

mysql

结构

animals: (tables)

| name(string) | species(string) | birthdate(date) | rows |
|---------------------|------------------------|------------------------|-------------|
| Max | gorilla | 2011-12-11 | |
| Jony | moose | 2014-05-13 | |
| column | | | |

怎么查询::

animals:

UD

| name string | species string | birthdate date |
|----------------|-------------------|-------------------|
| Max | gorilla | 2001-04-13 |
| Sue | gorilla | 1998-06-12 |
| Max | moose | 2012-02-20 |
| Alison | llama | 1997-11-24 |
| George | gorilla | 2011-01-09 |
| Spot | iguana | 2010-07-23 |
| Ratu | orangutan | 1989-09-15 |
| Eli | llama | 2002-02-22 |

这是一种数据存储

Table Structure

diet:

| species string | food string |
|-------------------|----------------|
| llama | plants |
| brown bear | fish |
| brown bear | meat |
| brown bear | plants |
| orangutan | plants |
| orangutan | insects |

Select food
from diet
Where species =
'orangutan'

-
-
-
-
-
-

Mark the rows and columns
that will appear in the
result table.

zoo=> select food from diet where species = 'orangutan';
food

I

plants
insects
(2 rows)

```

zoo=> select 2+2, 4+4, 6+6;
?column? | ?column? | ?column?
-----+-----+-----
        4 |          8 |         12
(1 row)

```

```

zoo=> select 2+2 as sum;
sum
-----
        4
(1 row)

```

表关系:

Related Tables

pictures:

| id | name | filename |
|----|---------|--------------------|
| 1 | Fluffy | fluffsocute.jpg |
| 2 | Monster | monstie-basket.png |
| 3 | George | george.jpg |
| : | : | : |

votes:

| left | right | winner |
|------|-------|--------|
| 2 | 3 | 3 |
| 1 | 3 | 1 |
| 2 | 1 | 1 |
| 3 | 2 | 3 |
| 3 | 1 | 1 |
| 2 | 3 | 3 |

2 was shown with 1, and 1 got the vote.

Monster was shown with Fluffy, and Fluffy got the vote.

这也是我们能用数据库查询进行的一项操作

唯一性和键

唯一性做为主键

Joining Tables

| name string | Species string | food string |
|----------------|-------------------|-----------------|
| Alison | llama | plants |
| Smokey | brown bear | fish |
| Smokey | brown bear | meat |
| Smokey | brown bear | plants |
| Ratu | orangutan | plants |
| Ratu | orangutan | insects |
| : | : | 让数据库只返回符合特定标准的行 |

select

animals.name,
animals.Species,

diet.food

from animals join diet

on animals.Species =

diet.Species

where food = 'fish';

Joining Tables

animals
name, species



join on species
name, species, food

restriction
food = 'fish'

count +
aggregation

diet
species, food



mysql数据类型:

文本和字符串类型

text — 任何长度的字符串，例如 Python **str** 或 **unicode** 类型。

char(n) — 长度为 n 个字符的字符串。

varchar(n) — 长度上限为 n 个字符的字符串。

数值类型

integer — 整型值，例如 Python **int**。

real — 浮点型值，例如 Python **float**。精确到小数点后 6 位。

double precision — 精度更高的浮点型值。精确到小数点后 15 位。

decimal — 精确的十进制值。

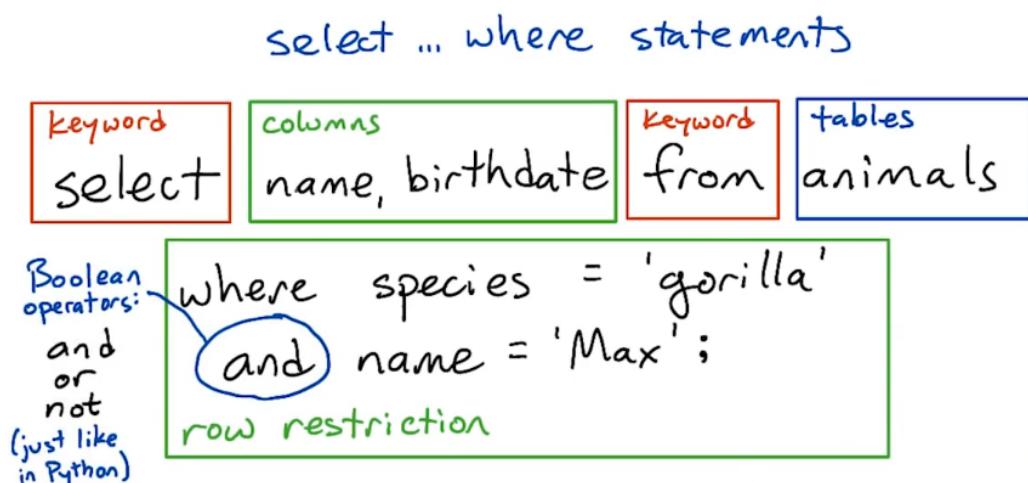
日期和时间类型

date — 日历日期，包括年月日。

time — 一天中的时间。

timestamp — 日期和时间相结合。

选择语法



具有 **where** 条件的 **select** 语句的语法：

```
select columns from tables where condition ;
```

每列用逗号分隔；使用 * 可选择所有列。

condition 是列值的布尔表达式。SQL 支持布尔运算 **and**、**or** 和 **not**，和 Python 中的运算规则一样。

表达式 **(not X) and (not Y)** 和 **not (X or Y)** 可以互换，这是由德摩根定律决定的。你可以阅读这篇[维基百科文章](#)了解详情。

Comparison Operators

```
select name from animals  
where species = 'llama' and  
birthdate >= '1995-01-01' and  
birthdate <= '1998-12-31' ;
```

select的其他语法

以下是上个视频中介绍的新 **select** 子句：

... **limit count** 只返回结果表格的前 **count** 行。

... **limit count offset skip** 返回前 **skip** 行之后的 **count** 行。

... **order by columns ... order by columns desc** 使用 **columns** (一列或多列，用逗号分隔) 作为排序键对列排序。数字列将按数字顺序排序；字符串列将按字母顺序排序。**desc** 表示顺序为逆序 (**desc**-结尾的排序)。

... **group by columns** 更改集合的行为，例如 **max**、**count** 和 **sum**。对于 **group by**，集合将为 **columns** 中的每个唯一值返回一行。

Just a few of the SELECT clauses

| Which clauses would you use for each of these situations? Check all appropriate boxes. | where | limit | offset | order by | group by |
|---|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Find the ten oldest gorillas. | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| List all the animals, in alphabetical order, ten per page. | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| Find out which one species we have the most of. | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |

支持切片工具的

Why do it in the database?

| SQL clauses + aggregation | Python list operations |
|----------------------------|---|
| count(*) | len(results) |
| Limit 100 offset 10 | results[10:110] |
| order by column | sorted(results, key=lambda x: x[column]) |

一个例子

Select 子句

这这些是我们到目前为止见到的所有 select 子句：

where

where 子句表示限制条件 — 从表格中过滤出符合特定规则的行。**where** 支持等于、不等于和布尔运算符等：

- **where species = 'gorilla'** — 仅返回物种列的值为“gorilla”的行。
- **where name >= 'George'** — 仅返回名称列在“George”之后（按字母顺序）的行。
- **where species != 'gorilla' and name != 'George'** — 仅返回物种不是“gorilla”并且名称不是“George”的行。

limit / offset

limit 子句对结果表格可以返回的行数做出限制。可选 **offset** 子句表示要在结果中跳过多少行。所以 **limit 10 offset 100** 将返回 10 条结果，从第 101 行开始。

order by

order by 子句告诉数据库如何对结果排序 — 通常根据一个或多个列。所以 **order by species, name** 表示首先按照物种列排序，然后在每个物种里按照名称排序。

排序发生在 **limit/offset** 之前，所以你可以使用它们来提取出按字母顺序排列的页面结果（想想字典的页面）。

可选 **desc** 修饰符告诉数据库按照降序对结果排序，例如从大到小或从 Z 到 A。

group by

group by 子句只能用于汇总，例如 **max** 或 **sum**。没有 **group by** 子句的话，对集合执行选择语句将对整个选定表格进行汇总，只返回一行。对于 **group by** 子句，它将对 **group by** 子句中的列或表达式的每个唯一值返回一行。

Count ALL the species!

One possible answer:

```
select count(*) as num, species columns
  from animals tables
 group by species aggregation
sorting order by num desc;
```

加入值

insert 语句的基本语法：

```
insert into table ( column1, column2, ... ) values ( val1, val2, ... );
```

如果值和表格的列顺序一样（从第一列开始），则不需要在 **insert** 语句中指定列：

```
insert into table values ( val1, val2, ... );
```

例如，如果表格有三列 (**a**, **b**, **c**)，你想要向 **a** 和 **b** 中插入值，你可以在 **insert** 语句中省略列名称。但是如果你想向 **b** 和 **c** 或 **a** 和 **c** 中插入值，则需要指定列。

单个 **insert** 语句只能插入一个表格中（而 **select** 语句可以使用 **join** 从多个表格中获取数据）。

```
insert into animals  
values ('Wibble', 'opossum', '2014-11-14');  
  
insert into animals (name, species, birthdate)  
values ('Wibble', 'opossum', '2014-11-14');
```

insert 语句的基本语法：

```
insert into table ( column1, column2, ... ) values ( val1, val2, ... );
```

如果值和表格的列顺序一样（从第一列开始），则不需要在 **insert** 语句中指定列：

```
insert into table values ( val1, val2, ... );
```

例如，如果表格有三列 (**a**, **b**, **c**)，你想要向 **a** 和 **b** 中插入值，你可以在 **insert** 语句中省略列名称。但是如果你想向 **b** 和 **c** 或 **a** 和 **c** 中插入值，则需要指定列。

单个 **insert** 语句只能插入一个表格中（而 **select** 语句可以使用 **join** 从多个表格中获取数据）。

引号不能有空格

加入 **join**

要连接 (join) 两个表格，首先选择连接条件，即数据库将表格一中的行与表格二中的行相匹配时采用的规则。然后编写连接语句，包含每个表格中的列。

例如，如果你想连接表格 T 和 S，其中 T.color 和 S.paint 要相同，则需要使用 `T join S on T.color = S.paint` 编写一个 select 语句。

Find the Fish-eaters

Find the names of all the animals that eat fish.

```
select animals.name  
from animals join diet  
on animals.species = diet.species  
where food = 'fish';
```

columns (just one!) joined tables
restriction join condition

Find the Fish-eaters

Find the names of all the animals that eat fish.

```
select name from animals, diet  
where animals.species = diet.species  
and diet.food = 'fish';
```

columns tables
boolean! restriction

汇总之后的having

having 子句和 **where** 子句工作原理差不多，但是它应用于 **group by** 汇总发生之后。语法如下所示：

```
select columns from tables group by column having condition;
```

通常，至少有一列将是汇总函数，例如对表格的某列执行 **count**、**max** 或 **sum** 操作。要对汇总列应用 **having**，你需要使用 **as** 为其设定名称。例如，如果你有一个商店所售商品的表格，并且想要找出售出数量超过 5 件的所有商品，则可以使用：

```
select name, count(*) as num* from sales having num > 5;
```

你可以在 **select** 语句中仅使用 **where**、仅使用 **group by**、或使用 **group by** 和 **having**、或使用 **where** 和 **group by** 或三个都用到！

但是如果沒有 **group by** 的情况下，使用 **having** 通常是不合理的。

如果你同时使用了 **where** 和 **having**，**where** 条件将过滤即将被汇总的行，**having** 条件将过滤汇总后的行。

关于 **having** 的更多详情请参阅以下网址：

<http://www.postgresql.org/docs/9.4/static/sql-select.html#SQL-HAVING>

你可以采用几种不同的方式来解决这一问题，下面是其中一个示例：

```
select food, count(animals.name) as num
      from diet join animals
      on diet.species = animals.species
     group by food
    having num = 1
```

下面是另一个示例：

```
select food, count(animals.name) as num
      from diet, animals
     where diet.species = animals.species
     group by food
    having num = 1
```

After Aggregating

Which species does the zoo have only one of?

RIGHT: select species, count(*) as num
from animals group by species
having num = 1;

where is a restriction on the source tables.

having is a restriction on the result...after aggregation!

```
1 #  
2 # Find the one food that is eaten by only one animal.  
3 #  
4 # The animals table has columns (name, species, birthdate) for each individual.  
5 # The diet table has columns (species, food) for each food that a species eats.  
6 #  
7  
8 QUERY = '''  
9 select food, count(animals.name) as num  
10    from diet, animals  
11    where diet.species = animals.species  
12    group by food  
13    having num = 1;  
14 ...  
15  
16
```

另外的链接

taxonomy

此表列出动物园中各物种的（部分）生物分类学名称。可用于辨别物种之间进化论角度的亲缘关系。

- name — 物种的俗称（例如，“jackal”（豺））
- species — 分类学物种名称（例如，“aureus”（亚洲胡狼））
- genus — 分类学属名（例如，“Canis”（犬属））
- family — 分类学科名（例如，“Canidae”（犬科））
- t_order — 分类学目名（例如，“Carnivora”（食肉目））

如果你对此分类一无所知，也无需担心，本课程不需要详细掌握这些信息。但如果你对此感兴趣，可查阅维基百科相关文章[分类学](#)和[生物分类](#)。

ordernames

此表列出 **taxonomy** 表中各分类学目的俗称。

- t_order — 分类学目名（例如，“Cetacea”（鲸目））
- name — 俗称（例如，“whales and dolphins”（鲸和海豚））

zoo 数据库中的所有表格

如果你不知道表格是什么样的，你始终可以使用 **select * from table** 查看表格内容。以下是 zoo 数据库中包含的所有表格的总结：

animals

此表列出动物园中的各个动物。每个动物仅占一行。可能存在多个动物同名，甚至多个同物种动物同名的情况。

- name — 动物的名字（例如：“George”）
- species — 动物所属物种（例如：“gorilla”（大猩猩））
- birthdate — 动物的出生日期（例如：'1998-05-18'）

diet

此表对照列出各物种及其所吃的食物。动物园中的每个物种至少吃一种食物，许多物种吃多种食物。如果某物种食用一种以上的食物，则该物种将占多行。

- species — 物种名称（例如：“hyena”（ 獬狗 ））
- food — 物种所吃食物的名称（例如：“meat”（ 肉类 ））

The SQL for it

上面的表由以下 SQL 命令创建。**create table** 命令将在第 4 课中详细介绍，可在此简单了解：

```
create table animals (
    name text,
    species text,
    birthdate date);
create table diet (
    species text,
    food text);
create table taxonomy (
    name text,
    species text,
    genus text,
    family text,
    t_order text);

create table ordernames (
    t_order text,
    name text);
```

以下是其中一个解决方案：

```
select ordernames.name, count(*) as num
  from animals, taxonomy, ordernames
 where animals.species = taxonomy.name
   and taxonomy.t_order = ordernames.t_order
 group by ordernames.name
 order by num desc
```

以下是其中一个解决方案：

```
select ordernames.name, count(*) as num
  from animals, taxonomy, ordernames
 where animals.species = taxonomy.name
   and taxonomy.t_order = ordernames.t_order
 group by ordernames.name
order by num desc
```

以下是另一个解决方案，这次使用的是显式连接格式：|

```
select ordernames.name, count(*) as num
  from (animals join taxonomy
        on animals.species = taxonomy.name)
       as ani_tax
  join ordernames
    on ani_tax.t_order = ordernames.t_order
 group by ordernames.name
order by num desc
```

我认为上面的版本比下面的更容易读懂，因为在显式连接格式中，你需要明确告诉数据库按照什么顺序连接表格 — `((a join b) join c)` — 而不是让数据库自己去判断。

如果你使用的是更加框架性的数据库（例如 SQLite），那么显式连接格式可能会存在性能优势。但是对于我们将在下节课中用到的面向服务器的数据库系统 PostgreSQL，query planner 应该消除任何差异。