있다.
- 예를 들어, 바우저의 생일이 변경되었는지 확인하기 위해 다음과 같이 바우저의 레코드를 선택 할 수 있다.:

```
mysql> SELECT * FROM pet WHERE name = "Bowser";
```

name	owner	species	sex	birth	death
Bowser	Diane	dog	m	1998-08-31	1995-07-29

```
1 row in set (0.00 sec)
```

## 특정 Row에 대한 Select

- 1998년 이 후에 태어난 동물을 조회할 때:

```
SELECT * FROM pet WHERE birth >= "1998-1-1";
```

- 암컷 강아지들을 조회 할 때, 논리 연산자 **AND**를 함께 사용해서:

```
SELECT * FROM pet WHERE species = "dog" AND sex = "f";
```

- 뱀과 새를 모두 조회할 때는 논리 연산자 **OR**와 함께:

```
SELECT * FROM pet WHERE species = "snake" OR species = "bird";
```

## 특정 Row에 대한 Select

- ❑ **Row**의 전체 **column**을 보고 싶지 않을 경우, **comma(,)**로 분리해서 관심있는 **column**를 적어주면 된다.
- ❑ 예를 들어, 애완동물의 생년만 알고 싶다면, **name**과 **borth**만 **select**하면 된다.
- ❑ 다음 페이지 예제 참고

### 3. Basic Queries

## 특정 Row에 대한 Select

```
mysql> select name, birth from pet;
```

name	birth
Fluffy	1999-02-04
Claws	1994-03-17
Buffy	1989-05-13
Fang	1999-08-27
Bowser	1998-08-31
Chirpy	1998-09-11
Whistler	1997-12-09
Slim	1996-04-29

```
8 rows in set (0.01 sec)
```

### 3. Basic Queries

## Data 정렬

- ❑ 결과를 정렬하고 싶을 때는, **ORDER BY** 절을 사용한다.
- ❑ 예를 들어, 동물의 생일이 날짜 순으로 정렬된 결과를 원하면:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
```

```
+-----+-----+
| name      | birth      |
+-----+-----+
| Buffy      | 1989-05-13 |
| Claws      | 1994-03-17 |
| Slim       | 1996-04-29 |
| Whistler   | 1997-12-09 |
| Bowser     | 1998-08-31 |
| Chirpy     | 1998-09-11 |
| Fluffy     | 1999-02-04 |
| Fang       | 1999-08-27 |
+-----+-----+
8 rows in set (0.02 sec)
```

## Data 정렬

- ❑ 역순 정렬을 해야 할 때에는 **DESC (descending keyword)**를 붙여주면 된다.

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
```

```
+-----+-----+
| name      | birth      |
+-----+-----+
| Fang      | 1999-08-27 |
| Fluffy    | 1999-02-04 |
| Chirpy    | 1998-09-11 |
| Bowser    | 1998-08-31 |
| Whistler   | 1997-12-09 |
| Slim      | 1996-04-29 |
| Claws     | 1994-03-17 |
| Buffy     | 1989-05-13 |
+-----+-----+
8 rows in set (0.02 sec)
```



## NULL 다루기

- **NULL** 이 의미하는 것은 빈 값 또는 알 수없는 값이다.
- **NULL**인지 아닌지 확인 하기 위해, **=**, **<** 또는 **<>** 와 같은 산술 비교 연산자를 사용할 수 없다.
- 대신에, **IS NULL** 그리고 **IS NOT NULL** 연산자를 사용해야 한다.

### 3. Basic Queries

## NULL 다루기

□ 예제 - 죽은 애완동물 조회

```
mysql> select name  
      >    from pet  
      >  where death IS NOT NULL;
```

```
+-----+
```

```
| name |
```

```
+-----+
```

```
| Bowser |
```

```
+-----+
```

```
1 row in set (0.01 sec)
```

## 패턴 매칭

### □ MySQL 기본적으로 제공하는 것:

- 표준 SQL pattern matching
- 정규표현식 pattern matching,  
vi, grep, sed와 같은 Unix에서의 유틸리티에서의 그 것과 같다.

### □ SQL Pattern matching:

- LIKE or NOT LIKE 비교 연산자를 사용해서 패턴매칭을 한다.
- 기본적으로 영문자인 경우 대소문자 구별을 안한다.

### □ 특수문자:

- \_ 는 한문자에 대응한다.
- % 여러문자열과 대응하게 된다.

## 패턴 매칭 예제

□ 'b'로 시작하는 이름의 동물 조회:

```
mysql> SELECT * FROM pet WHERE name LIKE "b%";
```

name	owner	species	sex	birth	death
Buffy	Harold	dog	f	1989-05-13	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29

## 패턴 매칭 예제

□ `fy`로 끝나는 동물 조회:

```
mysql> SELECT * FROM pet WHERE name LIKE "%fy";
```

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

## 패턴 매칭 예제

□ 'w'가 포함된 이름의 동물 조회:

```
mysql> SELECT * FROM pet WHERE name LIKE "%w%";
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29
Whistler	Gwen	bird	NULL	1997-12-09	NULL

## 패턴 매칭 예제

- pattern character `_` 를 사용해서 정확히 5문자 이름의 동물 조회:

```
mysql> SELECT * FROM pet WHERE name LIKE "_____";
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

## Row 카운팅(counting)

- ❑ 테이블에 어떤 특정 조건의 데이터가 어떤 빈도로 나타나는가에 대한 **Databases**의 응답이다.
- ❑ 예를 들어, 각 주인들이 몇 마리의 애완동물을 가지고 있는가를 알고 싶을 때
- ❑ 애완동물의 총 수는 테이블의 전체 **row**수이다. 왜냐하면 애완동물 한 마리당 하나의 레코드를 가지기 때문이다.
- ❑ **COUNT()** function counts **NULL**이 아닌 결과의 수이다.



## Row 카운팅(counting) 예제

□ 전체 애완동물 수를 산정하는 쿼리:

```
mysql> SELECT COUNT(*) FROM pet;
```

```
+-----+
```

```
| COUNT(*) |
```

```
+-----+
```

```
|          9 |
```

```
+-----+
```

## 4. 계정 / 권한 관리

### 1. 새로운 사용자 추가 예제

```
mysql>          grant all privileges
>              on webdb.*
>              to dev@localhost
> identified by 'dev';
Query OK, 0 rows affected (0.04 sec)
```

### 2. 관리자 추가 예제

```
mysql>          grant all privileges
>              on *.*
>              to admin@'%'
> identified by 'admin';
Query OK, 0 rows affected (0.04 sec)
```

### 3. mysql Database의 user, db, host 각 테이블에 insert 구문 실행

\*\* flush privileges 필요

## 4. 계정 / 권한 관리

---

### 사용자 삭제

```
mysql> drop user dev;  
Query OK, 0 rows affected (0.04 sec)
```

# MySQL

## 3. 데이터 검색 I

1. SELECT 기본구문
2. 데이터 정렬
3. 특정 행 검색
4. MySQL 함수

## 1. SELECT 기본 구문

### 1.1 SELECT 구문의 기본문형

SELECT(DISTINCT) 컬럼명(ALIAS)  
FROM 테이블명;

<b>SELECT</b>	검색하고자 하는 데이터(컬럼)를 나열 한다
<b>DISTINCT</b>	중복행을 제거
<b>ALIAS</b>	나타날 컬럼에 대한 다른 이름 부여
<b>FROM</b>	선택한 컬럼이 있는 테이블을 명시한다.

## 1. SELECT 기본 구문

### 1.2 SELECT 구문 예제

□ 전체 데이터 검색

□ `SELECT` 뒤에 `*`를 기술함으로써 나타낼 수 있다

□ 예제 : **deaprtments** 테이블의 모든 데이터를 출력.

```
SELECT *  
FROM departments;
```

## 1. SELECT 기본 구문

---

### 1.2 SELECT 구문 예제

□ 특정 칼럼 검색

□ SELECT 뒤에 컬럼을 콤마(,)로 구별해서 나열

□ 예제 : **employees** 테이블에서 직원의 이름, 성별, 입사일을 출력

```
SELECT first_name, gender, hire_date  
FROM employees;
```

## 1. SELECT 기본 구문

### 1.2 SELECT 구문 예제

□ 컬럼에 대한 **ALIAS** 부여

□ 컬럼에 대한 **ALIAS**(별칭)을 부여해서 나타내는 컬럼의 **HEADING**을 변경할 수 있다.

□ 예제 : **employees** 테이블에서 직원의 이름, 성별, 입사일을 출력

```
SELECT first_name AS 이름,  
       gender AS 성별,  
       hire_date AS 입사일  
FROM employees;
```



## 1. SELECT 기본 구문

### 1.2 SELECT 구문 예제

#### □ 컬럼의 합성 (Concatenation)

#### □ 문자열 결합함수 `concat` 사용

#### □ 예제 : **employees** 테이블에서 직원의 전체이름, 성별, 입사일을 출력

```
SELECT concat( first_name, ' ', last_name) AS 이름,  
        gender AS 성별,  
        hire_date AS 입사일  
FROM employees;
```

## 1. SELECT 기본 구문

### 1.2 SELECT 구문 예제

#### □ 중복행의 제거 (**DISTINCT**)

□ 중복되는 행이 출력되는 경우, **DISTINCT** 키워드로 중복행을 제거

□ 예제 1: **titles** 테이블에서 모든 직급의 이름 출력

```
SELECT title FROM titles;
```

□ 예제 2: **titles** 테이블에서 직급은 어떤 것들이 있는지 직급이름을 한 번씩만 출력

```
SELECT DISTINCT title FROM titles;
```

## 2. 데이터의 정렬

### 2.1 ORDER BY 절

```
SELECT(DISTINCT) 컬럼명(ALIAS)  
FROM 테이블명  
ORDER BY 컬럼이나 표현식 (ASC 또는 DESC);
```

<b>ASC</b>	오름차순 정렬, 기본값
<b>DESC</b>	내림차순

## 2. 데이터의 정렬

### 2.1 ORDER BY 절

- 예제 : **employees** 테이블에서 직원의 전체이름, 성별, 입사일을 입사일 순으로 출력

```
SELECT concat( first_name, ' ', last_name) AS 이름,  
        gender AS 성별,  
        hire_date AS 입사일  
FROM employees  
ORDER BY hire_date;
```

- 예제 : **salaries** 테이블에서 2001년 월급을 가장 높은순으로 사번,  
 월급순으로 출력

```
SELECT emp_no, salary  
FROM salaries  
WHERE from_date like '2001-%'  
ORDER BY salary DESC
```

### 3. 특정 행 검색

## 3.1 WHERE 절

SELECT(DISTINCT) 컬럼명(ALIAS)  
FROM 테이블명  
WHERE 조건식  
ORDER BY 컬럼이나 표현식 (ASC 또는 DESC)

조건식	컬럼이름이나 표현식의 상수, 연산자로 구성
-----	-------------------------

WHERE 형식   연산자   값

ex)

WHERE title = 'Staff'

WHERE salary BETWEEN 1000 AND 2000

### 3. 특정 행 검색

## 3.1 WHERE 절

□ 산술비교 연산자

□ 예제 : `employees` 테이블에서 1991년 이전에 입사한 직원의 이름,  
성별, 입사일을 출력

```
SELECT concat( first_name, ' ', last_name ) AS 이름,  
       gender AS 성별,  
       hire_date AS 입사일  
FROM employees  
WHERE hire_date < '1991-01-01'
```

### 3. 특정 행 검색

## 3.1 WHERE 절

□ 논리연산자

□ 예제 : `employees` 테이블에서 1989년 이전에 입사한 여직원의 이름,  
입사일을 출력

```
SELECT concat( first_name, ' ', last_name ) AS 이름,  
       hire_date AS 입사일  
FROM employees  
WHERE gender='f'  
       AND hire_date < '1989-01-01'
```

### 3. 특정 행 검색

## 3.2 SQL 비교 연산자

#### □ IN

- 예제 : dept\_emp 테이블에서 부서 번호가 d005나 d009에 속한 사원의 사번, 부서번호 출력

```
SELECT emp_no, dept_no  
FROM dept_emp  
WHERE dept_no in( 'd005', 'd009' )
```



### 3. 특정 행 검색

## 3.2 SQL 비교 연산자

### □ LIKE

- 와일드 카드를 사용하여 특정 문자를 포함한 값에 대한 조건을 처리
- % 는 0에서부터 여러 개의 문자열을 나타냄
- \_ 는 단 하나의 문자를 나타내는 와일드 카드

- 예제 : employees 테이블에서 1989년에 입사한 직원의 이름,  
입사일을 출력

```
SELECT concat( first_name, ' ', last_name ) AS 이름,  
       hire_date AS 입사일  
FROM employees  
WHERE hire_date LIKE '1989%'
```

- 과제 : 예제를 산술비교 연산자를 사용한 SQL문으로 변경해 보세요

## 4. MySQL 함수

### 4.1 문자형 함수

#### □ UCASE, UPPER

#### □ 예제

```
mysql> SELECT UPPER('SEoul'), UCASE('seOUL');
```

+-----+-----+	
UPPER('SEoul')	UCASE('seOUL')
+-----+-----+	
SEOUL	SEOUL
+-----+-----+	

## 4. MySQL 함수

### 4.1 문자형 함수

#### ❑ UCASE, UPPER

- ❑ 예제 : employees 테이블에서 last\_name이 acton인 사원의 이름, 성별, 입사일 출력

```
SELECT concat(first_name, ' ',last_name), gender, hire_date  
FROM employees  
WHERE last_name = 'ACTON'
```

- ❑ 결과는? UCASE나 UPPER 함수를 사용해서 결과가 나오도록 수정하세요.

## 4. MySQL 함수

### 4.1 문자형 함수

#### □ LCASE, LOWER

#### □ 예제

```
mysql> SELECT LOWER('SEoul'), LCASE('seOUL');
```

+	-----	+	-----	+
	LOWER('SEoul')		LCASE('seOUL')	
+	-----	+	-----	+
	seoul		seoul	
+	-----	+	-----	+

## 4. MySQL 함수

### 4.1 문자형 함수

#### □ substring

#### □ 예제

```
mysql> SELECT SUBSTRING('Happy Day',3,2);
```

```
+-----+-----+
| SUBSTRING('Happy Day',3,2) |
+-----+-----+
| pp                        |
+-----+-----+
```

## 4. MySQL 함수

### 4.1 문자형 함수

#### □ substring

- 예제: `employees` 테이블에서 1989년에 입사한 직원의 이름,  
입사일을 출력

```
SELECT concat( first_name, ' ', last_name ) AS 이름,  
       hire_date AS 입사일  
FROM employees  
WHERE substring( hire_date, 1, 4);
```

## 4. MySQL 함수

### 4.1 문자형 함수

#### □ LPAD, RPAD

#### □ 예제

```
mysql> SELECT LPAD('hi',5,'?'), LPAD('joe',7,'*');
```

```
+-----+-----+
| LPAD('hi',5,'?') | LPAD('joe',7,'*') |
+-----+-----+
| ???hi           |          ****joe  |
+-----+-----+
```

```
mysql> SELECT RPAD('hi',5,'?'), RPAD('joe',7,'*');
```

?

## 4. MySQL 함수

### 4.1 문자형 함수

#### □ LPAD, RPAD

- 예제 : `salaries` 테이블에서 2001년 급여가 70000불 이하의 직원만 사번, 급여로 출력하되 급여는 10자리로 부족한 자리수는 \*로 표시

```
SELECT emp_no, LPAD( cast(salary as char), 10, '*' )  
FROM salaries  
WHERE from_date like '2001-%'  
AND salary < 70000
```



## 4. MySQL 함수

### 4.1 문자형 함수

#### □ TRIM, LTRIM, RTRIM

#### □ 예제

```
mysql> SELECT LTRIM(' hello '), RTRIM(' hello ');
```

```
+-----+
| LTRIM(' hello ') | RTRIM(' hello ') |
+-----+
| 'hello '         | ' hello'         |
+-----+
```

```
mysql> SELECT TRIM(' hi '), TRIM(BOTH 'x' FROM 'xxxhixxx');
```

```
+-----+-----+
| TRIM(' hi ') | TRIM(BOTH 'x' FROM 'xxxhixxx') |
+-----+-----+
| hi           | hi                               |
+-----+-----+
```

**BOTH** 대신에 **LEADING**, **TRAILING**으로 바꾸어 각각 테스트 해보세요.

## 4. MySQL 함수

### 4.1 문자형 함수

#### □ TRIM, LTRIM, RTRIM

□ 예제 : **salaries** 테이블에 대한 LPAD 예제의 결과를 \*생략하여 표시

```
SELECT emp_no,  
       TRIM( LEADING '*' FROM LPAD( cast(salary as char), 10, '*' ) )  
FROM salaries  
WHERE from_date like '2001-%'  
       AND salary < 70000
```

## 4. MySQL 함수

### 4.2 숫자형 함수

❑ **ABS(x) : x의 절대값을 구한다.**

❑ 예제

```
mysql> SELECT ABS(2), ABS(-2);
```

+-----+-----+	
ABS(2)	ABS(-2)
+-----+-----+	
2	2
+-----+-----+	

## 4. MySQL 함수

### 4.2 숫자형 함수

❑ `MOD(n,m) %` : `n`을 `m`으로 나눈 나머지 값을 출력한다.

❑ 예제

```
mysql> SELECT MOD(234,10), 253 % 7, MOD(29,9);
```

+-----+-----+-----+			
MOD(234,10)	253 % 7	MOD(29,9)	
+-----+-----+-----+			
	4	1	2
+-----+-----+-----+			

## 4. MySQL 함수

### 4.2 숫자형 함수

❑ **FLOOR(x) : x보다 크지 않은 가장 큰 정수를 반환한다. BIGINT로 자동 변환됨**

❑ 예제

```
mysql> SELECT FLOOR(1.23), FLOOR(-1.23);
```

+-----+-----+	
FLOOR(1.23)	FLOOR(-1.23)
+-----+-----+	
1	-2
+-----+-----+	

## 4. MySQL 함수

### 4.2 숫자형 함수

❑ **CEILING(x) : x보다 작지 않은 가장 작은 정수를 반환한다.**

❑ 예제

```
mysql> SELECT CEILING(1.23), CEILING(-1.23);
```

+-----+-----+	
CEILING(1.23)	CEILING(-1.23)
+-----+-----+	
2	-1
+-----+-----+	

## 4. MySQL 함수

### 4.2 숫자형 함수

□ **ROUND(x) : x에 가장 근접한 정수를 반환한다.**

□ 예제

```
mysql> SELECT ROUND(-1.23), ROUND(-1.58), ROUND(1.58);
```

+-----+		
ROUND(-1.23)	ROUND(-1.58)	ROUND(1.58)
+-----+		
-1	-2	2
+-----+		

## 4. MySQL 함수

### 4.2 숫자형 함수

❑ `ROUND(x,d)` : `x`값 중에서 소수점 `d`자리에 가장 근접한 수로 반환한다.

❑ 예제

```
mysql> SELECT ROUND(1.298,1),ROUND(1.298,0);
```

+-----+-----+	
ROUND(1.298,1)	ROUND(1.298,0)
+-----+-----+	
1.3	1
+-----+-----+	



## 4. MySQL 함수

### 4.2 숫자형 함수

❑ `POW(x,y)` `POWER(x,y)` :  $x$ 의  $y$  제곱 승을 반환한다.

❑ 예제

```
mysql> SELECT POW(2,2),POWER(2,-2);
```

```
+-----+-----+
| POW(2,2) | POWER(2,-2) |
+-----+-----+
| 4.000000 | 0.250000    |
+-----+-----+
```

## 4. MySQL 함수

### 4.2 숫자형 함수

□ **SIGN(x)** : x=음수이면 -1을, x=0이면 0을, x=양수이면 1을 출력한다.

□ 예제

```
mysql> SELECT SIGN(-32), SIGN(0), SIGN(234);
```

+-----+-----+-----+		
SIGN(-32)	SIGN(0)	SIGN(234)
+-----+-----+-----+		
-1	0	1
+-----+-----+-----+		

## 4. MySQL 함수

### 4.2 숫자형 함수

❑ **GREATEST(x,y,...) : 가장 큰 값을 반환한다.**

❑ 예제

```
mysql> SELECT GREATEST(2,0),GREATEST(4.0,3.0,5.0),GREATEST("B","A","C");
```

+-----+-----+-----+			
GREATEST(2,0)	GREATEST(4.0,3.0,5.0)	GREATEST("B","A","C")	
+-----+-----+-----+			
2	5.0	C	
+-----+-----+-----+			

## 4. MySQL 함수

### 4.2 숫자형 함수

❑ **LEAST(x,y,...) : 가장 작은 값을 반환한다.**

❑ 예제

```
mysql> SELECT LEAST(2,0),LEAST(34.0,3.0,5.0),LEAST("b","A","C");
```

LEAST(2,0)	LEAST(34.0,3.0,5.0)	LEAST("b","A","C")
0	3.0	A

## 4. MySQL 함수

### 4.3 날짜형 함수

❑ **CURDATE( ), CURRENT\_DATE** : 오늘 날짜를 YYYY-MM-DD나 YYYYMMDD 형식으로 반환한다.

❑ **예제**

```
mysql> SELECT CURDATE( ), CURRENT_DATE;
```

+-----+-----+	
CURDATE( )	CURRENT_DATE
+-----+-----+	
+-----+-----+	

## 4. MySQL 함수

### 4.3 날짜형 함수

❑ **CURTIME( ) CURRENT\_TIME** : 현재 시각을 HH:MM:SS나 HHMMSS 형식으로 반환한다.

#### ❑ 예제

```
mysql> SELECT CURTIME(),CURRENT_TIME;
```

```
+-----+-----+
| CURTIME() | CURRENT_TIME |
+-----+-----+
| 14:31:51  | 14:31:51     |
+-----+-----+
```

## 4. MySQL 함수

### 4.3 날짜형 함수

#### □ NOW( )

`SYSDATE( )`

`CURRENT_TIMESTAMP` : 오늘 현시각을 `YYYY-MM-DD HH:MM:SS`나  
`YYYYMMDDHHMMSS` 형식으로 반환한다.

#### □ 예제

```
mysql> SELECT NOW(),SYSDATE(),CURRENT_TIMESTAMP;
```

+	-----+	-----+	-----+
	NOW( )	SYSDATE( )	CURRENT_TIMESTAMP
+	-----+	-----+	-----+
	09:33:38	09:33:38	09:33:38
+	-----+	-----+	-----+

## 4. MySQL 함수

### 4.3 날짜형 함수

❑ **DATE\_FORMAT(date,format) : 입력된 date를 format 형식으로 반환한다**

❑ **예제**

```
mysql> SELECT DATE_FORMAT(CURDATE(), '%W %M %Y');
```

```
+-----+
| DATE_FORMAT(CURDATE(), '%W %M %Y') |
+-----+
| Monday November                    |
+-----+
```

```
mysql> SELECT DATE_FORMAT(CURDATE(), '%Y.%m.%d');
```

```
+-----+
| DATE_FORMAT(CURDATE(), '%Y.%m.%d') |
+-----+
|                                     |
+-----+
```



## 4. MySQL 함수

### 4.3 날짜형 함수

□ **PERIOD\_DIFF(p1,p2)** : YYMM이나 YYYYMM으로 표기되는 p1과 p2의 차이가 개월을 반환 한다.

□ **예제** : 각 직원들에 대해 직원이름과 근무개월수 출력

```
SELECT concat(first_name, ' ', last_name) AS name,  
       PERIOD_DIFF( DATE_FORMAT(CURDATE(), '%Y%m'),  
                   DATE_FORMAT(hire_date, '%Y%m') )  
FROM employees
```

### 4.3 날짜형 함수

- `DATE_ADD(date, INTERVAL expr type)`  
`DATE_SUB(date, INTERVAL expr type)`  
`ADDDATE(date, INTERVAL expr type)`  
`SUBDATE(date, INTERVAL expr type) :`

**날짜 date에 type 형식으로 지정한 expr 값을 더하거나 뺀다.**  
**`DATE_ADD()`와 `ADDDATE()`는 같은 동작이고,**  
**`DATE_SUB()`와 `SUBDATE()`는 같은 의미이다.**

- **예제 : 각 직원들은 입사 후 6개월이 지나면 근무평가를 한다. 각 직원들에 이름, 입사일, 최초 근무평가일은 언제인지 출력 ( 다음 페이지의 참고 예제를 참고 )**

## 4. MySQL 함수

### 4.4 CAST (형 변환)

CAST 함수는 type을 변경(지정)하는데 유용하다.

CAST 함수의 사용법 : `CAST(expression AS type)` 또는  
`CONVERT(expression, type)`

MySQL 타입: `BINARY`  
`CHAR`  
`DATE`  
`DATETIME`  
`SIGNED {INTEGER}`  
`TIME`  
`UNSIGNED {INTEGER}`

## 4. MySQL 함수

### 4.4 CAST (형 변환)

#### □ 예제

```
mysql> select cast(now() as date);
```

```
+-----+  
| cast(now() as date) |  
+-----+  
|                     |  
+-----+  
1 row in set (0.00 sec)
```

```
mysql> select cast(1-2 as unsigned);
```

```
+-----+  
| cast(1-2 as unsigned) |  
+-----+  
| 18446744073709551615 |  
+-----+
```

### 4.4 CAST (형 변환)

#### □ 예제 (cont'd)

```
mysql> select cast(cast(1-2 as unsigned) as signed);
```

```
+-----+
| cast(cast(1-2 as unsigned) as signed) |
+-----+
|                                     -1 |
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> select cast(1 as unsigned) -2.0;
```

```
+-----+
| cast(1 as unsigned) -2.0 |
+-----+
|                   -1.0 |
+-----+
```

## 4. MySQL 함수

### 4.4 그룹 함수

COUNT(expr)	non-NULL인 row의 숫자를 반환
COUNT(DISTINCT expr,[expr...])	non-NULL인 중복되지 않은 row의 숫자를 반환
COUNT(*)	row의 숫자를 반환
AVG(expr)	expr의 평균값을 반환
MIN(expr)	expr의 최소값을 반환
MAX(expr)	expr의 최대값을 반환
SUM(expr)	expr의 합계를 반환
GROUP_CONCAT(expr)	그룹에서 concatenated한 문자를 반환
VARIANCE(expr)	분산
<hr/> <hr/>	
STDDEV(expr)	expr의 표준 편차를 반환

## 4. MySQL 함수

### 4.4 그룹 함수

- 예제 : salaries 테이블에서 사번이 10060인 직원의 급여 평균과 총합계를 출력

```
SELECT AVG(salary) , SUM(salary)
FROM salaries
WHERE emp_no = '10060';
```

- 예제 : 이 예제 직원의 최저 임금을 받은 시기와 최대 임금을 받은 시기를 각 각 출력해보세요.

?

## 4. MySQL 함수

### 4.4 그룹 함수

□ **예제** : dept\_emp 테이블에서 d008에 근무하는 인원수는

```
SELECT count(*)  
  FROM dept_emp  
 WHERE dept_no = 'd008';
```



# MySQL

## 4. 데이터 검색 II

1. 데이터의 분류(Grouping)
2. 조인(Join)
3. 서브쿼리(Subquery)

## 1. 데이터의 분류(grouping)

### 1.1 SELECT 구문의 전체 문형

SELECT(DISTINCT) **칼럼명**(ALIAS)  
FROM **테이블명**  
WHERE **조건식**  
GROUP BY **칼럼명**  
HAVING **조건식**  
ORDER BY **칼럼이나 표현식** (ASC 또는 DESC)

<b>GROUP BY</b>	전체 데이터를 소그룹으로 나루 칼럼을 명시
<b>HAVING</b>	<b>GROUP</b> 에 대한 조건을 기술

## 1. 데이터의 분류(grouping)

### 1.2 GROUP BY 절의 사용

□ **GROUP BY** 절에 기술된 칼럼이 반드시 **SELECT**절 뒤에 올 필요는 없으나 **SELECT** 문 결과의 의미를 명확히 하기 위해 기술하는 것이 좋다.

□ 예제1 : 각 사원별로 평균연봉 출력

```
SELECT emp_no, AVG(salary)
      FROM salaries
GROUP BY emp_no
```

□ 예제 2: 각 현재 **Manager** 직책 사원에 대한 평균 연봉은?

## 1. 데이터의 분류(grouping)

### 1.3 GROUP 함수 사용 오류

#### ❑ SELECT 절

**SELEC**절에 그룹함수가 오면 나머지 컬럼은 **GROUP BY** 절에 기술 되어야 한다.

즉, **SELECT**절에 그룹함수가 오거나, **GROUP BY**절 이하에 기술된 컬럼이 오면 나머지 칼럼은 **SELECT** 절 뒤에 기술할 수 없다.

#### ❑ 예제 3: 사원별 몇 번의 직책 변경이 있었는지 조회

```
SELECT emp_no, COUNT(title)
FROM titles
```

-> 결과?

-> 바르게 수정해 보세요.

## 1. 데이터의 분류(grouping)

### 1.3 GROUP 함수 사용 오류

#### ❑ WHERE 절

**GROUP**에 대한 조건은 **WHERE**절에서 기술할 수 없고 **HAVING**절에서 기술 되어야 한다.

#### ❑ 예제4 : 각 사원별로 평균연봉 출력하되 50,000불 이상인 직원만 출력

```
SELECT emp_no, AVG(salary)
FROM salaries
WHERE AVG(salary) > 50000
GROUP BY emp_no
```

-> 결과?

-> 바르게 수정해 보세요.

## 1. 데이터의 분류(grouping)

### 1.4 GROUP BY 절에 여러 컬럼 사용

- **GROUP BY** 절에 여러 컬럼을 사용함으로써 여러 **Subgroup**으로 데이터를 분류할 수 있다.
- 예제5: 현재 직책별로 평균 연봉과 인원수를 구하되 직책별로 인원이 100명 이상인 직책만 출력하세요.
- 예제6: 현재 부서별로 현재 직책이 **Engineer**인 직원들에 대해서만 평균급여를 구하세요.
- 예제7: 현재 직책별로 급여의 총합을 구하되 **Engineer**직책은 제외하세요 단, 총합이 2,000,000,000이상인 직책만 나타내며 급여총합에 대해서 내림차순(**DESC**)로 정렬하세요.

## 2. 조인(join)

### 2-1. 조인의 개념

- 하나 이상의 테이블로부터 연관된 데이터를 검색해 오는 방법
- **Primary Key(PK)** 와 **Foreign Key(FK)** 값의 연관에 의해 **JOIN**이 성립  
(아닌 경우도 있다. 논리적인 값들의 연관으로만 성립 가능)
- 조인의 기본 유형
  - **equijoin** : **=(equal)** 연산자를 사용하는 조인
  - **inner join** : 조인 조건을 만족하는 행에 대해서만 결과값이 나오는 조인
  - **outter join** : 조인 조건을 만족하지 않아도 출력이 가능해야하는 결과를 얻고자 할 때 사용

## 2. 조인(join)

### 2-2. EQUIJOIN 예

□ 컬럼에 있는 값이 정확하게 일치하는 경우에 = 연산자를 사용하여 **JOIN**

name	emp_no	emp_no	title
Facello Georgi	10001	10001	Manager
Simmel Bezal	10002	10002	Senior Staff
Peac Sunmant	10003	10003	Staff
Sluis Mary	10004	10004	Senior Engineer
...	...	10001	Senior Staff
		10002	Staff

**employees** **titles**

각 사원의 이름과 그 사원이 근무 했던 부서들을 알고  
싶다면 테이블 하나로는 원하는 데이터를 검색할 수 없다.

원래 테이블에 있던 데이터에서 점선 (=관계)을 따라 합쳐보면...



## 2. 조인(join)

### 2-2. EQUIJOIN 예

□ **Equal** 관계만으로 하나의 테이블로 결합한 결과

name	emp_no
Facello Georgi	10001
Simmel Bezal	10002
Peac Sunmant	10003
Sluis Mary	10004
Facello Georgi	10001
Simmel Bezal	10002

emp_no	title
10001	Manager
10002	Senior Staff
10003	Staff
10004	Senior Engineer
10001	Senior Staff
10002	Staff

**employees**      **titles**

## 2. 조인(join)

### 2-2. EQUIJOIN의 문형

```
SELECT 테이블명.컬럼명, 테이블명.컬럼명. ...  
FROM 테이블1, 테이블2  
WHERE 테이블1.컬럼1 = 테이블2.컬럼2
```

테이블명.컬럼명	검색해올 데이터가 어디에서 오는지 테이블과 컬럼을 밝혀둔다.
테이블1.컬럼1 = 테이블2.컬럼2	두 테이블간에 논리적으로 값을 연결시키는 컬럼간의 조건을 기술한다.

## 2. 조인(join)

### 2-2. EQUIJOIN의 문형

- 컬럼에 있는 값들이 정확히 일치하는 경우에 = 연산자를 사용해서 조인
- 일반적으로 **PK-FK** 관계에 의하여 **JOIN**이 성립
- **WHERE** 절 혹은 **ON**절을 이용
- 액세스 효율을 향상시키고 좀더 명확히 하기 위해 컬럼이름앞에 테이블 이름을 밝힌다.
- 같은 이름의 컬럼이 조인대상 테이블에 존재하면 반드시 컬럼이름앞에 테이블이름을 밝혀주어야 한다.
- **JOIN**을 위한 테이블이 **N**개라고 하면 **JOIN**을 위한 최소한의 =조건은 **N-1**이다.

## 2. 조인(join)

### 2-2. EQUIJOIN의 문형

- 예제 8: **employees** 테이블과 **titles** 테이블을 **join**하여 직원의 이름과 직책을 모두 출력 하세요.

```
SELECT  CONCAT(employees.last_name, ' ', employees.first_name) AS name,  
        employees.emp_no,  
        titles.emp_no,  
        titles.title  
FROM    employees, titles  
WHERE   employees.emp_no = titles.emp_no;
```

## 2. 조인(join)

### 2-3. Alias 사용

- 테이블명.칼럼명 으로 기술할 때, 테이블명이 길어지는 경우는 많은 시간이 소요되므로 **ALIAS**를 지정하고 **ALIAS**가 지정되면 지정된 **ALIAS**만 사용해야 한다.
- **ALIAS**를 사용하면 칼럼헤딩에 대한 애매함을 피할 수 있다.
- 예제9: 예제 8번에서 각 컬럼과 테이블에 **ALIAS** 를 지정하여 사원의 이름과 직책을 출력하세요.

## 2. 조인(join)

### 2-4. 추가적인 조건 기술

- ❑ **WHERE**절에 **JOIN**조건 이외의 추가적인 조건을 가질 수 있다.
- ❑ 조인을 만족하는 데이터중 특정행만 선택하여 결과를 얻고 싶을 때 추가조건을 **AND**로 연결한다.
- ❑ 예제10:  
employees 테이블과 titles 테이블을 join하여 직원의 이름과 직책을 출력하되 여성 엔지니어만 출력하세요.

### 2-5. Cartesian Join

- **Join**에 대한 조건이 생략되거나 잘못 기술되어 한 테이블에 있는 모든 행들이 다른 테이블에 있는 모든 행들과 **Join**이 되어서 얻어진 경우를 **Cartesian Product**한다
- **Cartesian Product**를 얻지 않기 위해서 반드시 **WHERE** 절을 써 준다.

## 2. 조인(join)

### 2-6. Natural Join

□ 두 테이블에 공통 칼럼이 있는 경우 별다른 조인 조건없이 공통 칼럼처럼 묵시적으로 조인이 되는 유형

□ ANSI / ISO SQL1999를 따르는 ANSI JOIN 문법.

□ 예제11 : 앞의 예제의 Natural Join 법

```
SELECT      emp_no,  
            CONCAT(last_name, ' ', first_name) AS name,  
            title  
  
FROM        employees  
  
NATURAL JOIN titles
```



## 2. 조인(join)

### 2-7. JOIN~ USING

- ❑ **Natural join**의 문제점 : 조인하고자 하는 두 테이블에 같은 이름이 칼럼이 많을 때
- ❑ 위와 같을 시 특정한 칼럼으로만 조인하고 싶다면 **USING**절을 사용해서 기술한다.
- ❑ **ANSI / ISO SQL1999**를 따르는 **ANSI JOIN** 문법.
- ❑ **예제11** : 앞의 예제를 **JOIN~ USING**을 사용해서 같은 결과가 나오도록 해 보세요.

```
SELECT b.emp_no,  
       CONCAT( a.last_name, ' ', a.first_name ) AS name,  
       b.title  
FROM   employees a  
       JOIN titles b USING( emp_no )
```

## 2. 조인(join)

### 2-8. JOIN ~ ON

- 공통된 이름의 칼럼이 없는 경우 가장 보편적으로 사용할 수 있는 유형
- **WHERE** 절에 일반조건 만 쓸 수 있게하고 조인 조건은 **ON**에 두어 보다 의미를 명확히 하고 알아보기 도 쉽다.
- ANSI / ISO SQL1999를 따르는 ANSI JOIN 문법.

- 예제12: 예제10 번을 JOIN ~ ON를 사용해서 수정해 보세요.

```
SELECT a.emp_no,  
       CONCAT(a.last_name, ' ', a.first_name) AS name,  
       b.title  
FROM employees a  
     JOIN titles b ON ( a.emp_no = b.emp_no )  
WHERE a.gender = 'f'  
      AND b.title = 'Engineer' ;
```

### 2-9. EQUIJOIN 실습문제

- 실습문제 1: 현재 회사 상황을 반영한 직원별 근무부서를 사번, 직원 전체이름, 근무부서 형태로 출력해 보세요.
- 실습문제 2: 현재 회사에서 지급되고 있는 급여체계를 반영한 결과를 출력하세요. 사번, 전체이름, 연봉 이런 형태로 출력하세요.

### 3. 서브쿼리(Subquery)

## 3-1. 서브쿼리의 개념

- 하나의 **SELECT(SQL)**문 안에 포함되어 있는 **SELECT** 문장
- 여러 절에서 사용할 수 있으나 **SELECT**문안에 포함되어 있는 것이 일반적

```
SELECT ..  
FROM ..  
WHERE ..
```

```
( SELECT ..  
  FROM ..  
  WHERE .. );
```

### 3. 서브쿼리(Subquery)

## 3-2. Subquery의 문형

```
SELECT 검색할 컬럼들
FROM 테이블명
WHERE 형식 연산자 ( SELECT 검색할 컬럼들
                    FROM 테이블 명
                    ... )
```

- ❑ 서브쿼리는 괄호로 묶여 있어야 한다.
- ❑ 서브쿼리 내에서 **ORDER BY** 절을 포함할 수 없음
- ❑ 서브쿼리는 거의 모든 구문에서 사용이 가능(**GROUP BY** 절 제외)
- ❑ 형식 연산자
  - 단일행 연산자 ( =, >, >=, <, <=, <> )
  - 복수행 연산자 ( IN, ANY, ALL, NOT IN )

### 3. 서브쿼리(Subquery)

## 3-3. 단일행 서브쿼리 예제

- 메인 쿼리로 전달되는 행이 단 하나인 경우
- 단일행 연산자를 사용한다.
- 예제: 현재 **Fai Bale**이 근무하는 부서에서 근무하는 직원의 사번, 전체 이름을 출력해보세요.

#### 1. dept\_id를 입력 받아, 직원이름, 부서 이름을 출력하는 SQL

```
SELECT a.emp_no,  
       concat( a.first_name, ' ', a.last_name) AS name,  
       b.dept_name  
FROM employees a, departments b, dept_emp c  
WHERE a.emp_no = c.emp_no  
      AND c.dept_no = b.dept_no  
      AND c.to_date = '9999-01-01'  
      AND c.dept_no = Fai Bale 가 근무하는 부서의 부서의 ID
```

### 3. 서브쿼리(Subquery)

---

## 3-3. 단일행 서브쿼리 예제

2. 이름(**Fai Bale**)을 입력 받아 **dept\_id**를 출력하는 **SQL**

```
SELECT c.dept_no  
FROM employees a,  
      dept_emp c  
WHERE a.emp_no = c.emp_no  
      AND c.to_date = '9999-01-01'  
      AND concat( a.first_name, ' ', a.last_name) = 'Fai Bale'
```

### 3. 서브쿼리(Subquery)

## 3-3. 단일행 서브쿼리 예제

### 3. 전체 SQL

```
SELECT a.emp_no,  
       concat( a.first_name, ' ', a.last_name) AS name,  
       b.dept_name  
FROM employees a, departments b, dept_emp c  
WHERE a.emp_no = c.emp_no  
      AND c.dept_no = b.dept_no  
      AND c.to_date = '9999-01-01'  
      AND c.dept_no = ( SELECT c.dept_no  
                        FROM employees a,  
                        dept_emp c  
                        WHERE a.emp_no = c.emp_no  
                              AND c.to_date = '9999-01-01'  
                              AND concat( a.first_name, ' ', a.last_name) = 'Fai  
Bale' );
```



### 3. 서브쿼리(Subquery)

---

## 3-4. 단일행 서브쿼리 실습문제

- 실습문제 1: 현재 전체사원의 평균 연봉보다 적은 급여를 받는 사원의 이름, 급여를 나타내세요.
  
- 실습문제 2: 현재 가장 적은 평균 급여를 받고 있는 직책에해서 평균 급여를 구하세요

### 3. 서브쿼리(Subquery)

#### 3-4. 다중행 서브쿼리 예제

- 메인 쿼리로 전달되는 행이 여러 개인 경우
- 다중 행 연산자 **IN, ANY, ALL**를 사용한다.
- 예제: 현재 급여가 **50000** 이상인 직원 이름 출력

```
SELECT *  
FROM employees  
where emp_no =any ( SELECT emp_no  
                        FROM salaries  
                        WHERE salary < 50000  
                        AND to_date = '9999-01-01' );
```

# MySQL

## 5. 테이블 생성 및 데이터 입력, 수정, 삭제

1. MySQL 데이터 타입
2. 데이터 정의어(DDL)
3. 데이터 조작어(DML)

# 1. MySQL의 데이터 유형

## 1.1 MySQL 데이터 타입

<b>TINYINT(M)</b>	부호 있는 수는 -128~127까지, 부호 없는 수는 0~255까지 표현. 1 바이트
<b>SMALLINT(M)</b>	부호 있는 수는 -32768~32767까지, 부호 없는 수는 0~65535까지 표현. 2 바이트
<b>MEDIUMINT(M)</b>	부호 있는 수는 -8388608~8388607까지, 부호 없는 수는 0~16777215까지 수를 표현. 3 바이트
<b>INT(M) or INTEGER(M)</b>	부호 있는 수는 -2147483648~2147483647까지, 부호 없는 수는 0~4294967295까지 . 4 바이트
<b>BIGINT(M)</b>	부호있는 수는 -92233720036854775808 ~ 92233720036854775808 부호 없는 수는 0~18446744073709551615
<b>FLOAT(M,D)</b>	부동 소수점을 나타낸다. 언제나 부호 있는 수임. (-3.402823466E+38~3.402823466E+38)
<b>DOUBLE(M,D)</b>	2배 정밀도를 가진 부동 소수점. (-1.79769313486231517E+308~6931348623157E+308)
<b>DATE</b>	날짜를 표현하는 타입. '9999-12-31'. 3 바이트
<b>DATETIME</b>	날짜와 시간을 같이 나타내는 타입. '9999-12-31 23:59:59'. 8 바이트
<b>TIMESTAMP</b>	'1970-01-01 00:00:00' 부터 2037년까지 나타낼 수 있다. 4 바이트

# 1. MySQL의 데이터 유형

## 1.1 MySQL 데이터 타입

<b>TIME</b>	시간을 나타낸다. '-839:59:59' 부터 '838:59:59'까지 나타낼 수 있다.
<b>YEAR</b>	년도를 나타낸다. 1901년부터 2155년, 0000년을 나타낼 수 있다.
<b>CHAR(M)</b>	고정 길이를 갖는 문자열을 저장할 수 있다. M은 1 부터 255 까지 이다.
<b>VARCHAR(M)</b>	CHAR는 고정 길이인 반면 VARCHAR는 가변 길이이다.
<b>TINYBLOB, TINYTEXT</b>	255개의 문자를 저장할 수 있다.
<b>BLOB,TEXT</b>	63,535개의 문자를 저장할 수 있다.
<b>MEDIUMBLOB, MEDIUMTEXT</b>	16,777,215개의 문자를 저장할 수 있다.
<b>LOBLOB, LONGTEXT</b>	4,294,967,295(4기가)개의 문자를 저장할 수 있다.

## 2. 데이터 정의를어 ( DDL )

### 2.1 Table 생성

```
create table 테이블명(  
    필드명1 타입 [NULL | NOT NULL][DEFAULT ][AUTO_INCREMENT],  
    필드명2 타입 [NULL | NOT NULL][DEFAULT ][AUTO_INCREMENT],  
    필드명3 타입 [NULL | NOT NULL][DEFAULT ][AUTO_INCREMENT],  
    .....  
    PRIMARY KEY(필드명)  
);
```

- ❑ 데이터 형 외에도 속성값의 빈 값 허용 여부는 NULL 또는 NOT NULL로 설정
- ❑ DEFAULT 키워드와 함께 입력하지 않았을 때의 초기값을 지정할 수 있다.
- ❑ 입력하지 않고 자동으로 1씩 증가하는 번호를 위한 AUTO\_INCREMENT

## 2. 데이터 정의어 ( DDL )

### 2.1 Table 생성

[예제1] 다음과 같은 TABLE CHART를 보고 테이블 member를 작성하세요.

컬럼명	데이터 타입	길이	NOT NULL	default	P.K	auto_increment
no	INT		YES		YES	
email	CHAR	50	YES			
passwd	CHAR	20	YES			
name	CHAR	25	NO			
department_name	CHAR	25	NO			

## 2. 데이터 정의어 ( DDL )

### 2.2 Table 수정 ( column 추가 / 삭제 )

```
alter table 테이블명  
    add 필드명 타입 [NULL | NOT NULL][DEFAULT ][AUTO_INCREMENT];  
  
alter table 테이블명  
    drop 필드명;
```



## 2. 데이터 정의어 ( DDL )

### 2.2 Table 수정 ( column 추가 / 삭제 )

[예제2]

member 테이블에 juminbunho char 타입, 반드시 입력되어야 하는 칼럼을 추가 하세요.  
desc member로 추가 결과를 확인해 보세요.

[예제3]

예제 2에서 추가했던, juminbunho 칼럼을 삭제 하세요.  
desc member로 결과를 확인해 보세요.

[예제4]

member 테이블에 회원 가입날짜 DATETIME 타입의 join\_date 이름의 칼럼을 추가 하세요.  
반드시 입력되어야 하는 컬럼입니다.

## 2. 데이터 정의어 ( DDL )

### 2.2 Table 수정 ( column 변경 )

```
alter table 테이블명  
  change 필드명 새필드명 타입 [NULL | NOT NULL][DEFAULT ][AUTO_INCREMENT];
```

❑ change 키워드를 사용하고 칼럼을 새롭게 재정의 (이름부터 속성까지 전부)

[예제5]

Member 테이블의 Primary Key no에 자동 1씩기 증가하는 속성을 추가 해 보세요.

```
ALTER TABLE member
```

```
  CHANGE no no INT NOT NULL AUTO_INCREMENT
```

## 2. 데이터 정의어 ( DDL )

---

### 2.2 Table 수정 ( column 변경 )

[예제6]

member의 컬럼 deparment\_name의 이름이 길어 dept\_name으로 바꿀려고 합니다.  
수정해 보세요.

[예제7]

테이블 member의 name 컬럼의 길이 제한을 10자로 줄이세요.

## 2. 데이터 정의어 ( DDL )

### 2.3 Table 삭제

**drop table 테이블명**

[예제8]

member 테이블의 description을 저장하고 member 테이블을 삭제하세요.  
그리고 다시, 마지막 수정이 적용된 member테이블을 한 번에 생성해 보세요.

## 2. 데이터 정의어 ( DDL )

### 2.4 Table 이름 변경

```
alter table 테이블명 rename 변경이름
```

[예제9]

member 테이블의 이름을 user 로 변경하세요.

### 3. 데이터 조작용어 ( DML )

## 3.1 데이터 삽입 ( INSERT )

```
INSERT INTO 테이블명(필드1, 필드2, 필드3, 필드4, ... )  
VALUES ( 필드1의 값, 필드2의 값, 필드3의 값, 필드4의 값, ... )
```

```
INSERT INTO 테이블명  
VALUES ( 필드1의 값, 필드2의 값, 필드3의 값, 필드4의 값, ... )
```

- ❑ 필드명을 지정해주는 방식은 디폴트 값이 세팅되는 필드는 생략할 수 있다.
- ❑ 필드명을 지정해주는 방식은 추 후, 필드가 추가/변경/수정 되는 변경에 유연하게 대처 가능
- ❑ 필드명을 생략했을 경우에는 모든 필드 값을 반드시 입력해야 한다.

### 3. 데이터 조작용어 ( DML )

---

## 3.1 데이터 삽입 ( INSERT )

[예제10]

member의 컬럼에 각자의 데이터를 입력해 보세요.

주) 한글 입력전에 set names utf8 를 입력 하세요.

auto\_increment 필드는 null로 입력해야 합니다.

passwd 컬럼에는 password 함수를 사용해서 입력합니다.

join\_date 에는 sysdate() 함수를 사용하세요.

### 3. 데이터 조작용어 ( DML )

## 3.2 데이터 변경 ( UPDATE )

```
UPDATE 테이블명  
SET 필드1=필드1의값, 필드2=필드2의값, 필드3=필드3의값, ...  
WHERE 조건식
```

- ❑ 조건식을 통해 특정 row만 변경할 수 있다.
- ❑ 조건식을 주지 않으면 전체 로우가 영향을 미치니 조심해서 사용하도록 한다.

[예제11]

user 번호가 1인 사용자의 이름을 영문으로 바꾸고, join\_date을 현재 시간이 적용되도록 수정하세요.



### 3. 데이터 조작용어 ( DML )

## 3.2 데이터 삭제 ( DELETE )

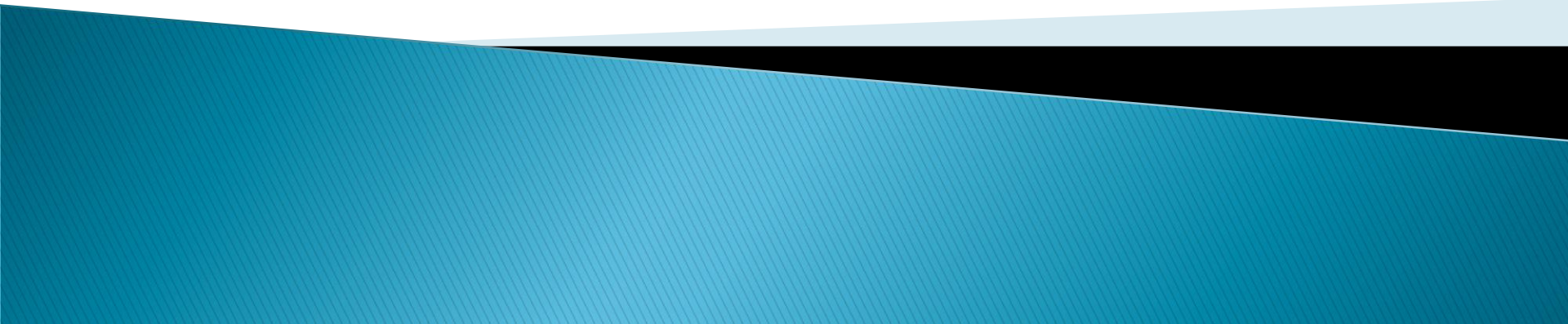
```
DELETE  
FROM 테이블명  
WHERE 조건식
```

- ❑ 조건식을 통해 특정 row만 삭제할 수 있다.
- ❑ 조건식을 주지 않으면 전체 로우가 영향을 미치니 조심해서 사용하도록 한다.

[예제12]

user 번호가 1인 사용자를 삭제하여라,

# DBMS 데이터베이스 설계



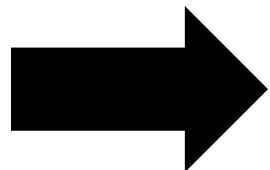
# 1. 데이터베이스 설계

- ▶ 어떤 데이터를 저장할 것인가?
- ▶ 음원검색 ( 네이버 뮤직 )

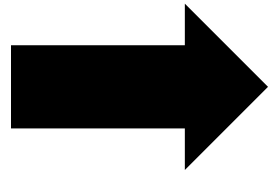
가수이름	앨범 타이틀	배급사	노래
	2집 Last Fantasy	로엔 엔터테인먼트	비밀, 잠자는 숲 속의 왕자, 너랑 나, 삼촌
	ALONE	로엔 엔터테인먼트	Come Closer, 나혼자(Alone), No Mercy, Lead Me
버스커 버스커	1집 버스커 버스커	씨제이이엔엠	봄바람, 첫사랑, 여수 밤바다, 벚꽃 엔딩, 골목길
	ALIVE	㈜케이엠피홀딩스	BLUE, 사랑먼지, BAD BOY, 재미없어, FANTASTIC BABY
	So Cool	로엔 엔터테인먼트	So Cool, Girls Do It, 약한 남자 싫어, 니까짓게
	Sherlock	(주)케이엠피홀딩스	Sherlock, Clue, Note, 알람시계

# 1. 데이터베이스 설계 (cont'd)

- ▶ 데이터베이스 설계를 하는 주된 목적



**중복성 제거**



**정규화 (Normalization)**

## 2. 데이터 모델 (Data Model)

- ▶ 데이터베이스 설계를 도식화한 도표
- ▶ 엔티티(entity), 속성(attribute), 관계(relation)  
3가지 요소로 구성

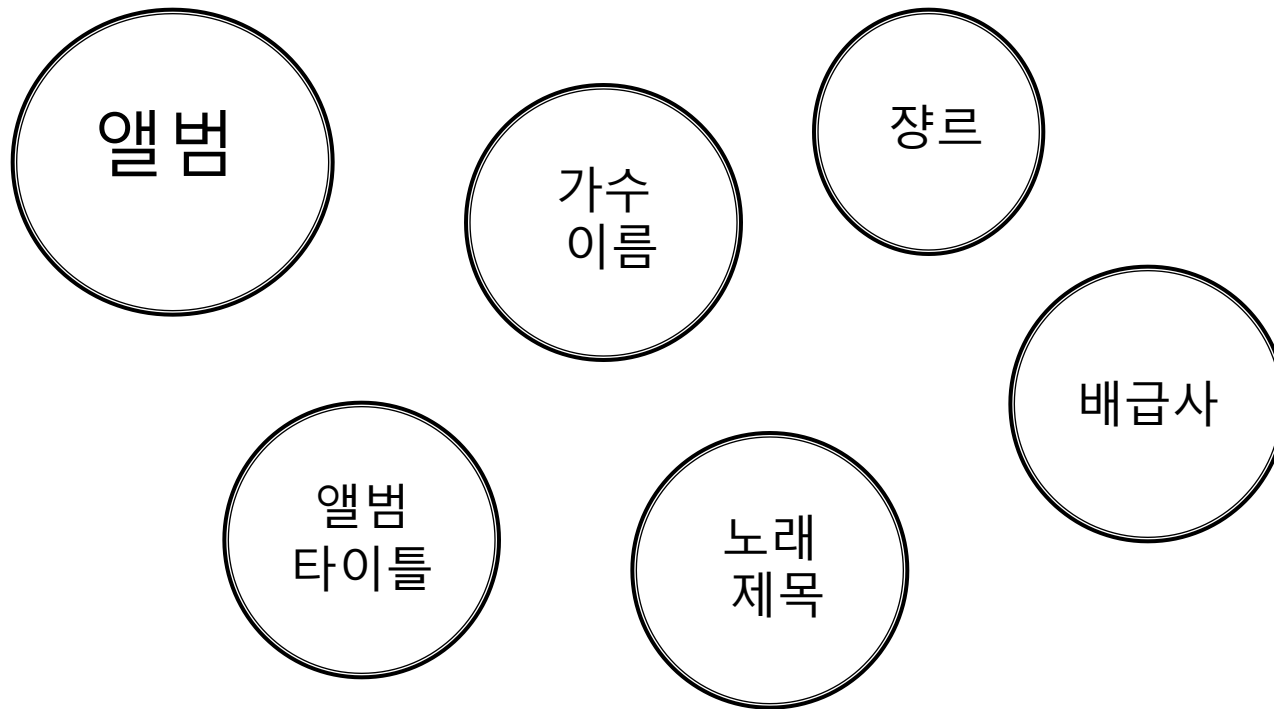
## 2. 데이터 모델 (cont'd)

### ▶ 엔티티(entity)

- 데이터가 저장되어야 하는 중요한 사물, 물체
- 엔티티에 대한 정보는 **속성, 관계**의 폼으로 저장
- 엔티티 속성(attribute)
  1. 엔티티의 정보들 (0 혹은 여러개)
  2. 엔티티 인스턴스(테이블에서의 열)는 각각 정확하게 하나의 값
  3. 또는 NULL(empty)일 수 있다.
  4. 각 속성의 값은 숫자, 문자열, 날짜, 시간, 혹은 다른 기본적인 데이터의 값

## 2. 데이터 모델 (cont'd)

- ▶ 어떤 것이 엔티티 이고 속성 일까?



### 3. 첫 번째 데이터 모델



- ▶ 엔티티 네이밍 규칙

엔티티 이름은 단수 - 각 엔티티는 하나의 인스턴스로 명명



### 3. 첫 번째 데이터 모델 (cont'd)

앨범
앨범 타이틀 가수 이름 배급사 이름 노래

## 4. 정규화

- ▶ E.F.Codd의 정규화 개념 (1970년대)
- ▶ 오늘날도 정규화의 목적은 동일
  - 데이터의 중복성을 제거
  - 갱신이상(update anomalies) 회피
- ▶ 데이터 모델을 좀 더 구체적으로 해준다

## 5. 정규화 - 제 1 정규형(1NF)

- ▶ 하나의 엔티티가 모든 속성들이 하나의 값을 가질 때 “제 1 정규형(1st Normal form)” 이라 한다.
- ▶ 각 속성들은 엔티티의 각 인스턴스를 위한 하나의 값을 가지고 있는지 확인
- ▶ 중복된 속성을 가진 엔티티는 그 안에 최소한 1개 이상의 다른 엔티티가 존재한다는 것을 의미

## 6. 두 번째 데이터 모델

앨범
앨범 타이틀 가수 이름 배급사 이름 노래

노래
노래 제목 노래 길이

1NF로 두 개의 엔티티가 생김

속성의 중복성 제거

두 엔티티간의 연관되는 어떤 방법이 필요

-> 관계(relation) 이 필요

## 7. 유일한 식별자

- ▶ 각각의 엔티티는 ID라고 불리는 유일한 식별자를 가져야 한다.
- ▶ ID는 다음 규칙을 가지는 엔티티의 속성
  - 엔티티의 모든 인스턴스에 유일
  - 인스턴스의 전체 라이프 타임 동안 NULL이 아니어야 한다.
  - 전체 라이프 타임 동안 변하지 않는 값
- ▶ ID는 관계를 모델링 하기위한 중요 요소

## 6. 세 번째 데이터 모델

앨범
<u>앨범ID</u> 앨범 타이틀 가수 이름 배급사 이름 노래

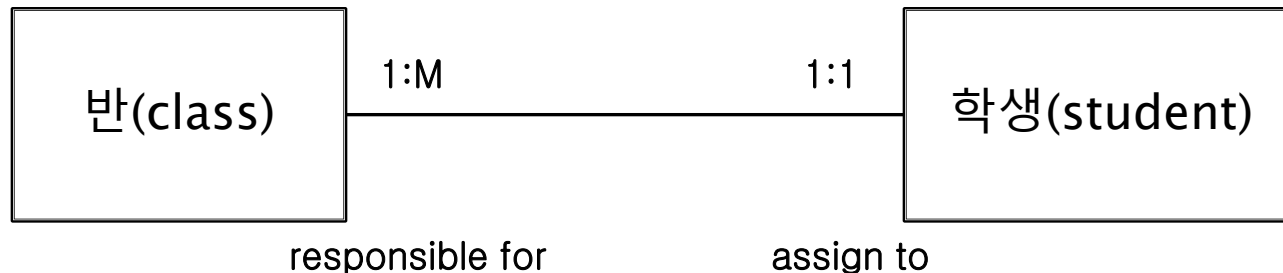
노래
<u>노래ID</u> 노래 제목 노래 길이

식별자 결정의 문제점

엔티티와 전혀 무관한 인위적으로 만들어 냄

## 8. 관계(Relation)

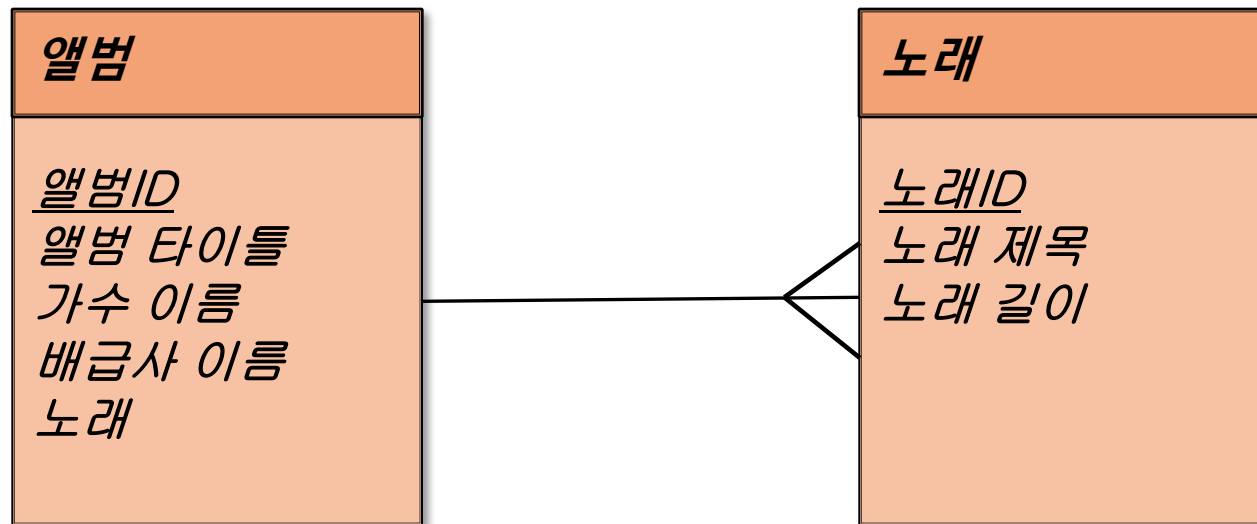
- ▶ 엔티티 안에 존재하는 식별자는 엔티티 사이의 관계를 설정할 수 있게 한다.
- ▶ 관계안에서 설정된 엔티티는 각각에 대해 설명이 되고 관계 양단에는 이름(name)과 정도(degree)를 가진다



Entity1은 entity2와 [일대일 | 일대다] 관계를 갖는다

1. 학생은 반과 ? 관계를 갖는다
2. 반은 학생과 ? 관계를 갖는다

## 6. 네 번째 데이터 모델 (1NF 완성)



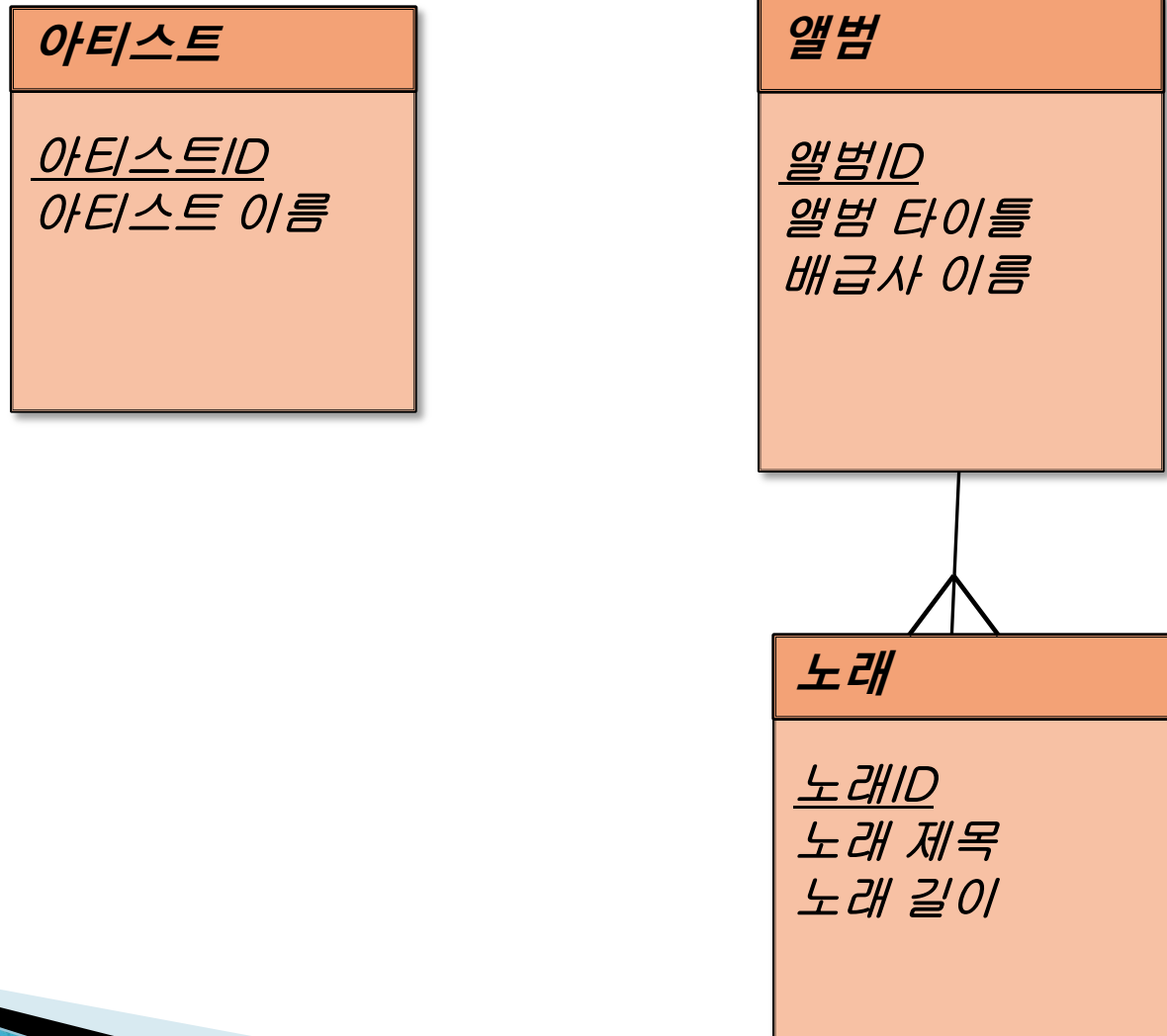
반복되었던 노래 속성이 새로운 엔티티로 정규화  
앨범과 노래간의 관계가 모델화



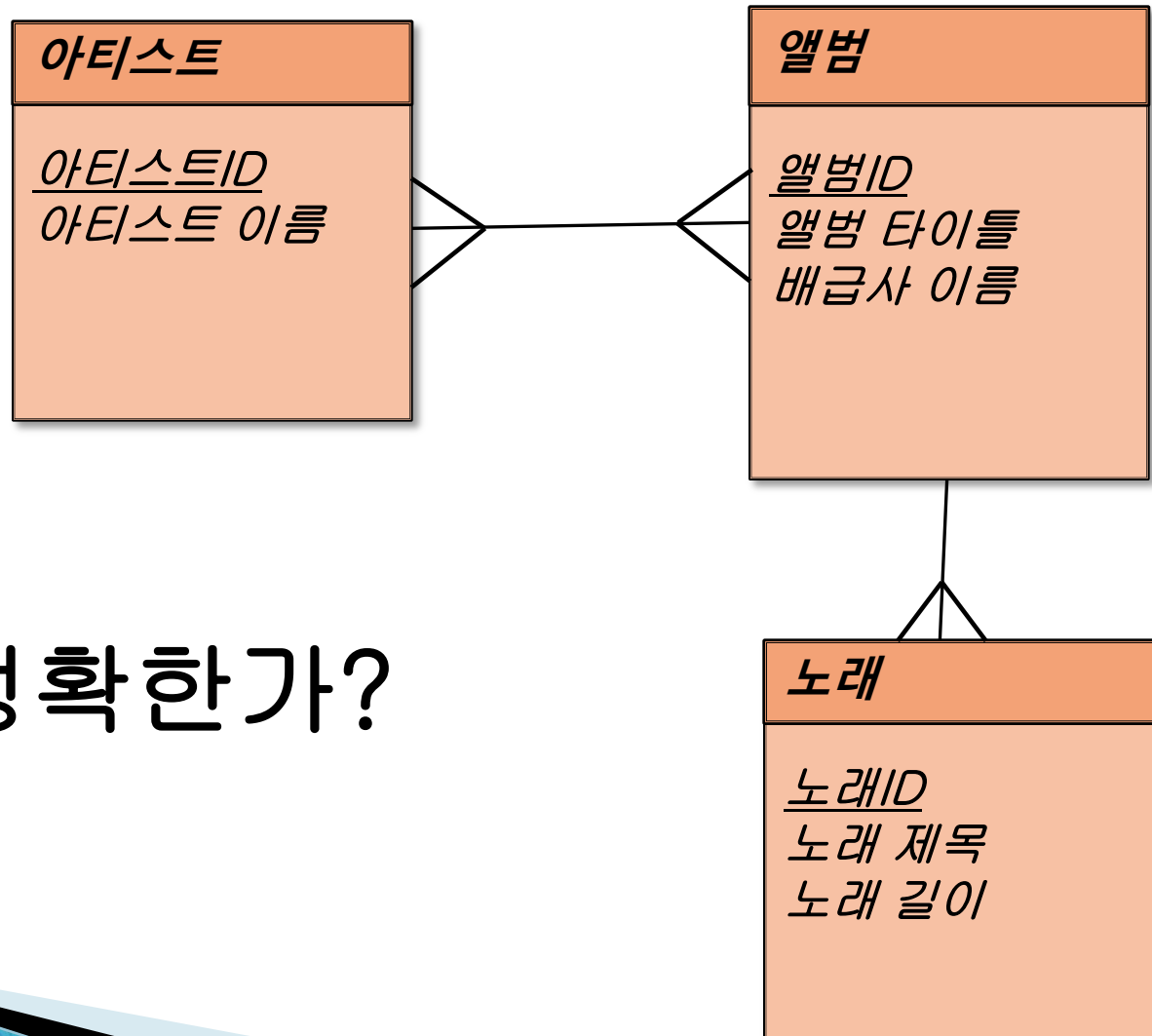
## 7. 정규화 - 제 2 정규형(2NF)

- ▶ 하나의 엔티티가 이미 1NF로 되어 있고 모든 식별되지 않은 속성들이 엔티티의 유일한 식별자에 종속적이면 “제 2 정규형(2nd Normal form)” 이라 한다.
- ▶ 앨범 엔티티의 가수이름은 앨범ID에 완전히 종속적이지 못하다.  
(씨스타는 두 개의 다른 CD의 가수 이름이다)
- ▶ “가수이름은 무엇으로 나타나야 하는가?”

## 8. 다섯 번째 데이터 모델



## 9. 여섯 번째 데이터 모델



정확한가?

# 10. 일곱 번째 데이터 모델



정확한가?

# 11. 관계의 종류

- ▶ 두 엔티티간의 양방향을 결정하는 것은 매우 중요
- ▶ 1:1 (one-to-one, 1-to-1)  
매우 드물다  
음원 검색에서는 안 나타난다.
- ▶ 1:M (one-to-many, 1-to-M)  
한 방향이 1:M, 다른 방향이 1:1 인 경우  
가장 흔하다.

# 11. 관계의 종류 (cont'd)

- ▶ M:M (many-to-many, M-to-M)

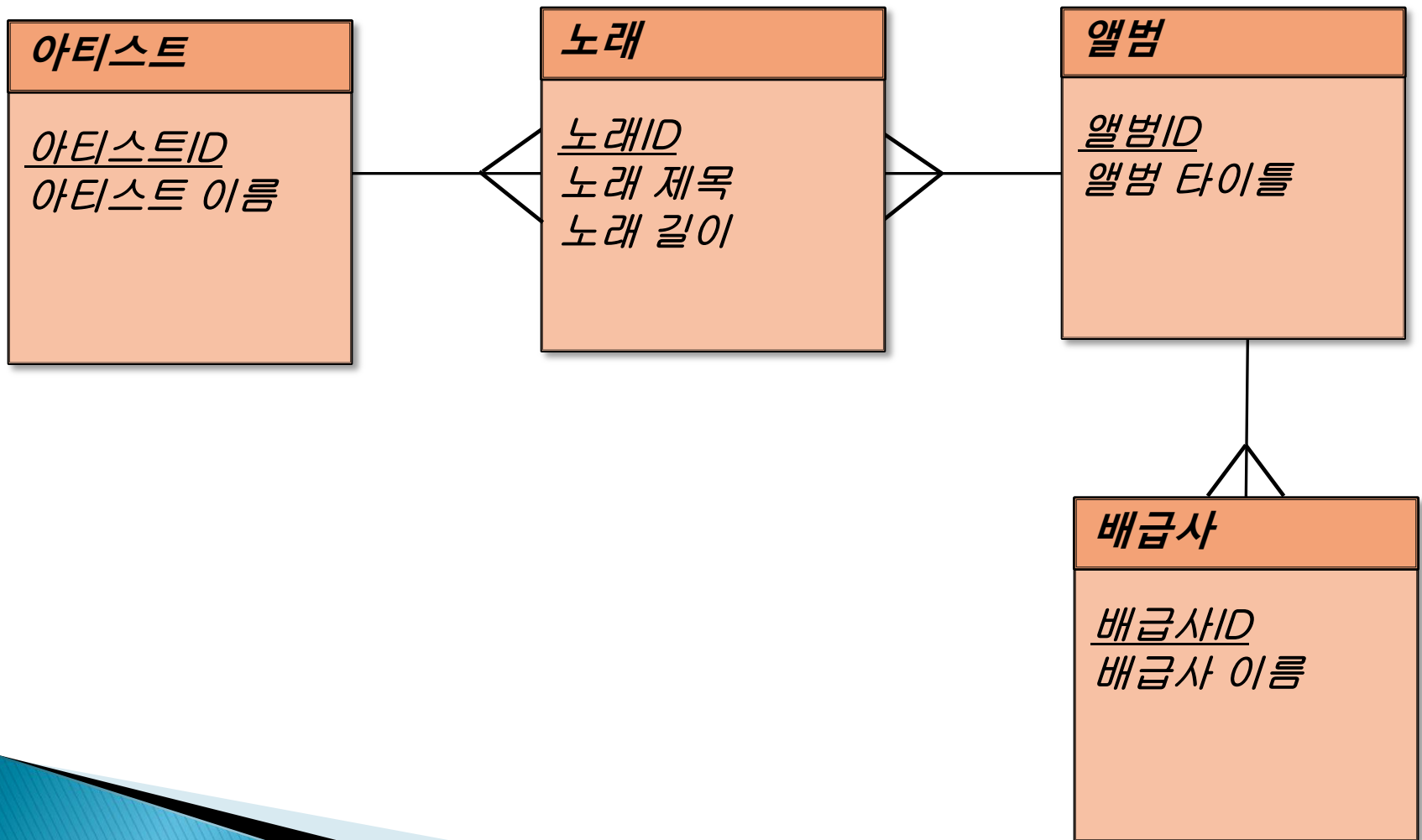
양방향인 1:M인 경우

음원 검색에서는 여섯 번째 데이터 모델

## 12. 관계의 재정립

- ▶ 1:1 (one-to-one, 1-to-1)
  - 설계 자체를 다시 점검해 볼 필요가 있음
  - 두 엔티티가 동일한 엔티티일 가능성을 함축
- ▶ 1:M (one-to-many, 1-to-M)
  - 새로운 엔티티(교차 엔티티)를 생성하고 적당한 이름을 정한다. -> AritistAlbum
  - 새로운 엔티티를 원래의 두 개의 엔티티와 관계를 맺는다 -> 각 각의 원래 엔티티는 교차 엔티티와 1:M의 관계를 맺는다.

# 13. 여덟 번째 데이터 모델(2NF완성)

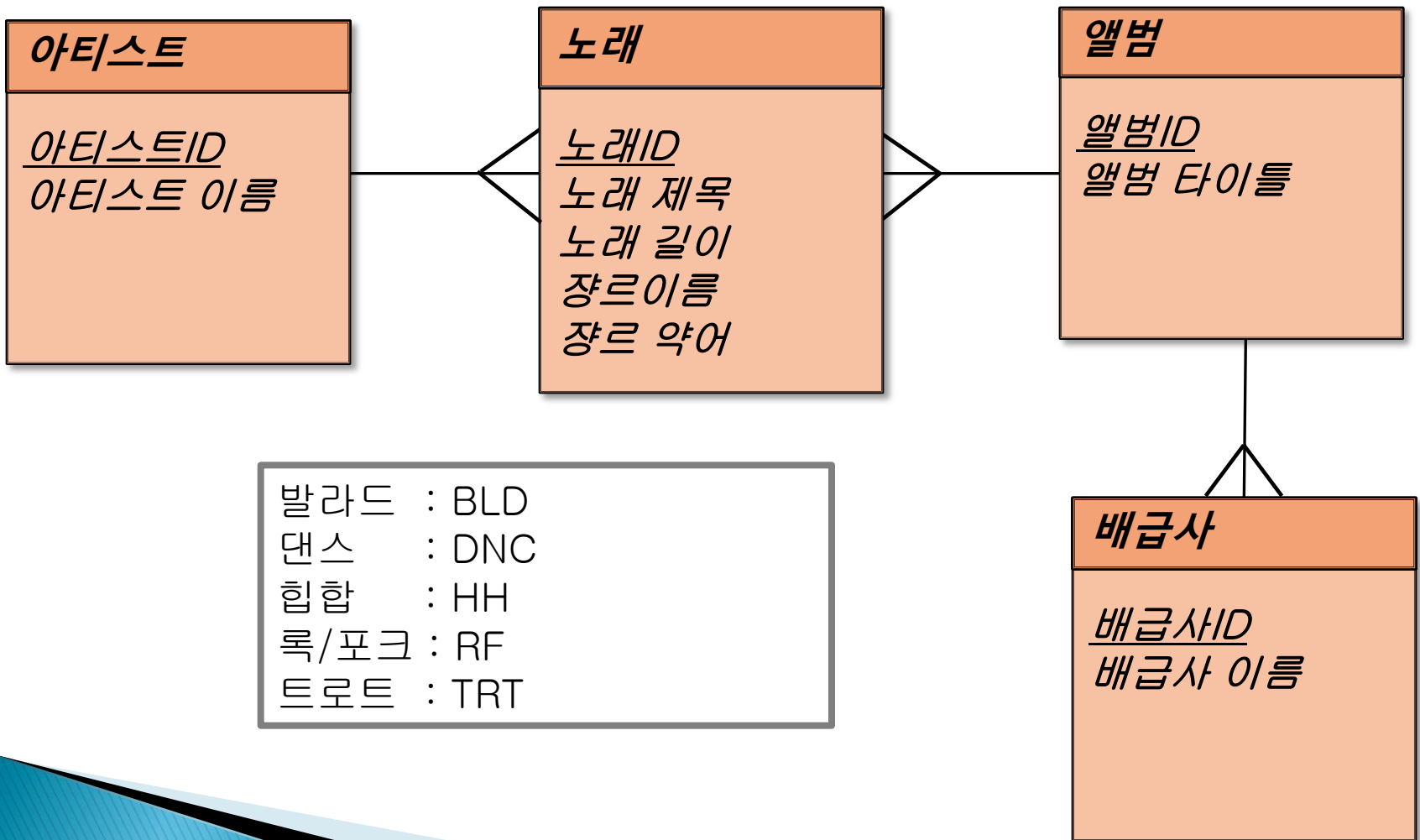




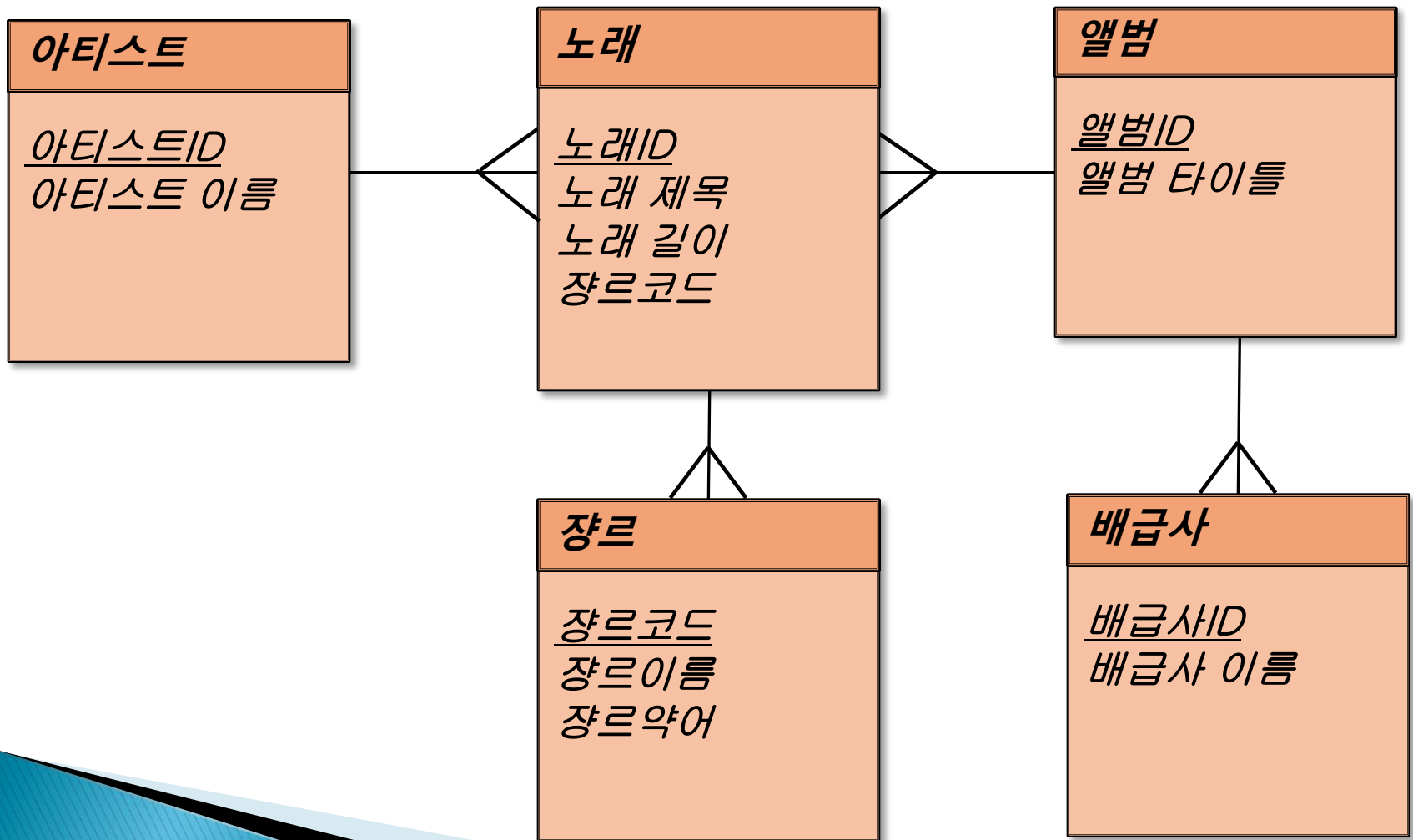
# 14. 정규화 - 제 3 정규형(3NF)

- ▶ 하나의 엔티티가 이미 2NF로 되어 있고 식별할 수 없는 어떠한 속성도 어떤 다른 식별할 수 없는 속성들에게 종속적이지 않으면 “제 3 정규형(3rd Normal form)” 이라 한다.
- ▶ 식별할 수 없는 다른 속성들에게 종속적인 속성들은 종속적인 속성과 새로운 엔티티에 종속적인 속성으로 이동

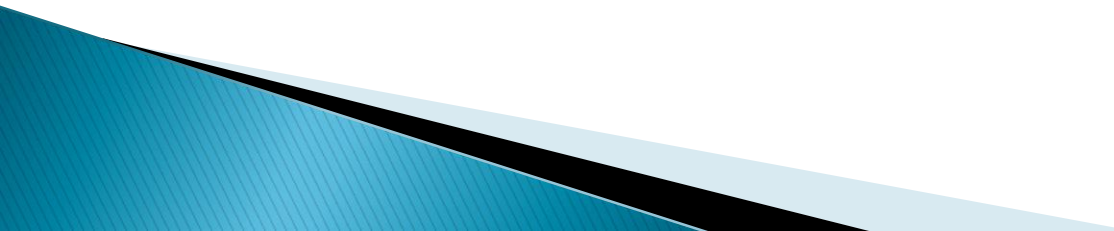
# 15. 3NF 위반 데이터 모델



# 16. 아홉 번째 데이터 모델 (3NF 완성)



# 17. 논리 데이터 모델링 방법론

- ▶ 1. 엔티티를 식별하고 모델화한다.
  - ▶ 2. 엔티티간의 관계를 식별하고 모델화한다.
  - ▶ 3. 속성들을 식별하고 모델화한다.
  - ▶ 4. 각 엔티티의 유일한 식별자를 식별한다.
  - ▶ 5. 정규화 한다.
- 
- ▶ 이 과정은 일률적이지 않고 순서적이지 않다.
  - ▶ 정확한 과정을 따르는 것은 중요하지 않다.
- 

# 18. 물리 데이터베이스 설계

- ▶ 논리적인 데이터 모델링을 하는 이유는 무엇인가?
- ▶ 어떻게 이루어지는가?

postgresSQL, mySQL, Oracle 등의 DBMS가 각각 정의한 SQL 문들의 집합으로 변환

- ▶ 규칙

1. 엔티티는 물리적인 데이터베이스의 테이블
2. 속성은 물리적 데이터베이스의 행(column)
3. 유일한 식별자는 NULL 값이 될 수 없다.

물리적 데이터베이스에서 Primary Key가 된다

4. 관계에서 “일(one)”부분에 있는 Primary Key를 “다(many)”부분의 테이블에 배치함으로 맵핑
5. 매핑된 속성은 Foreign Key가 된다.