Database

Kris

What is Database?

- 체계화된 데이터의 모음
- 검색과 갱신의 효율성을 중시
- 여러 유저가 공유하여 사용할 목적으로 통합/관리한다.

Why Database?

1. 데이터 중복 최소화
2. 데이터 공유
3. 데이터 접근이 쉬움

But…
비용이 높고 전문가가 필요
백업 및 복구가 어려움
시스템의 복잡함 등..

Database Models

- 관계형 모델(Relational Model)
- 계층형 모델(Hierarchical Model)
- 객체-관계형 모델(Object-relational Model)
- 키-값형 모델(Key-Value Store Model)
- 문서형 모델(Document-oriented Model)
- So many more!

Database Models

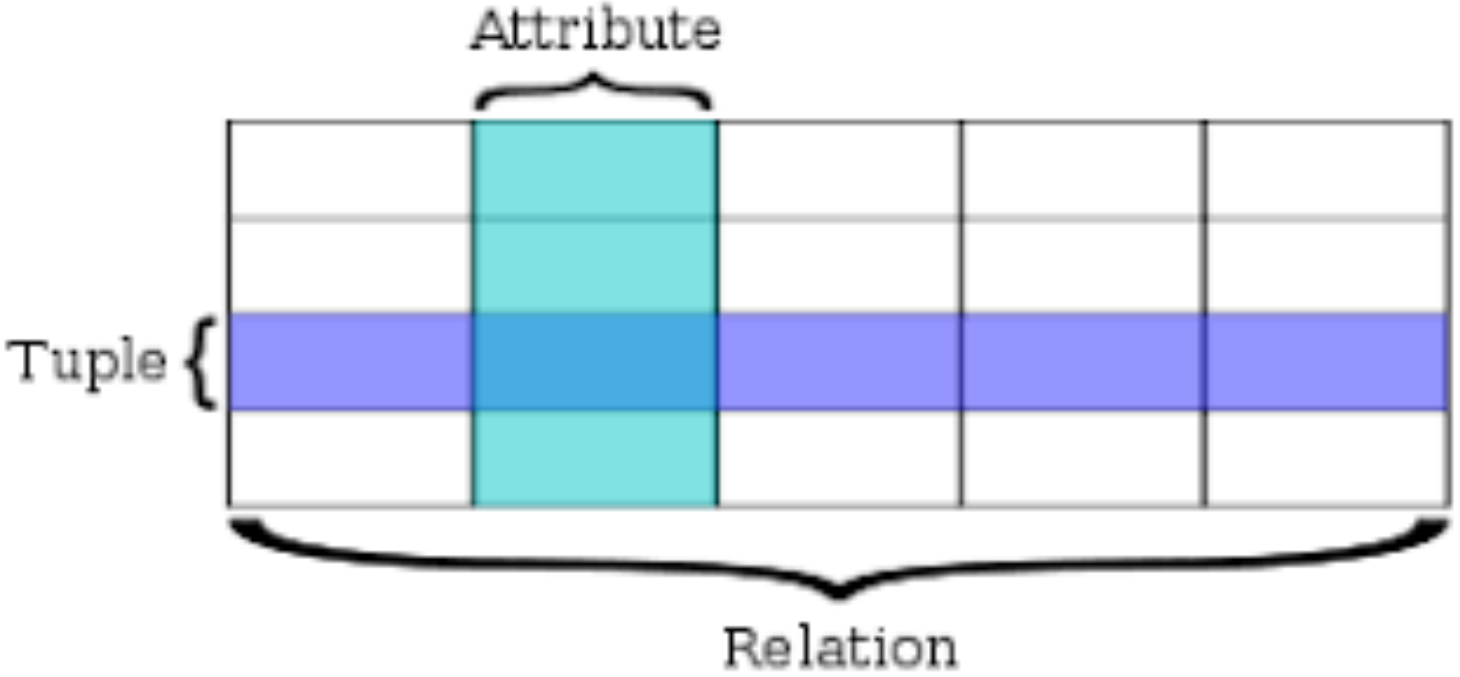
- 관계형 모델(Relational Model)
- 계층형 모델(Hierarchical Model)
- 객체-관계형 모델(Object-relational Model)
- 키-값형 모델(Key-Value Store Model)
- 문서형 모델(Document-oriented Model)
- So many more!

I didn’t want to cover this

- 계층형 데이터베이스
  - 데이터의 관계를 트리 구조로 정의. 즉, 부모 자식 형태의 구조. 단, 일대다 관계이므로 Child가 한 개 이상의 부모를 가지지 못하는 단점이 있다.
  - LDAP은 아직 이 구조를 쓴다고 한다
- 네트워크형 데이터베이스
  - 계층형에서 many to many를 도입하여 유연성을 늘린 모델.
  - 두 모델은 쿼리속도가 아주 빠르나 초기 개발자들이 설계한 대로만 수행이 가능하므로 아주 제한적이다.
  - 데이터를 가져오는 과정에서 아주 느려지거나, 실패하여 데이터베이스 설계 자체를 다시해야 될 수도 있는 것.

관계형 모델(Relational Model)

- 일련의 정형화된 테이블로 구성된 데이터 항목들의 집합.



Key

해당 Data(Tuple)를 고유하게 만들어주는 Attribute(s).
하나 또는 여러개의 Attribute의 조합일 수 있다.

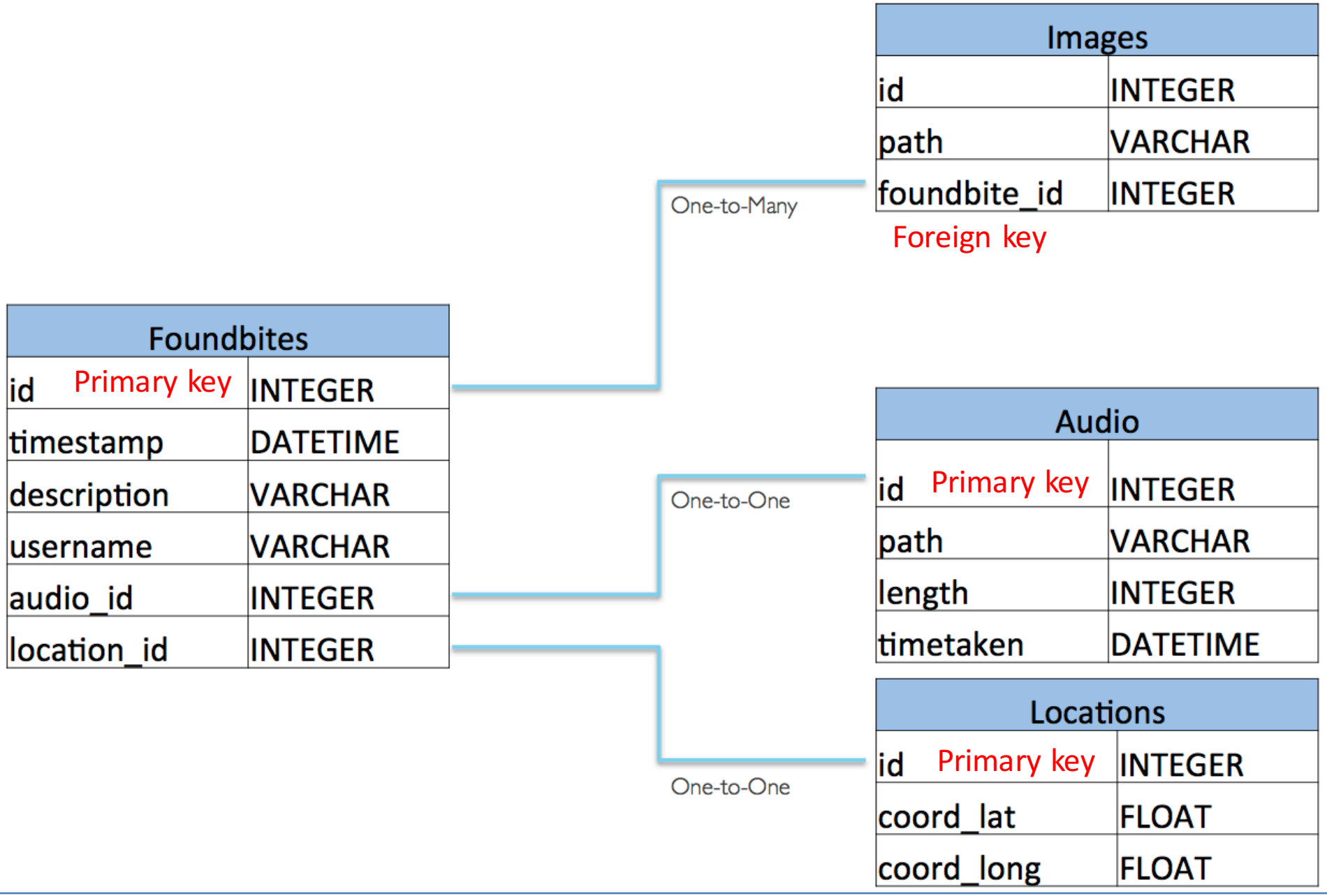
Characteristics of Key

유일성(Uniqueness)
하나의 테이블에서 모든 tuple이 다른 값을 가짐

최소성(Minimality)
꼭 필요한 최소의 속성들로만 구성됨

Different Types of Keys

- Super key
  - 유일성을 만족하는 key
- Candidate key
  - 유일성 및 최소성을 만족하는 key
- Primary Key
  - Candidate key들 중, 기본적으로 사용하는 key
- Alternate Key
  - Primary key를 제외한 나머지 Candidate key
- Foreign Key
  - 다른 table을 참조하기 위한 key
  - 참조할 table의 primary key의 값과 match


Images
id INTEGER
path VARCHAR
foundbite_id INTEGER
Foreign key
One-to-Many
Foundbites
id Primary key INTEGER
timestamp DATETIME
description VARCHAR
username VARCHAR
audio_id INTEGER
location_id INTEGER
One-to-One
Audio
id Primary key INTEGER
path VARCHAR
length INTEGER
timetaken DATETIME
One-to-One
Locations
id Primary key INTEGER
coord_lat FLOAT
coord_long FLOAT

문서형 모델(Document-Oriented Model)

- JSON 유사 형식의 문서로 데이터를 저장 및 쿼리하도록 설계된 비관계형 데이터베이스 유형

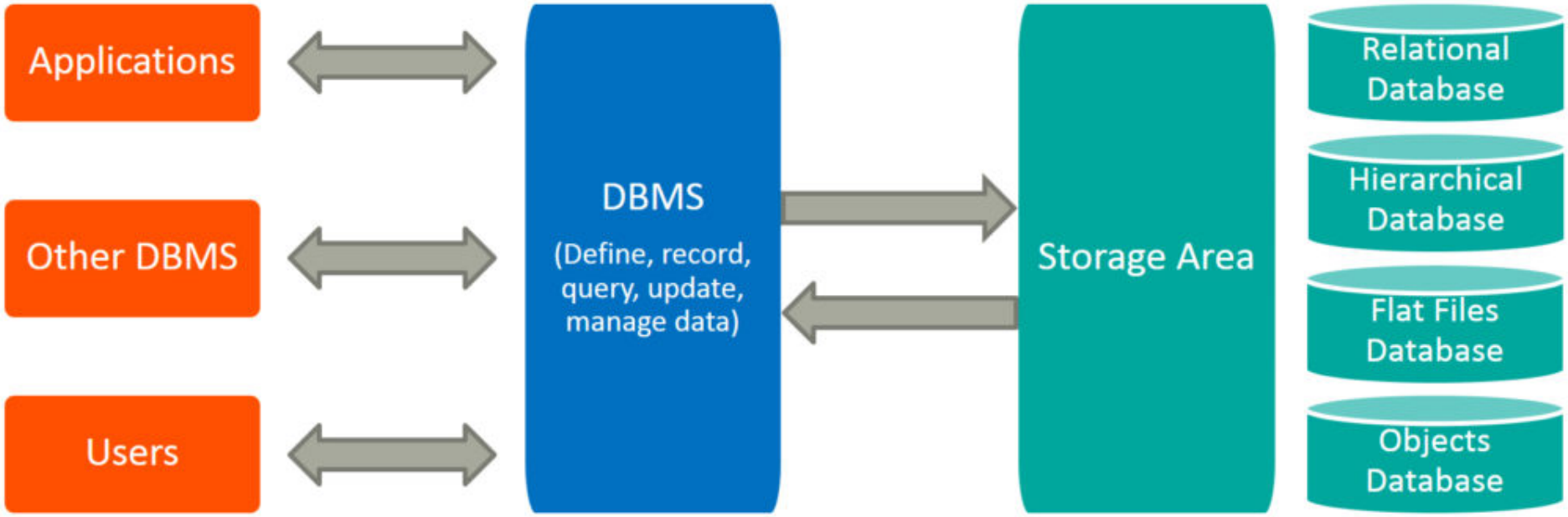
JSON

```json
[
    {
        "year" : 2013,
        "title" : "Turn It Down, Or Else!",
        "info" : {
            "directors" : [ "Alice Smith", "Bob Jones"],
            "release_date" : "2013-01-18T00:00:00Z",
            "rating" : 6.2,
            "genres" : ["Comedy", "Drama"],
            "image_url" : "http://ia.media-imdb.com/images/N/O9ERWAU7FS797AJ7LU8HN09AMUP908RLlo5JF90EWR7LJKQ7@@._V1_SX400_.jpg",
            "plot" : "A rock band plays their music at high volumes, annoying the neighbors.",
            "actors" : ["David Matthewman", "Jonathan G. Neff"]
        }
    },
    {
        "year": 2015,
        "title": "The Big New Movie",
        "info": {
            "plot": "Nothing happens at all.",
            "rating": 0
        }
    }
]
```

DBMS(Database Management System)

- 데이터 자체로만은 별로 쓸모가 없음
- 데이터를 관리하는 시스템이 필요한데..

DBMS(Database Management System)



DBMS Examples

Structured Query Language

- 관계형 DBMS를 관리하기 위해 설계된 프로그래밍 언어

- DBMS 처리 언어의 종류
  - **DDL**(데이터 정의어): 물건을 담을 서랍 자체를 만들고, 바꾸고, 필요 없으면 버릴 수 있는 기능
  - **DML**(데이터 조작어): 서랍에 물건을 넣고, 빼고, 서랍안에 있는 물건이 뭐가 있는지 찾을 수 있는 등의 기능
  - **DCL**(데이터 제어어): 서랍의 물건을 누가 쓸 수 있게 할 것인지 등 사용하는 행위 외 문제에 대해 정의하는 기능

-> SQL은 위의 모든 기능을 수행할 수 있다

MySQL

- Open source RDBMS
- MySQL AB -> SUN -> Oracle
- When Oracle aquired SUN in 2010, MariaDB was created from a forked MySQL project
- Part of LAMP stack(Linux, Apache, MySQL, PHP/Python/Perl)
- Used by many websites
- www.mysql.com

MySQL Tables

- **ISAM**
  - MySQL 3.x버전까지의 테이블
  - 테이블 최대용량은 4GB
  - 5.0 버전부터 사라짐
- **MyISAM**
  - MySQL 4.x 버전의 기본 테이블타입
  - ISAM의 확장
  - 용량은 OS에 달림
  - 작은 규모의 DB에 적합
- **InnoDB** (ara에서 사용)
  - 트랜잭션, Foreign key, row-level, locking 지원
  - 용량 제한이 거의 없음
  - MyISAM에 비해 용량을 많이 사용함
  - 대규모의 DB에 적합

Important Difference

- **MyISAM**
  - Foreign key Constraint, Transaction 지원 X
- **InnoDB** (ara에 사용)
  - Foreign key Constraint, Transaction 지원 O

Then, What is Foreign key Constraint, Transaction?

Foreign Key Constraint

- Constraint: 제약
- Foreign Key: 외부테이블에서 참조한 값
- Foreign Key Constraint
  - 어떤 attribute 값을 외부 테이블에서 가져올 수 있는 제약
  - 못쓴다는 얘기는? 외부에서 값을 가져올 수 없다

Transaction

- Database 처리에 필요한 작업들에 대한 연속처리 단위

- **Atomicity** (원자성)
  - Transaction을 구성하는 모든 명령이 실행되거나 어떠한 명령도 실행되지 않아야 한다.
- **Consistency** (일관성)
  - Transaction이 끝난 후 DB는 일관성을 유지해야 한다.
- **Isolation** (독립성)
  - 모든 Transaction은 동시에 일어나는 다른 Transaction과 상관없이 데이터베이스에 대해 일관된 뷰를 가지고 있다.
- **Durability** (지속성)
  - Transaction이 끝난 후, 데이터베이스는 데이터를 정확히 저장하고 정전이나 그 외의 이상 상황으로부터 데이터를 보호해야 한다.

MySQL Commands

- /etc/init.d mysql [command]
  - (service mysql [command]도 동일하게 작동)
- Commands : start, stop, restart, status

```
root@82e954b6a744:~/Crosswalk# service mysql start
 * Starting MySQL database server mysqld
No directory, logging in with HOME=/
                                                                                [ OK ]
root@82e954b6a744:~/Crosswalk# service mysql status
 * /usr/bin/mysqladmin  Ver 8.42 Distrib 5.7.26, for Linux on x86_64
Copyright (c) 2000, 2019, 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.

Server version          5.7.26-0ubuntu0.16.04.1
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/run/mysqld/mysqld.sock
Uptime:                 7 sec

Threads: 1  Questions: 8  Slow queries: 0  Opens: 105  Flush tables: 1  Open tables: 98  Queries per second avg: 1.14
2
root@82e954b6a744:~/Crosswalk#
```

MySQL Commands

- Mysql [-u/-p/-h/-P/-S] *대소문자에 주의
  - u: user id (-u [user id]꼴)
  - p: user password (-p [password]꼴)
  - (실행할때 그냥 mysql만 쳐도 ID/Password를 추후에 입력 가능)
  - h[host], -P[port], -S[socket address]을 사용하면 원격접속 가능
  - 위의 것들 언급 안하면 localhost로 인식

Basic SQL Queries

- 보기: mysql> **SHOW** DATABASES;
- 생성: mysql> **CREATE** DATABASE [DB name];
- DB 사용: mysql> **USE** [DB name];

selecting data

select "column1“ from "tablename" [where "condition"]; [] = optional

where 중: Like: like 뒤에 쓴 row만 가져온다

inserting data

insert into "tablename" (first_column,...last_column) values (first_value,...last_value);

updating/deleting data

update: UPDATE [table name] SET [field name] = [new value] WHERE …

delete: DELETE FROM [table name] WHERE …

Basic SQL Queries

- 반드시 use [DB name]; 입력 후 사용
- 단순보기: show tables;
- 생성:
  - Create table [table name] ([column1] [datatype] [constraint], [cloumn2] [datatype] [constraint],)
  - Varchar의 경우 괄호안에 크기 지정

- 예시:

```
create table employee(first varchar(15),
last varchar(20), age int(3),
address varchar(30));
```

Here are the most common Data types:

char(size)	Fixed-length character string. Size is specified in parenthesis. Max 255 bytes.
varchar(size)	Variable-length character string. Max size is specified in parenthesis.
number(size)	Number value with a max number of column digits specified in parenthesis.
date	Date value
number(size,d)	Number value with a maximum number of digits of "size" total, with a maximum number of "d" digits to the right of the decimal.

Basic SQL Queries

- select "column1“ from "tablename" [where "condition"]; [] = optional
- Where:

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal to
LIKE	*See note below

- Like: like 뒤에 쓴 row만 가져온다

Basic SQL Queries

- select "column1“ from "tablename" [where "condition"];
  [] = optional

Sample Table: empinfo					
first	**last**	**id**	**age**	**city**	**state**
John	Jones	99980	45	Payson	Arizona
Mary	Jones	99982	25	Payson	Arizona
Eric	Edwards	88232	32	San Diego	California
Mary Ann	Edwards	88233	32	Phoenix	Arizona
Ginger	Howell	98002	42	Cottonwood	Arizona
Sebastian	Smith	92001	23	Gila Bend	Arizona
Gus	Gray	22322	35	Bagdad	Arizona

= Equal
> Greater than
< Less than
>= Greater than or equal
<= Less than or equal
<> Not equal to
LIKE *See note below

```
select * from empinfo
   where first = 'Eric';
```

```
select first, last, city
   from empinfo
   where first LIKE 'Er%';
```

```
select first, last
   from empinfo
   where last LIKE '%s';
```

Basic SQL Queries

Sample Table: empinfo					
first	**last**	**id**	**age**	**city**	**state**
John	Jones	99980	45	Payson	Arizona
Mary	Jones	99982	25	Payson	Arizona
Eric	Edwards	88232	32	San Diego	California
Mary Ann	Edwards	88233	32	Phoenix	Arizona
Ginger	Howell	98002	42	Cottonwood	Arizona
Sebastian	Smith	92001	23	Gila Bend	Arizona
Gus	Gray	22322	35	Bagdad	Arizona
Mary Ann	May	32326	52	Tucson	Arizona
Erica	Williams	32327	60	Show Low	Arizona
Leroy	Brown	32380	22	Pinetop	Arizona
Elroy	Cleaver	32382	22	Globe	Arizona

```
select last, city, age from empinfo
       where age > 30;

select first, last, city, state from empinfo
       where first LIKE 'J%';

select * from empinfo;

select first, last, from empinfo
       where last LIKE '%s';

select first, last, age from empinfo
       where last LIKE '%illia%';
```

= Equal

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

<> Not equal to

LIKE *See note below

Basic SQL Queries

Sample Table: empinfo					
first	last	id	age	city	state
John	Jones	99980	45	Payson	Arizona
Mary	Jones	99982	25	Payson	Arizona
Eric	Edwards	88232	32	San Diego	California
Mary Ann	Edwards	88233	32	Phoenix	Arizona
Ginger	Howell	98002	42	Cottonwood	Arizona
Sebastian	Smith	92001	23	Gila Bend	Arizona
Gus	Gray	22322	35	Bagdad	Arizona
Mary Ann	May	32326	52	Tucson	Arizona
Erica	Williams	32327	60	Show Low	Arizona
Leroy	Brown	32380	22	Pinetop	Arizona
Elroy	Cleaver	32382	22	Globe	Arizona

= Equal

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

<> Not equal to

LIKE *See note below

Select “column1” from “tablename” [where "condition"];

1. Display everyone’s first name and their age for everyone that’s in the table

1. Display the first name, last name, and city for everyone that’s not from Payson

2. Display the first and last names for everyone whose last name ends in an “ay”

Basic SQL Queries

```
insert into "tablename"
  (first_column,...last_column)
   values (first_value,...last_value);
```

```
insert into employee
   (first, last, age, address, city, state)
   values ('Luke', 'Duke', 45, '2130 Boars Nest',
           'Hazard Co', 'Georgia');
```

Basic SQL Queries

- **Update**
  - UPDATE [table name] SET [field name] = [new value] WHERE …
- **Delete**
  - DELETE FROM [table name] WHERE …
- Select 커맨드와 뒷부분은 동일하다

```
update phone_book
  set area_code = 623
  where prefix = 979;

update phone_book
  set last_name = 'Smith', prefix=555, suffix=9292
  where last_name = 'Jones';

delete from employee
  where lastname = 'May';

delete from employee
  where firstname = 'Mike' or firstname = 'Eric';
```

Basic SQL Queries

- 문자는 ' 또는 "로 감싸져 있어야 한다.
  - Ex) 'hello world'

- 숫자는 감쌀 필요가 없다. 그냥 쓴다.
  - Ex) 0x41, 1234, …

- AND, OR을 자유롭게 사용 가능하다.
- #은 주석을 의미한다.
- If, sleep와 같은 작동이 가능하다

Practice Time!

- ssh ubuntu@13.125.175.179
- mysql –uroot –p
- Password: 휠세미나

준비 끝!

Practice Task

1. Check entries of the table and Update if needed
2. Remove all users that is not “Male”
3. Select all users whose student number is 15
4. Delete all users who is born in 96 or later.
5. Check who’s left!

MongoDB

Document 지향 DBMS
BSON

How to Install MongoDB

https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/

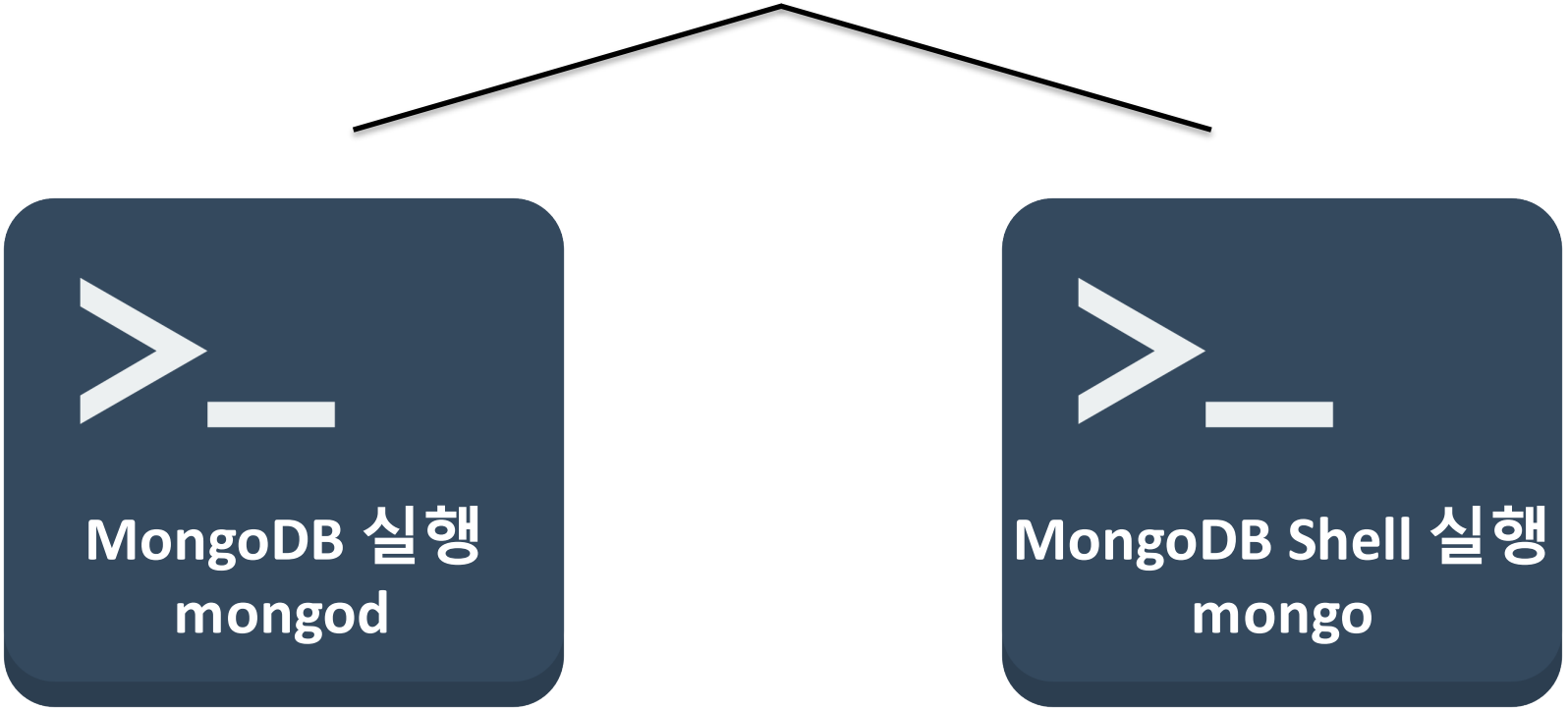
1. sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0C49F3730359A14518585931BC711F9BA15703C6

2. echo "deb [arch=amd64,arm64] https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.0.list

3. sudo apt-get update
4. sudo apt-get install –y mongodb-org
5. sudo mkdir –p /data/db
6. sudo chown [username]:[username] /data/db

Using MongoDB

Use Two Terminals

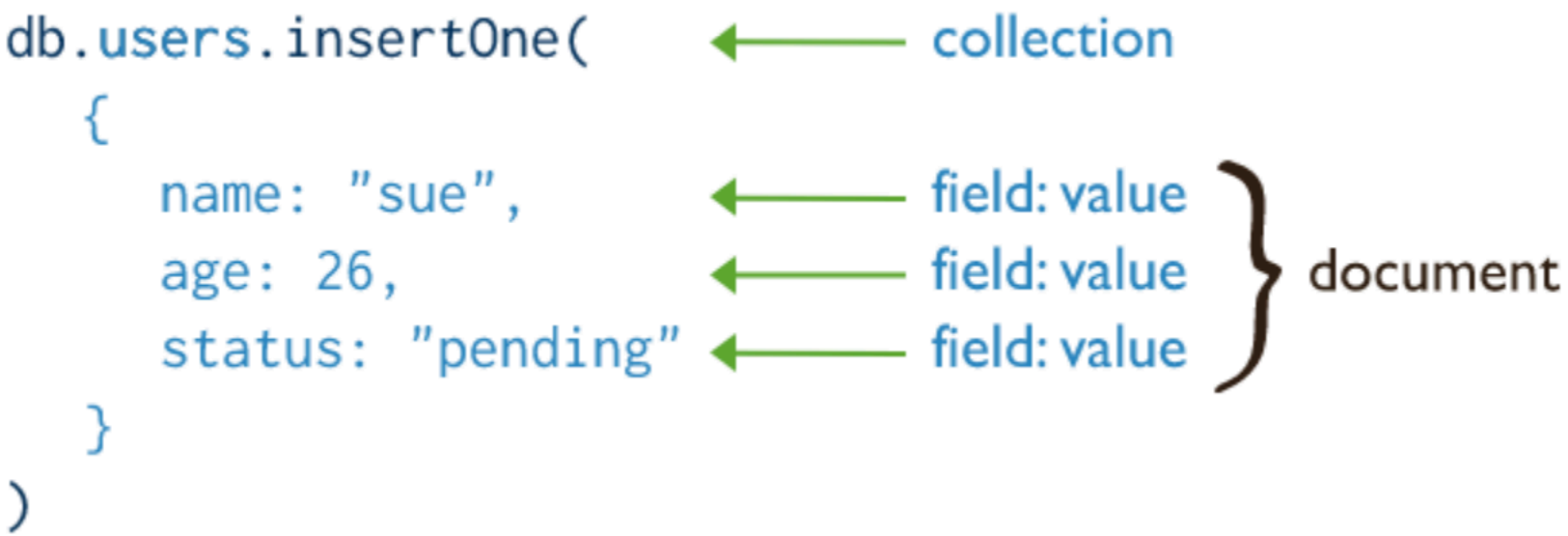


Basic MongoDB Commands

Create

db.{{dbname}}.insertOne(~~)

db. {{dbname}}.insertMany(~~,~~…)



Basic MongoDB Commands

Read

db.{{dbname}}.find(~~)

```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

.limit(n) : n개만 표시
.pretty() : 예쁘게 표시

Basic MongoDB Commands

Update

db.{{dbname}}.updateOne(~~)
db. {{dbname}}.updateMany(~~)
db.{{dbname}}.replaceOne(~~)

Basic MongoDB Commands

Delete

db.{{dbname}}.deleteOne(~~)
db. {{dbname}}.deleteMany(~~)

```
db.users.deleteMany(
   { status: "reject" }
)
```

Other NoSQLs

- NoSQL이란?
  - 고전적인 SQL 기반 관계형 데이터베이스의 대안으로 등장한 데이터베이스들을 아울러 부르는 말
- 높은 트래픽, 많은 계산량, 빠른 레이턴시 등 기존 데이터베이스로 해결하기 어려운 문제들을 해결
- 예시
  - **redis**는 인-메모리 데이터베이스로, 기존에 보조 기억장치에 데이터베이스를 저장하던 것과 달리, RAM 에 데이터를 저장하여 접근 속도, 결과적으로 레이턴시를 개선하였다. 채팅, 푸시 알림 등을 서비스하는데 널리 쓰인다
  - **mongodb**는 관계형 데이터베이스 대신 도큐먼트 모델을 사용하여 스키마가 없다. 따라서 관계형 데이터베이스보다 마이그레이션이 용이하다.
  - **realm**은 모바일용 데이터베이스로 원격 서버에 데이터 조작을 요구하는 (그리고 response를 기다리는) 대신, 로컬 데이터베이스에 데이터를 조작한 후 네트워크 연결 상황이 좋을 때 서버와 동기화하는 방식으로 작동한다. 여러 디바이스에서 동시에 데이터를 조작하더라도 충돌하거나

And more...

Keywords :

SQL join query

Database sharding

Database migration(SQL v.s NoSQL)

IMPORTANT

NoSQL이 RDBMS를 대체하는 것은 불가능!

그러니 적재적소에 잘 쓰면 됩니다☺

Thank you!