Mysql Tutorial

1999

리눅스코리아(주)

제목차례

mysql이라는 클라이언트 프로그램을 이용하여 MySQL을 익혀 보도록 하자. mysql은 간단히 데이터베이스를 만들고 사용할 수 있게 해 주는 프로그램으로 '터미널 모니터' 혹은 간단히 '모니터'라고도 한다.

mysql은 대화식 프로그램으로서 서버에 연결하고, 질문을 수행하고, 결과를 화면에 보여주는 일을 한다. mysql은 배치 모드(batch mode)에서도 사용할 수 있다: 미리 파일에 sql 명령문을 넣어두고 mysql에게 파일의 명령을 수행하라고 하면 된다(뒤에서 알아 보겠지만 'mysql -vvv < batch_test.txt' 식으로 사용하면 된다).

mysql의 옵션들을 보려면 --help 옵션을 붙여서 실행하면 된다:

```
shell> mysql --help
```

이 튜토리얼에서는 mysql에 설치되어 있으며 접근할 수 있는 MySQL 서버가 있다는 것을 가정한다. 그렇지 않으면 MySQL 관리자에게 문의하라(여러분이 관리자라면 MySQL 문서의 다른 부분을 살펴볼 필요가 있을 것이다).

본 튜토리얼에서는 데이터베이스를 설계하고 사용하는 모든 과정을 다룬다. 이미 존재하는 데이터베이스를 사용하는 것에만 관심이 있다면 데이터 베이스와 그 안에 있을 테이블을 만드는 방법을 설명한 절은 건너뛰어도 좋다.

튜토리얼 성격의 글이라 자세한 것은 설명되지 않는다. 여기에 언급된 것에 대해 더 자세히 알고 싶거든 MySQL의 관련 매뉴얼을 보면된다.

1. 서버에 연결하기/연결끊기

서버에 접속하려면 mysql 명령을 내릴 때 MySQL 사용자 이름과 대개의 경우 패스워드를 써 주어야 할 것이다. 서버가 여러분이 로긴한 컴퓨터가 아닌 것에서 운영된다면 호스트 이름도 써 줄 필요가 있을 것이다(호스트 이름, 사용자 이름, 패스워드). 모든 것을 알았다면 다음처럼 연결할 수 있다:

```
shell> mysql -h host -u user -p
Enter password: ********
```

******** 부분은 패스워드다. 'Enter password' 프롬프트가 보이면 패스워드를 쳐주면 된다. 성공하면 간단한 소개 메시지들을 보고 'mysql>' 프롬프트를 볼 수 있을 것이다.

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 459 to server version: 3.22.20a-log

Type 'help' for help.

mysql>
```

'mysql>' 프롬프트가 의미하는 바는 준비되었으니 명령어를 입력하라는 말이다.

어떻게 설치하는 가에 따라 MySQL은 로컬 호스트(local host)에서 운영되는 서버에 "무명의 사용자(anonymous user)"로 접속할 수 있게 한다. 이럴 경우에는 단순히

shell> mysql

처럼 해서 연결할 수 있다.

성공적으로 접속하였다면 'mysql>' 프롬프트에서 언제든지 'QUIT'이라고 쳐서 서버에서 나올 수 있다:

mysql> QUIT
Bye

Ctrl키와 D 키를 동시에 눌러 빠져 나올 수도 있다.

이어지는 절에서 나오는 대부분의 예는 서버에 연결한 상태라는 것을 가정한다. 'mysql>' 프롬프트는 서버에 연결된 상태라는 것을 나타낸다.

2. 질문 하기(Entering Queries)

이전 절에서 언급하였듯이 서버에 접속된 것을 확인하자. 이렇게 한다고 작업할 데이터베이스 어떤 것도 선택하는 것은 아니지만 어쨌든 접속은 해야 한다. 지금 상황에서는 데이터베이스안에 테이블을 만들고, 테이블에 자료를 올리고, 테이블에서 자료를 빼 내는 것보다는 질문하는 법을 약간이라도 배우는 게 더 중요하다. 이번 절에서는 명령어 입력의 기본 원칙을 몇가지 질문 예를 통해 알아 본다. 예를 통해 어떻게 mysql이 동작하는지 익숙해 질 것이다.

아래에 MySQL의 버전과 오늘 날짜를 출력하는 명령어를 보인다. 'mysql>' 프롬프트 다음

에 나오는 대로 쳐 넣자. 그리고 엔터키를 친다.

```
mysql> select version(), current_date;
+------------+-----------------+
| version()  |  current_date   |
+------------+-----------------+
| 3.22.20a-log| 1999-03-19     |
+------------+-----------------+
1 row in set (0.01 sec)
mysql>
```

이 예로부터 mysql에 대한 몇가지 것들을 알 수 있다:

● 명령은 SQL 문과 그 뒤에오는 세미콜론(;)으로 이루어 진다(세미콜론이 필요없는 예외가 있긴하다. QUIT이 그 중 하나다. 나중에 이것에 대해 다시 언급하겠다).

● 여러분이 명령을 내리면, mysql은 서버로 그 명령을 보내어 실행되게 하고, 그 결과를 보여 주고 다시 명령 대기 상태 프롬프트('mysqld>')를 낸다.

● mysql은 테이블 형식(행과 열로 이루어진)으로 결과를 보여준다. 첫 행은 각 열에 대한 라벨을 갖고 있다. 두 번째 행 부터는 질문의 답이 놓인다. 보통, 열의 라벨은 데이터 베이스 테이블에서 가져오는 열의 이름이다. 방금 보인 것처럼 테이블 열이 아닌 표현식(expression)을 사용할 때는 라벨명은 그 표현식이 된다.

● mysql은 행의 수와 명령 실행 시간(대략적인 서버 성능 측정 도구다)을 보여 준다. 명령 실행 시간은 정확한 값은 아니다. 왜냐하면 이 시간은 wall clock time(CPU 시간이 아니다)이라는 것과 서버 부하 및 네트워크 부하에 의한 지연시간에 영향을 받기 때문이다(앞으로 나올 예에서는 지금 설명한 부분은 나타내지 않겠다).

키워드(역자 주: '예약어'라고 합니다)는 대문자로 하던, 소문자로 하던 상관없다. 아래 세 개의 명령은 다 동일하다:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> seLect vErSiOn(), current_DATE;
```

다른 예를 하나 더 살펴 보자. mysql을 간단한 계산기로 사용한 예이다:

```
mysql> select sin(pi()/4), (4+1)*5;
+------------------------+
 | sin(pi()/4) | (4+1)*5) |
```

```
 +-----------+-----------+
 | 0.707107 |        25  |
 +------------+-----------+
```

지금까지의 예에서 명령어는 비교적 짧았고, 한 줄 짜리였다. 한줄에 여러 명령을 기술할 수 있다. 각 명령을 세미콜론으로 끝내기만 하면 된다:

```
mysql> select version(); select now();
+------------+
|  version()   |
+------------+
| 3.22.20a-log|
+------------+

+----------------------+
| now()                |
+----------------------+
| 1999-03-19 00:15:33 |
+----------------------+
```

명령어는 한줄에 모두 다 기술해야만 하는 건 아니다. 긴 명령인 경우 몇 줄에 걸쳐 기술할 수 있다. mysql은 세미콜론을 보고 어디서 명령이 끝나는 지를 분간한다(mysql은 임의의 포맷을 갖는 입력을 받아 들인다: 입력 줄을 모아 세미콜론을 볼 때까지 실행한다).

여러 줄을 걸쳐 명령을 준 예를 보자:

```
mysql> select
    -> user()
    -> ,
    -> current_date;
+--------------------+-------------+
| user()               |  current_date|
+--------------------+-------------+
| joesmith@localhost | 1999-03-18 |
+--------------------+-------------+
```

여러줄 입력할 때 첫줄을 입력하고 엔터키를 쳤을 때 프롬프트가 'mysql>'에서 '->'로 바뀐 것을 주목하라. 이것은 아직 명령이 다 완성되지는 않았으며, 따라서 더 입력을 기다린다라고 mysql이 여러분에게 알리는 것이다. 프롬프트는 여러분의 친절한 안내자다. 귀한 정보를 여러분에게 알려 준다. 프롬프트가 알려주는 것들을 통해 mysql이 무엇을 기다리고 있는지 항상 알 수 있을 것이다.

명령어 입력 도중 취소하려면 \c를 쳐주면 된다:

```
mysql> select
    -> user()
    -> \c
mysql>
```

프롬프트 변화를 잘 보라. \c를 친후 'mysql>'로 바뀌었다. 새 명령어를 받아들일 준비가 되었다는 것을 알리는 것이다.

다음 표는 마주치게 될 프롬프트들과 그 의미를 설명한 것이다.

프롬프트	의미
mysql>	새 명령을 받아 들일 준비가 되었음
->	명령어를 여러 줄에 기술할 때 다음 줄을 기다리고 있음
'>	다음줄 입력을 나타낸다. 현재 '로 시작하는 문자열을 수집하는 중이라는 것을 나타냄 (문자열 입력을 끝내려면 문자열을 다 입력한 후 '를 붙여 줄것)
">	'>와 같다. 단지 차이는 문자열을 '가 아니라 " 로 두른다는 점이다.

세미콜론을 붙이는 것을 잊어버려 우연히 혹은 실수로 여러 줄에 걸치는 명령을 입력할 때가 종종 있다. 이 경우 물론 mysql은 입력을 더 기다린다:

```
mysql> select user()
    ->
```

이럴 때는 mysql은 세미콜론을 기다리고 있는 것이다(여러분은 명령을 제대로 완전히 다 입력했다고 생각하지만 mysql은 그렇지 않다. 세미콜론이 빠졌기 때문이다). 프롬프트가 바뀐 것을 눈치 채지 못한다면 결과를 기다리며 한참 동안의 시간을 낭비할 수도 있다. 세미콜론을 쳐 주어 명령을 완성하면 실행결과를 볼 수 있을 것이다:

```
mysql> select user()
    -> ;
+-------------------+
| user()            |
+-------------------+
| joesmith@localhost |
```

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

'>와 ">는 문자열을 모으는 중에 나타나는 프롬프트이다. MySQL에서는 문자들을 ' 나 " 로 둘러싸면 문자열이 된다(예를 들면 'hello', "goodbye"등이다). 또한 여러 줄에 걸쳐 문자열을 입력할 수도 있다.'> 나 "> 프롬프트가 나타나면 이것은 '나 "로 시작하는 문자열을 포함하는 명령어를 쳐 넣었으나 닫는 '나 "를 아직 쳐 넣지 않았다는 것을 의미하는 것이다. 여러 줄에 걸쳐 문자열을 입력할 때는 상관없다. 하지만 문자열을 여러 줄에 입력하고자 하는 경우가 얼마나 될까? 그다지 많지 않다. 대부분의 경우, '> 나 "> 프롬프트는 닫는 '나 "를 빼먹었다고 알려주는 의미일 것이다. 예를 들면 다음과 같다:

```
mysql> select * from my_table where name = "Smith And age < 30;
    ">
```

위와 같은 select 문을 입력하고 엔터키를 치고 결과를 기다린다해도 아무 결과도 볼 수 없을 것이다. "왜 아무 반응도 없을 것일까?"라고 이상하게 생각하지 말고 "> 프롬프트가 나태내는 의미를 생각해 보자. 문자열을 닫는 인용 부호를 빼먹었다는 것을 알리고 있다. 사실 위의 문장은 잘못이 있다. "Smith 다음에 "를 빼먹은 것이다.

자, 어떻게 해야 할까? 가장 간단한 방법은 명령을 취소하는 것이다. 그러나 간단히 \c를 칠수는 없다. 왜냐하면 \c도 "를 입력하기 전까지는 문자열의 일부로 취급을 받을 것이기 때문이다. 대신 "\c를 입력하면 된다:

```
mysql> select * from my_table where name = "Smith AND age < 30;
    "> "\c
mysql>
```

프롬프트가 mysql>로 되돌려 졌다. 물론 이것은 "새 명령어 실행 준비 완료"의 뜻이다.

'>와 ">가 의미하는 바를 기억하는 것은 중요하다. 잘못하여 닫는 인용 부호을 빼먹었을 때 계속 입력하는 것들은 모두 무시되는 듯하게 보이기 때문이다(여기에는 QUIT도 포함된다). 현재 명령을 취소하기 전에 닫는 인용부호를 꼭 써야한다는 것을 모르면 이것은 매우 혼동스러운 일일 것이다.

3. 데이터 베이스 만들고 사용하기

명령어 입력 방법을 알았으니 데이터 베이스를 만들고 사용해 볼 때가 되었다.

집에서 애완동물을 키운다고 가정해 보자. 애완동물 각각에 대해서 여러 가지 정보를 두고 유지하고 싶을 것이다. 데이터 베이스를 만들고 그 안에 테이블을 만들어서 여기에 원하는 데이터를 넣어두면 된다. 그렇게 하면 테이블에서 자료를 가져와서 애완동물에 대한 여러 가지 정보들을 알아 낼 수 있다. 이 절에서는 이러한 것들을 포함하여 다음과 같은 사항들

을 다루어 본다:

- 데이터 베이스 만들기
- 테이블 만들기
- 테이블에 자료 넣기
- 테이블에서 자료 빼 내기
- 여러개의 테이블 사용하기

데이터 베이스 이름을 menagerie('동물원'이라는 뜻이다)라고 짓자. menagerie 데이터 베이스는 매우 간단하나 실제 생활에서도 간단한 데이터 베이스를 사용하는 경우가 있다. 예를 들면 지금 만들고자 하는 데이터 베이스를 가축을 사육하는 농부나 애완동물의 치료 기록을 남겨두어야 하는 수의사에 의해 사용되어 질 수 있다.

SHOW 문을 사용하여 현재 서버가 유지 중인 데이터 베이스 목록을 볼 수 있다:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

실제 목록은 위와 다를 수 있다. 하지만 mysql, test 데이터베이스는 항상 볼 수 있을 것이다. mysql 데이터베이스는 사용자 접근 권한 정보를 갖고 있는 중요한 데이터베이스이다. test는 말 그대로 연습하기 위해 있는 데이터베이스다. test 데이터베이스가 있다면 다음처럼 해서 사용할 수 있다:

```
mysql> USE test
Database changed
```

QUIT처럼 USE 문은 세미콜론이 필요하지 않다는 것을 기억하자(세미콜론으로 끝내도 상관없다. 그냥 간단하게 모든 문을 세미콜론으로 끝낸다고 기억해 두는 것도 좋다). USE 문은 또한 한 줄에 기술해야 한다는 것도 반드시 기억하자.

test 데이터베이스에 접근할 수 있으면 이것을 사용할 수 있다. 하지만 동일한 데이터베이스(이 경우엔 test)에 접근할 수 있는 사람이 여러 사람이라면 여러분이 만들어 놓은 어떤 자료라도 다른 사람에 의해 접근이 가능하다. 이것은 삭제 및 변경될 소지가 있다는 말이다. 그래서 MySQL 관리자에게 여러분만의 데이터베이스를 사용할 권한을 달라고 요청해야 한다. 여기서는 menagerie라고 하자. 관리자는 다음과 같은 명령문을 실행할 필요가 있

다:

```
mysql> grant all on menagerie.* to your_mysql_name;
```

your_mysql_name은 물론 허락해 줄 MySQL 계정명으로 대치해야 한다.

3.1 데이터베이스 만들고 선택하기

관리자가 접근 권한을 설정할 때 데이터베이스를 만들어 주었다면 그것을 그냥 사용하면 된다. 그렇지 않으면 다음 처럼 하여 손수 여러분이 만들어 주면 된다:

```
mysql> create database menagerie;
```

유닉스에서는 데이터베이스 이름은 대소문자를 구별한다(SQL 키워드는 그렇지 않다). 따라서 데이터베이스 이름을 항상 'menagerie'로 해야지 Menagerie, MENAGERIE, meNaGerIE같은 것은 안된다. 테이블 이름도 마찬가지로 대소문자를 구분한다.

데이터베이스를 만든다고 사용하겠다고 알리는 것은 아니다. 명시적으로 사용하겠다고 해야 한다:

```
mysql> use menagerie
Database changed
```

데이터베이스는 한번만 만들면 되지만 사용할 때마다 use 문을 이용하여 사용할 데이터 베이스를 선택해야 한다. 당연한 논리가 아닐까? 다른 방법으로는 mysql을 시작할 때 데이터베이스 이름을 써 주어도 된다:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

여기서 menagerie가 패스워드는 아니다. 혼동하지 마라. 패스워드를 쓰려면 공백없이 바로 -p 뒤에 붙여 써 주어야 한다(하지만 이 방법은 보안상 바람직한 방법이 절대 아니다. 패스워드가 글자그대로 화면에 보이기 때문이다. 누가 어깨 너머로 보고 있다면 어떻게 할 것인가? 패스워드가 글자그대로 화면에 써진다는 것은 정말 위험하다. 명청한 관리자 및 사용자라면 -p 뒤에 패스워드를 적어 주는 "짓"을 할 것이다. MySQL 개발자들은 왜 이렇게 했을까?). 패스워드가 아니라 사용할 데이터베이스 이름이다.

3.2 테이블 만들기

데이터베이스를 만드는 것은 쉽다. 다음 처럼

```
mysql> show tables;
Empty set (0.00 sec)
```

데이터베이스는 비어 있다. 당연하다. 이제 막 만들었는데 들어있는 것이 있을 리 없다. show tables; 문은 선택된 데이터베이스에 있는 테이블을 보이는 명령이다.

어려운 것은 데이터베이스를 어떻게 설계할 것인가이다. 어떤 테이블이 필요하고 이 안에 무슨 자료들을 넣어야 할지를 생각해야 한다.

여기서의 예에서는 각 애완 동물마다 한 개의 레코드를 두어야 할 것이다. pet 테이블이라고 이름 짓자. 각 테이블에는 애완 동물의 이름, 소유주(식구 이름이 될 것이다), 종, 성(암컷인지 수컷인지) 등등의 정보를 입력하고 싶을 것이다.

나이는? 나이도 필요할 것 같지만 시간에 따라 변하는 것이 나이이므로 나이에 대한 정보를 자주 갱신해 주어야 할 것이다. 보다 나은 방법을 강구해야 한다. 이런 대목을 생각하고 설계하는 것이 데이터베이스 설계시 겪는 어려움은 아닐까 생각한다. 나이는 시간에 따라 변하므로 출생일을 기록해두고 현재 날짜와의 차이로부터 계산하면 좋을 것이다. MySQL은 몇가지 산술 루틴을 제공하므로 이것은 어려운 일이 아니다. 나이대신 출생일을 기록해 두는 것은 다음 두가지 잇점이 잇다:

● 다가오는 애완 동물의 생일을 미리 알려주는 일에 사용할 수 있다(동물에게 생일이라... 비현실적인 것 같지만, 이것은 다른 관점에서 생각해 볼 수 있다. 여러분의 고객의 생일은 어떤가? 언제 생일 축하카드를 보내야 하는 지 알 필요가 있지 않은가?).

● 오늘 날짜 말고 다른 날짜를 기준으로도 나이를 계산할 수 있다. 예를 들어 사망일을 기록해 놓으면 애완 동물의 수명을 알 수 있을 것이다.

애완 동물에 대한 정보로서 다른 것들도 생각할 수 있겠지만 이정도로 해 두자. 충분하다.

create table 문으로 테이블에 둘 자료 구조를 명시할 수 있다:

```
mysql> create table pet (name varchar(20), owner varchar(20),
    -> species varchar(20), sex char(2), birth date, death date);
```

create table 다음에 테이블 이름을 써 주고 괄호 안에 열의 이름과 그 열의 자료형을 한 짝으로 하여 쉼표로 구분하여 열거해 주면 된다. name, owner, species, sex, birth등이 열의 이름이며, varchar(20), char(1), date가 자료형이다. 자료형이란 말 그대로 자료의 형태이다. 자료는 문자열일 수 있고, 날짜 일 수 있고, 수 일수 있다.

다음 표와 같은 테이블이 만들어 진다:

pet table

열	1열	2열	3열	4열	5열	6열
열이름	name	owner	species	sex	birth	death

어떻게 자료를 입력하느냐에 따라 다르겠지만 애완견 "용감이"의 경우 다음처럼 될 수 있다.

name	owner	species	sex	birth	death
용감이	홍길동	진돗개	수컷	1998-3-4	NULL

---

varchar는 길이가 변하는 문자열에 사용한다. 이름, 소유주, 종은 길이가 고정적이지 않은 문자열을 그 자료형으로 할 때 적당할 것이다. varchar 형의 열들에 대해, 길이는 반드시 모두 같은 필요도 없고 20으로 고정될 필요도 없다. 1에서 255사이의 길이를 가질 수 있다. 적당하게 잡아 주면된다(나중에 alter table 문으로 조정할 수도 있다).

테이블을 만들었으니 데이터베이스내 테이블 목록에 추가되었는가 확인하자:

```
mysql> show tables;
+--------------------+
| Tables in menagerie |
+--------------------+
| pet                |
+--------------------+
```

테이블이 명시한대로 만들었는지 확인하기 위해서는 describe 문을 사용한다:

```
mysql> describe pet;
+-------+------------+-----+-----+--------+------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+-----+-----+--------+------+
| name  | varchar(20) | YES |     | NULL   |      |
+-------+------------+-----+-----+--------+------+
| owner | varchar(20) | YES |     | NULL   |      |
+-------+------------+-----+-----+--------+------+
| species | varchar(20) | YES |     | NULL   |      |
+-------+------------+-----+-----+--------+------+
| sex   | char(2)    | YES |     | NULL   |      |
+-------+------------+-----+-----+--------+------+
| birth | date       | YES |     | NULL   |      |
```

```
+--------+------------+-----+-----+-------+-----+
| death  | date       | YES |     | NULL  |     |
+--------+------------+-----+-----+-------+-----+
```

Filed 부분과 Type 부분을 보고 열의 이름과 자료형을 확인하자. describe는 언제든지 사용할 수 있다. 테이블 내의 열의 이름 및 자료형을 잊었을 때 사용하면 유용하다.

3.3 테이블에 자료를 넣어 보자.

테이블을 만든 후에는 테이블에 자료를 넣어야 한다.load data 혹은 insert 문을 사용하면 된다.

애완 동물 자료가 다음과 같다고 가정하자(MySQL은 YYYY-MM-DD 형식의 날짜 포맷을 요구한다).

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	Dianne	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	

여러분은 빈 테이블에서 시작하므로 미리 파일에 각 동물에 대한 자료를 적어 두고 파일에서 읽어서 테이블을 채우면 좋을 것이다.

pet,txt라는 파일(파일이름은 아무것이든 상관없다)에 한 줄에 하나의 레코드를 기록하면 된다. 다음 처럼:

```
# cat pet,txt
Fluffy  Harold  cat     f       1993-02-04
-이후 생략-
```

열의 값들은 탭키 하나로 구분하며 creat table 문에 명시한 순서대로 각 열의 값들을 열거해야 한다. 생략해도 되는 값(위에서 죽은 날짜와 성)에 대해서는 NULL 값을 사용할 수 있다. 텍스트 파일에서 NULL값을 나타내기 위해서는 \N 이라고 써주면 된다. 예를 들어 Whistler의 예는 다음과 같을 것이다.

```
Whistler Gwen bird \N 1997-12-09 \N
```

pet,txt 파일을 로드하기 위해서는 다음처럼 load data 문을 사용한다:

mysql> load data local infile "pet.txt" into table pet;

사용형식은 다음과 같다:

load data local infile "파일이름" into table 테이블이름;

한번에 한 개의 레코드를 추가하고자 할 때가 있을 것이다. create table 문을 사용하였을 때 열거한 순서대로 각 열의 값을 적어 주면 된다. 물론 맞는 자료형으로 말이다:

```
mysql> insert into pet
    -> values('Puffball', 'Diane', 'hamster', 'f', '1999-03-30', NULL);
```

문자열 및 날짜를 작은 따옴표 ' 로 인용하였다. 위에서 언급한 NULL값도 입력할 수 있다 (\N이라고 적으면 안된다).

3.4 테이블로부터 정보를 검색해 보자.

select 문을 사용하면 된다. 일반 형식은 다음과 같다:

select <검색대상> from <테이블> where <검색조건>

<검색대상>은 무엇을 보고 싶은가를 알리는 것이다. 여러 열을 쉼표로 구분하여 적을 수 있으며 모든 열을 의미하는 *를 쓸 수도 있다. where 부분은 생략할 수 있다. where 문을 쓸 때는 검색조건을 써 준다. 검색조건을 만족하지 않는 행은 검색대상에서 제외된다.

3.4.1 모든 데이터를 검색하자.

가장 간단한 select 문의 형태로 다음 처럼 사용할 수 있다:

```
mysql> select * from pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1998-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
```

```
| Slim      | Benny   | snake    | m      | 1996-04-29 | NULL       |
| Puffball  | Diane   | hamster  | f      | 1999-03-30 | NULL       |
+----------+--------+----------+-------+------------+------------+
```

이런 식으로 select문을 사용하는 것은 테이블의 전체 정보를 보고자 할 때 유용하다. 방금 막 초기 데이터 뭉치를 올렸을 때 제대로 올려졌는지 확인코자 사용할 수 있다. 사람 사는 일이 그렇듯, 방금 본 결과에는 잘못된 것이 있다: Bower의 출생일자가 사망일자보다 늦다. 죽은 뒤에 태어났다?! 확인해 보니 birth는 1989-08-31이 되어야 함을 알수 있었다고 하면 이를 어떻게 고칠까?

두가지 방법을 사용할 수 있다:

● 파일 pet.txt를 편집하여 수정한다. 테이블을 비운후 pet.txt에서 다시 읽어 들인다:

```
mysql> delete from pet;
mysql> load data local infile "pet.txt" into table pet;
```

하지만 이렇게 하면 3.3절에서 개별적으로 insert문을 이용하여 입력한 Puffball에 대해서 다시 입력해야 한다. 더 간단하고 바람직한 방법은?

● 잘못된 곳만 수정한다. update 문을 사용한다:

```
mysql> update pet set birth="1989-08-31" where name="Bowser";
```

위에서 볼 수 있듯이, 전체 테이블 내용을 보는 것은 쉽다. 그러나 보통 이렇게 하지는 않는다. 테이블 크기가 커지면 어떻게 할 것인가? 어떤 자료들을 검색할 때 그 많은 것을 일일이 다 볼것인가? 대신 특별한 조건을 만족하는 자료들만 뽑아서 보길 원할 것이다.

3.4.2 주어진 조건에 맞는 특정 행만을 검색해 보자.

여러분은 테이블에서 특별한 행들만 뽑아낼 수 있다. 예를 들어 Bower의 생일이 정말로 바뀌었는 가 확인하기 위해 Bower의 레코드만 뽑아낼 수 있다:

```
mysql> select * from pet where name = "Bower";
+--------+-------+---------+------+-----------+------------+
| name   | owner | species | sex  | birth     | death      |
+--------+-------+---------+------+-----------+------------+
| Bowser | Diane | dog     | m    |1989-08-31 |1995-07-29 |
+--------+-------+---------+------+-----------+------------+
```

birth 열의 값이 1998년도가 아닌 1989년으로 올바르게 수정됨을 확인할 수 있다.

문자열 비교는 대소문자를 무시하는 비교다. 따라서 "bowser", "BOWSER" 등은 같은 문자열을 의미한다(위에서는 "Bowser"를 사용했다).

어떤 열에대해서도 조건을 명시해 줄수가 있다. 예를 들어 1998년 이후에 태어난 동물을 알고 싶다면 birth 열을 대상으로 검사하면 된다:

```
mysql> select * from pet where birth >= "1998-1-1"
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirp y  | Gwen  | bird    |  f   |1998-09-11 | NULL |
| Puffball | Diane | hamster |  f   |1999-03-30 | NULL |
+----------+-------+---------+------+------------+------+
```

조건을 조합할 수 도 있다:

```
mysql> select * from pet where species = "dog" AND sex = "f";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    |1989-05-13 | NULL |
+-------+--------+---------+------+------------+-------+
```

위의 예는 개이면서 숫컷인 동물을 검색하는 것이다.

위에서는 AND를 사용하였지만 OR를 사용할 수도 있다:

```
mysql> select * from pet where species = "snake" or species = "bird";
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    |1998-09-11 | NULL  |
| Whistler | Gwen  | bird    |NULL |1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    |1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

AND와 OR를 섞어서 사용할 수 있다. 이렇게 할 때는 그룹지어지는 조건들을 괄호로 묶는 것이 좋다:

```
mysql> select * from pet where (species = "cat" AND sex = "m")
    -> OR (species = "dog" AND sex = "f");
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

3.4.3 특정한 열 선택하기

테이블에서 한행 전체를 보기 보다는 "관심 거리" 열들만 보고 싶다면 보고자 하는 열 이름을 다음처럼 사용하면 된다(아래 예는 name, birth열을 보고 싶은 경우이다):

```
mysql> select name, birth from pet;
+----------+------------+
| name     |   birth    |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

소유주만 보고자 할 때:

```
mysql> select owner from pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
```

```
| Benny  |
| Diane  |
+--------+
```

하지만 중복된 행이 있다. 중복된 행을 없애 보자:

```
mysql> select distinct owner from pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

distinct 키워드를 사용하면 된다.

where 절을 이용하여 행의 선택과 열의 선택을 조합할 수 있다. 예를 들어 개와 고양이에 대해서만 이름, 종, 생일을 알고자 할 때 다음 처럼 select문을 사용할 수 있다:

```
mysql> SELECT name, species, birth FROM pet
    -> WHERE species = "dog" OR species = "cat";
+--------+---------+------------+
| name   | species |   birth    |
+--------+---------+------------+
| Fluffy | cat     |1993-02-04 |
| Claws  | cat     |1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     |1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

3.4.4 행 정열하기

지금까지 보인 예에서 결과가 전혀 정렬되어 있지 않았음을 알 수 있을 것이다. 결과가 정렬된다면 원하는 자료를 훨씬 쉽게 알아 볼 수 있다. 정렬하기 위해서는 order by 절을 사용하면 된다. 아래는 생일을 기준으로 정렬한 것이다.

```
mysql> select name, birth from pet order by birth;
+----------+------------+
```

```
| name     |  birth      |
+----------+------------+
| Buffy    | 1989-05-13|
| Bowser   | 1989-08-31|
| Fang     |1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

역순으로 정렬하려면 desc 키워드를 열 이름 뒤에 적어준다:

```
mysql> select name, birth from pet order BY birth desc;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

하나의 열에 대해서만이 아니라 여러 열에 대해서도 정렬할 수 있다. 예를 들어 먼저 동물 이름으로 정렬하고 각 동물에 대해 생일로 정렬하되 가장 어린 것(가장 늦게 태어난 것; 역순 정렬)이 먼저 오게 하려면 다음처럼 한다:

```
mysql> SELECT name, species, birth FROM pet ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    |1998-09-11 |
| Whistler | bird    |1997-12-09 |
| Claws    | cat     |1994-03-17 |
```

```
| Fluffy    | cat      |1993-02-04 |
| Fang      | dog      |1990-08-27 |
| Bowser    | dog      |1989-08-31 |
| Buffy     | dog      |1989-05-13 |
| Puffball  | hamster  |1999-03-30 |
| Slim      | snake    |1996-04-29 |
+----------+---------+------------+
```

DESC 키워드는 바로 그 앞에 열 이름(birth)에만 적용된다는 것을 주의하라. species는 여전히 오름차순으로 정렬된다.

3.4.5 날짜 계산

MySQL은 날짜를 다루는 몇가지 함수를 제공해 준다.

애완 동물의 나이가 얼마나 되는 지 계산하려면 오늘 날짜와 출생일을 구하고, 두 날짜를 일수로 환산한 후, 그 차를 연간 일수 즉 365일로 나누어 주면 될 것이다:

```
mysql> select name, (to_days(now())-to_days(birth))/365 from pet;
+----------+-----------------------------------------+
| name     | (TO_DAYS(NOW())-TO_DAYS(birth))/365 |
+----------+-----------------------------------------+
| Fluffy   |                                    6.15 |
| Claws    |                                    5.04 |
| Buffy    |                                    9.88 |
| Fang     |                                    8.59 |
| Bowser   |                                    9.58 |
| Chirpy   |                                    0.55 |
| Whistler |                                    1.30 |
| Slim     |                                    2.92 |
| Puffball |                                    0.00 |
+----------+-----------------------------------------+
```

여기서 두가지 사항을 개선해 보자. 결과가 이름 혹은 나이 순으로 정렬되었으면 좋겠고, 나이에 해당하는 라벨명을 표현식 그대로 쓰는 것 보다는 "age"같은 것으로 하는 것이 좋을 것이다:

```
mysql> select name, (to_days(now())-to_days(birth))/365 as age
    -> from pet order by name;
```

나이순으로 정렬하려면 order by name 대신 order by age로 써 주면 된다.

사망시 나이도 비슷한 방법으로 알아 낼 수 있다:

```
mysql> select name, (to_days(death)-to_days(birth))/365 as age
    -> from pet where death is not null order by age;
```

now()대신 death를 사용하면 된다. 여기서 아직 죽지 않은 동물의 수명을 조사한다는 것은 무의미하기 때문에 death 필드가 null이 아닌 경우를 조건으로 해 주었음을 주의하자. 조심할 것은 death is not null처럼 조건을 주어야 한다. death != null 처럼 주어서는 안된다. null값에 비교 연산자를 적용할 수 없다. 나중에 이 문제는 다시 다룰 것이다.

다음 달에 생일인 동물을 알려면 어떻게 해야 할까? 이러한 문제를 위해 MySQL은 날짜에서 연도나, 달을 계산하는 함수를 제공한다: year(), month, day(),month() 예를 통해 알아 보자:

```
mysql> select name, birth, month(birth) from pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| Fluffy   | 1993-02-04 |            2 |
| Claws    | 1994-03-17 |            3 |
| Buffy    | 1989-05-13 |            5 |
| Fang     | 1990-08-27 |            8 |
| Bowser   | 1989-08-31 |            8 |
| Chirpy   | 1998-09-11 |            9 |
| Whistler | 1997-12-09 |           12 |
| Slim     | 1996-04-29 |            4 |
| Puffball | 1999-03-30 |            3 |
+----------+------------+--------------+
```

month는 달에 해당하는 수를 반환해 주며, 물론 그 범위는 1에서 12까지이다. 다음달을 나태내기 위해서는 1을 더한 달을 명시해주면 된다:

```
mysql> select name, birth from pet where month(birth) = 10;
```

그런데 문제가 있다. 12월인 경우 13을 명시해 주어야 하나? 13월이란 없다. 현재 달이 몇월이던지 상관없도록 새로운 조건식을 생각해 내야 한다. 여기에 두가지를 소개한다:

● month(date_add(now(), interval 1 month));
now()는 현재 날짜 및 시간을 반환해 준다. 여기에 1달이라는 기간을 더 해주고 달로 바꾸면 해결된다.

● mod(month(now()), 12) + 1;

mod는 어떤 수를 다른 수로 나눈 나머지 값을 반환하는 함수이다. 첫 번째 인자를 두 번째 인자로 나눈 결과를 반환한다. 여기서는 현재 달 month(now())를 12로 나눈 뒤 다음 달을 나타내기 위해 1을 더해 준다. 이번 달이 12월이라면 12로 나눈 나머지가 0이므로 여기에 1을 더해 다음달 1월을 나태내 줄 수 있다.

완전한 SQL문은 각각 다음과 같다:

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MONTH(DATE_ADD(NOW(), INTERVAL 1
MONTH));

mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(NOW()),12) + 1;
```

3.4.6 null 값에 대해

NULL값은 특별한 값이다. 익숙해 질 때 까지 혼동될 것이다. 개념적으로 NULL이 의미하는 바는 "빠진, 빼먹은 값", "아직 정해지지 않은 불확정 값"을 의미한다. 이것은 다른 값들과는 다르게 취급된다. NULL에는 산술 비교 연산을 수행할 수 없다. 어떤 값과 NULL값을 =, <, !=을 이용하여 비교하는 것은 의미가 없다. 불확정 값을 어떻게 확정된 값과 비교할 수 있을 것인가? 다음을 보라:

```
mysql> SELECT 1 = NULL, 1 != NULL, 1 < NULL, 1 > NULL;
+----------+----------+----------+----------+
|1 = NULL |1 != NULL |1 < NULL| 1 > NULL |
+----------+----------+----------+----------+
|    NULL |    NULL |    NULL |   NULL |
+----------+----------+----------+----------+
```

위에서 보듯 불확정 값과 확정값과의 비교는 불확정값이 된다. 의미가 없다. 다음과 같이 하면 의미가 있다:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL| 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

MySQL에서 거짓 값은 0으로 참값은 1로 나타낸다.

3.4.7 패턴 일치

패턴 일치 기능은 매우 유용한 기능이다. 보다 빠르고 정교하게 원하는 조건을 명세하여 검색할 수 있게 해주는 기능이기 때문이다.

MySQL은 표준 SQL 패턴 뿐만아니라 유닉스에서 사용하는 정규 표현식에 해당하는 패턴 일치 기능도 지원한다.

SQL에서 _ 은 임의 한 문자를 의미하며, %는 임의의 수의 문자(0개의 문자를 포함)를 가르킨다. SQL 패턴은 대소문자를 비교하지 않는다. LIKE 이후에 패턴을 준다는 것을 기억하자. 아래 예를 보라:

b로 시작하는 이름에 대해서 검색할 때:

```
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 |
+--------+--------+---------+------+------------+------------+
```

fy로 끝나는 이름에 대해서 검색할 때:

```
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  |
+--------+--------+---------+------+------------+-------+
```

w를 포함하는 이름을 검색할 때:

```
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       |
```

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

정확하게 5개의 글자로 이루어진 이름에 대해서 검색할 때는?

```
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  |
+-------+--------+---------+------+-----------+-------+
```

밑줄을 5개 적어 준다.

이젠 정규표현에 기반한 패턴일치에 대해 알아 보자.

정규표현식에 사용되는 문자	설명
.	문자 하나
*	앞에 나온 문자의 0개 이상의 반복
^	문자열 처음
$	문자열 끝
[,]	괄호안의 문자들에 일치
{,}	반복을 나타낼 때, 예로 n번 반복할 때 {n}으로 적는다.

여기서 SQL 패턴은 전체 값과 일치해야 "일치한다"고 하지만 정규표현은 값의 어느 부분과 일치해도 "일치한다"고 단정한다는 것을 유의해야 한다. 예를 들어,

```
select * from pet where name regexp "ffy";
```

와

```
select * from pet where name like "ffy";
```

는 전혀 다른 결과를 낳는다.

문자 a나 b나 c중 하나를 가르키는 표현은 [abc]이다. 범위를 주어서 표현할 수도 있다. 정규표현은 대소문자를 구별한다. 따라서 대문자던 소문자던 상관없이 알파벳 문자 하나를 가르키는 표현은 [a-zA-Z]로 해야 한다.

는 0개 이상의 문자들이라고 했다. x 는 x, xx, xxx ... 에 해당한다. [0-9]*는 7, 12, 345, 678등의 임의의 길이를 갖는 수를 나타낸다. ^abc는 줄 처음에 abc로 시작하는 패턴

을 abc$는 abc로 끝나는 문자열을 의미한다.

정규 표현식을 쓸 때는 LIKE대신 REGEXP을 사용한다.

예를 보며 익혀 보자.

이름이 소문자 b 혹은 대문자 B로 시작하는 조건으로 검색:

```
mysql> SELECT * FROM pet WHERE name REGEXP "^[bB]";
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

이름이 fy로 끝날 때($를 사용한다):

```
mysql> SELECT * FROM pet WHERE name REGEXP "fy$";
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    |1993-02-04 | NULL |
| Buffy  | Harold | dog     | f    |1989-05-13 | NULL |
+--------+--------+---------+------+------------+-------+
```

정확하게 5개의 문자로 이름어진 값은 다음 정규 표현에 일치한다:

```
^.....$
```

이것은 반복 연산자를 이용하여 다음처럼 쓸 수도 있다.

```
^.{5}$
```

3.4.8 행수 세기

누가 어떤 애완 동물을 몇이나 소유했는지 어떻게 알아낼 수 있을까?

이에 대한 답으로 count()함수를 사용하면 되며 적당하게 조건을 부여해 주면 된다.

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

```
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

각 소유주가 소유한 애완동물의 수는 다음 처럼 하면 확인 할 수 있다:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

각 owner의 모든 레코드들을 한데 묶기 위해 GROUP BY 절을 사용한 것을 주목하라. 이렇게 하지 않으면 에러 메시지를 보게 될 것이다.

```
mysql> SELECT owner, COUNT(owner) FROM pet;
ERROR 1140 at line 1: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

COUNT()와 GROUP BY는 데이터에 여러 모양으로 특성을 부여하는 데 쓸모가 있다. 다음 예제들도 참고하자:

각 종에 해당하는 동물의 수:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

성에 따른 동물의 수:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

NULL은 "값을 모름"의 의미이다.

종과 성에 따른 동물의 수:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

바로 위의 경우와는 달리, 특정한 동물에 대해서만 조사해 볼 수 도 있다. 개와 고양이의 경우에만 각 성에 대해 몇마리인지 조사해 보자:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE species = "dog" OR species = "cat"
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
```

```
| dog      | m    |         2 |
+---------+------+----------+
```

3.5 테이블 여러개 사용하기

pet 테이블은 애완동물에 대한 정보를 갖고 있다. 수의사에 치료 받으러 갔던 횟수나 새끼를 나은 날짜 같은 사건들에 대한 다른 정보를 기록하고싶다면 별도의 테이블이 필요할 것이다. 테이블은 다음과 같은 조건을 요구할 것이다:

● 해당 동물의 이름을 갖고 있어야 한다. 어떤 애완 동물에게 일어난 사건인지 분별해야 하기 때문이다.
● 언제 일어난 일인지 알기 위해 날짜 정보가 필요하다.
● 어떤 사건인지 묘사해 둘 필요가 있다.
● 사건을 카테고리화하려면 사건 유형을 나타내는 필드도 있으면 좋을 것이다.

이와 같은 조건을 생각하여, 다음 처럼 테이블을 만들어 보자:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

pet 테이블의 경우 처럼 파일로부터 데이터를 테이블로 올리자. event.txt에 다음처럼 적혀 있다고 하자.

```
Fluffy  1995-05-15  litter  4 kittens, 3 female, 1 male
Buffy  1993-06-23  litter  5 puppies, 2 female, 3 male
Buffy  1994-06-19  litter  3 puppies, 3 female
Chirpy  1999-03-21  vet  needed beak straightened
Slim  1997-08-03  vet  broken rib
Bowser  1991-10-12  kennel
Fang  1991-10-12  kennel
Fang  1998-08-28  birthday  Gave him a new chew toy
Claws  1998-03-17  birthday  Gave him a new flea collar
Whistler  1998-12-09  birthday  First birthday
```

다음 처럼 테이블을 채우자:

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

지금까지 pet 테이블을 다루면서 배웠듯이 event 테이블에 대해 여러 가지 질의를 해 볼 수 있을 것이다. 하지만 정보가 불충분할 때는 어떻게 하는가?

새끼를 낳았을 때 어미의 나이를 알려면 어떻게 해야 하는가? event 테이블을 통해 언제 새끼를 낳았는지는 알 수 있지만 어미의 나이라든가, 소유주라든가 하는 것은 pet 테이블을 통해서 알아 내야 한다. 따라서 select 문을 사용할 때 두 개의 테이블이 필요하다:

```
mysql> SELECT pet.name, (TO_DAYS(date) - TO_DAYS(birth))/365 AS age,
    -> remark FROM pet, event
    -> WHERE pet.name = event.name AND type = "litter";
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy | 2.27 | 4 kittens, 3 female, 1 male |
| Buffy  | 4.12 | 5 puppies, 2 female, 3 male |
| Buffy  | 5.10 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

위의 예로부터 몇가지 알아 두어야 할 사항이 있다:

● FROM 절에 사용할 테이블을 모두 적어 주어야 한다. 이것들 모두로부터의 정보가 필요하기 때문이다.
● 여러 테이블에서 정보를 뽑아 합할 때는 한 테이블의 레코드가 다른 테이블의 레코드와 어떻게 일치하는 지 명시해 주어야 한다. 여기서는 두 테이블 모두 name 필드를 갖고 있으므로 이것을 이용하면 된다. 위에서 where 절에 pet.name = event.name 조건을 줌으로써 두 개의 테이블의 같은 동물에 해당하는 레코드에 대해서 질의를 하게 된다. 서로 다르다면 의미가 없다.
● 두 테이블 모두 name 필드를 갖고 있으므로 어느 테이블에 속하는 필드인지를 구분하기 위해 <테이블이름>.<필드이름>의 형식으로 적어 주었다. 즉 테이블 이름과 필드 이름을 점으로 구분하여 적어 준다.

위에서 테이블은 서로 달랐다. 하지만 동일한 테이블에 대해서 위에서처럼 사용할 필요가 있을 때가 있다. 예를 들어 개의 수컷과 암컷을 짝지어 주려면 어떻게 해야 하는가? 동일한 테이블에 대해서 성이 같은지 다른지 검사해야 한다. 다음 한 예를 든다:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
    -> FROM pet AS p1, pet AS p2
    -> WHERE p1.species = p2.species AND p1.sex = "f" AND p2.sex = "m";
+--------+------+--------+------+---------+
| name   | sex  | name   | sex  | species |
+--------+------+--------+------+---------+
| Fluffy | f    | Claws  | m    | cat     |
| Buffy  | f    | Fang   | m    | dog     |
| Buffy  | f    | Bowser | m    | dog     |
```

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

3.6 배치 모드(일괄 처리 모드)로 사용하기

지금까지는 대화식으로 사용하였다. 질의를 쳐 넣고 결과를 보는 식의 반복적인 작업이었다.

작업 내용 전부를 파일에 기술해 준 후 한꺼번에 처리할 수도 있다. 이렇게 하는 작업을 배치 작업이라고 한다는 것쯤은 알아 두자. 다음과 같은 식으로 사용한다:

```
shell> mysql < batch-file
```

작업 내용을 써 둔 파일 내용을 표준 입력으로 받으면 된다. 호스트명 및 사용자 명, 패스워드를 입력할 필요가 있으면 추가로 써 준다:

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

여러분이 배치 모드에서 사용할 파일을 작성하는 것은 바로 스크립트를 작성하는 것이다.

기본 결과는 대화식으로 할 때와는 다르다. SELECT DISTINCT species FROM pet을 시켰을 때 대화식과 배치 모드에서의 결과를 보자. 내용은 같지만 형식이 다르다.

대화식:

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

배치 모드:

```
species
bird
cat
dog
hamster
snake
```

배치 모드에서도 대화 모드에서와 같은 형식으로의 출력을 원하면
mysql 실행시 -t 옵션을 넣어 주면 된다. 만약 실행되는 명령어도 출력에 포함하고 싶다면 -vvv를 붙여라.

그렇다면 무슨 유익이 있길래 배치 모드를 사용할까? 다음에 몇가지 적어 두었다.

● 질의를 자주 한다면 스크립트로 만들어 두는 것이 실행할 때마다 매번 다시 쳐 넣어주는 수고를 없애 준다.
● 이미 작성한 스크립트를 수정하여 개선할 수 있고 새로운 스크립트를 작성할 수 있는 잇점이 있다.
● 여러 줄에 걸치는 매우 복잡한 질의를 수행할 때는 배치 모드가 적당할 것이다. 실수를 했을 때 대화모드라면 전부 다시 쳐 넣어 주어야 한다. 배치 모드일때는 파일만 수정해 주면 된다. 하지만 MySQL은 readline 라이브러리(히스토리기능을 구현한 라이브러리)기능을 사용하므로 대화 모드일 때도 다시 명령어를 쳐 넣는 수고를 크게 덜 수 있다.
● 출력 결과가 굉장히 많다면 배치모드로 실행시키고 페이저(일정한 페이지 줄수로 문서를 보여주는 프로그램을 통칭하는 명칭)를 통해 보면 좋을 것이다. 다음 처럼:

shell> mysqlk < batch-file | less
● 출력 결과를 다른 파일로 저장할 수 있다. 저장된 파일은 추가 작업의 출발점으로 활용될 수 있다.

shell> mysql < batch-file > mysql.out
● 작성한 스크립트를 다른 사람과 공유할 수 있다. 다른 사람도 여러분이 작성한 스크립트를 실행할 수 있으며 참고할 수 있다.
● 어떤 작업은 성격상 배치모드에서만 실행할 수 있다. 일정한 시간 간격으로 어떤 작업을 할 때는 cron을 이용하여 배치모드에서 처리할 수 밖에 없다.