

关系数据库 MySQL 和 ORM

简介

Web开发中，离不开数据库的支持，在本节实验中我们将学习关系数据库 MySQL 和 ORM SQLAlchemy。MySQL (<https://www.mysql.com/>) 是应用最广泛的数据库，而 SQLAlchemy (<https://www.sqlalchemy.org/>) 是 Python 语言中最流行的 ORM。

知识点

- MySQL 基础知识
- 关系数据库基础
- SQLAlchemy 基础知识

MySQL

MySQL 是应用最广泛的关系数据库。关系数据库使用关系模型作为数据组织存储的方式，可以分为四级结构：

- 数据库，比如一个应用对应一个数据库；
- 表，一个数据库包含若干张表；
- 记录，一个表包含多条记录；
- 字段，一条记录包含若干字段；

一张表的结构可以想象成一张 Excel 表，由多个字段组成，每一个字段都可以存储特定类型的数据，比如字符串或者数字，在创建表的时候可以指定类型。表与表之间通过关系连接（逻辑上的关系），查询数据时可以通过关系跨越多张表进行查询。

关系数据库最重要的特性是满足 ACID 性质：

- A atomicity 代表原子性；
- C consistency 代表一致性；
- I isolation 代表隔离性；
- D durability 代表持久性；

ACID 能够保证事务的可靠性，什么意思呢？就是说能够保证一系列的数据库操作组成的一个完整逻辑过程，要么全部被执行，要么彻底不执行，不会出现执行一半的情况。例如银行转账，从原账户扣除金额，以及向目标账户添加金额，这两个数据库操作的总和，构成一个完整的逻辑过程，不

可拆分。

🔗 楼+之Python实战第10期 (/courses/1190)

为了满足 ACID 性质，MySQL 支持各种约束，比如插入一条数据时，需要检查外键是否存在，这些操作虽然能确保数据的一致性，但是很多时候却降低了开发操作的能力，所以在如今的互联网网站中，如果有高并发的需求往往不再使用关系数据库的 ACID 性质，更有的直接使用非关系数据库。

除此之外，这里再补充几点关系数据库中关系键的基本概念知识，以帮助理解下文内容。

- **主键**（英语：primary key）。数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。一个数据列只能有一个主键，且主键的取值不能缺失，即不能为空值（Null）。
- **外键**（英语：foreign key）。其实在关系数据库中，每个数据表都是由关系来连系彼此的关系，父数据表（Parent Entity）的主键（primary key）会放在另一个数据表，当做属性以创建彼此的关系，而这个属性就是外键。

比如，学生跟老师之间对应着教学关系，学生数据表会有个属性叫指导老师（外键），而这个值就是对应到老师数据表的老师代号（主键）。

在接下来的内容中，我们将学习基本的 MySQL 操作。推荐学习实验楼的 Mysql 基础课程 (<https://www.shiyanlou.com/courses/9>)，完成前三个实验即可。

virtualenv

在本节实验中会用到使用 Python 操作 MySQL 数据库，由于 Python 版本众多，相应依赖的模块的版本也很多，如果在一台电脑上同时开发多个应用就常常会出现版本冲突，这个时候虚拟环境 virtualenv (<https://virtualenv.pypa.io/en/latest/>) 就派上了用场。

使用虚拟环境可以隔离出一个独立的 Python 运行环境，在这个环境中安装私有包，不会影响全局 Python 解释器，这样就能有效的避免包的混乱和版本冲突。

安装 virtualenv：

```
sudo pip3 install virtualenv
```

创建一个虚拟环境：

```
$ virtualenv venv # 创建了一个名为 venv 的虚拟环境
Using base prefix '/usr'
New python executable in /home/shiyanlou/venv/bin/python3
Also creating executable in /home/shiyanlou/venv/bin/python
Please make sure you remove any previous custom paths from your /home/shiyanlou/.pydistutils.cfg file.
Installing setuptools, pip, wheel...
done.
```

进入 venv 的虚拟环境：

📍 楼+之Python实战第10期 (/courses/1190)

```
shiyanolou:~/ $ source venv/bin/activate
(venv) shiyanolou:~/ $ # 进入虚拟环境后，在最前面的小括号中会显示虚拟环境的名称 (venv)
```

查看虚拟环境中默认的 pip 安装包：

```
(venv) shiyanolou:~/ $ pip list
Package      Version
-----
pip          18.1
setuptools   40.6.2
wheel        0.32.3
```

后续的开发工作就可以全部在虚拟环境中进行。当开发完成后可以把环境中的安装包依赖名称导出：

```
pip freeze > requirements.txt
```

退出虚拟环境：

```
(venv) shiyanolou:~/ $ deactivate
shiyanolou:~/ $
```

如果需要删除某个虚拟环境可以直接找到该虚拟环境的文件夹，执行删除操作就行：

```
rm -rf venv
```

环境准备


实验楼环境中已经安装了 MySQL 数据库软件，每次启动实验后，需要手动启动 MySQL 服务。在实验桌面上启动终端后，通过以下命令启动数据库：

```
$ sudo service mysql start
```

还需要安装一个包

```
$ sudo pip3 install mysqlclient
```

这个包的作用是让 Python 可以连接 MySQL 数据库，如果不安装使用 Python 及 SQLAlchemy 连接 MySQL 数据库的时候则会出现下面的错误：

 Import Error: No module named MySQLdb (courses/1190)

或者

```
Module MySQLdb not found
```

这类的错误，原因就是实验环境中没有安装 `mysqlclient`，或者你在使用 `virtualenv`（下面 Python 连接数据库部分会学习到）的时候没有在虚拟环境中安装这个包（也有可能安装了，但没有重新激活 `virtualenv` 环境也会造成这个错误）。

如果这个错误出现在用 `virtualenv`（虚拟环境）创建的虚拟环境里，需要在 `virtualenv` 中执行 `pip3 install mysqlclient`，然后执行 `deactivate` 退出虚拟环境再进到 `virtualenv` 就可以解决。

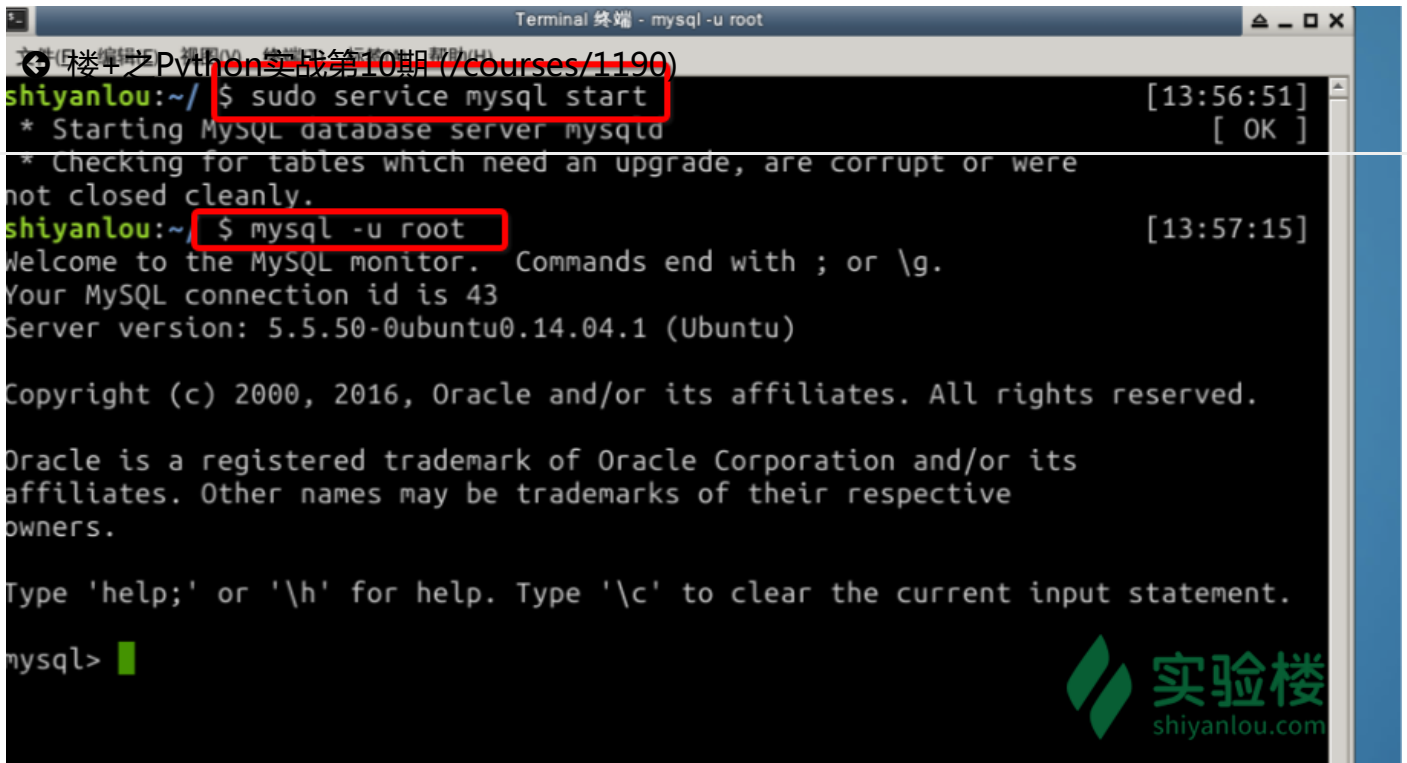
如果没用 `virtualenv` 创建虚拟环境，报这个错执行 `sudo pip3 install mysqlclient` 即可。

`virtualenv`（虚拟环境）的作用是避免 `python` 版本和软件包版本的使用冲突，例如系统环境里只安装了 `python3.5`，而某程序需要运行在 `python2.7` 上，或者不同程序用到软件包的不同版本，都可以用 `virtualenv` 来解决，在 `virtualenv` 中安装一些 `Python` 软件包，需要重新进入 `virtualenv` 环境才能生效。

数据库启动成功以后，通过以下命令链接到数据库：

```
$ mysql -u root
```

`MySQL` 数据库由服务器端和客户端组成，可以通过客户端连接到服务器。以上命令中，使用 `root` 账户登陆服务器，`root` 账户是 `MySQL` 数据库的超级管理员账户，未指定密码信息，这是因为实验楼环境中未设置 `root` 账户的密码。登录成功后如下图所示：



The screenshot shows a terminal window titled "Terminal 终端 - mysql -u root". The user shiyanlou is at the prompt ~/ \$ and enters the command `sudo service mysql start`. The output shows the MySQL service starting successfully. Then, the user enters `mysql -u root` to log into the MySQL monitor. The terminal displays the MySQL welcome message and the prompt `mysql>`. A red box highlights the command `mysql -u root`. A watermark for "实验楼 shiyanlou.com" is visible in the bottom right corner.

```
Terminal 终端 - mysql -u root
shiyanlou:~/ $ sudo service mysql start [13:56:51]
* Starting MySQL database server mysqld [ OK ]
* Checking for tables which need an upgrade, are corrupt or were
not closed cleanly.
shiyanlou:~/ $ mysql -u root [13:57:15]
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43
Server version: 5.5.50-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

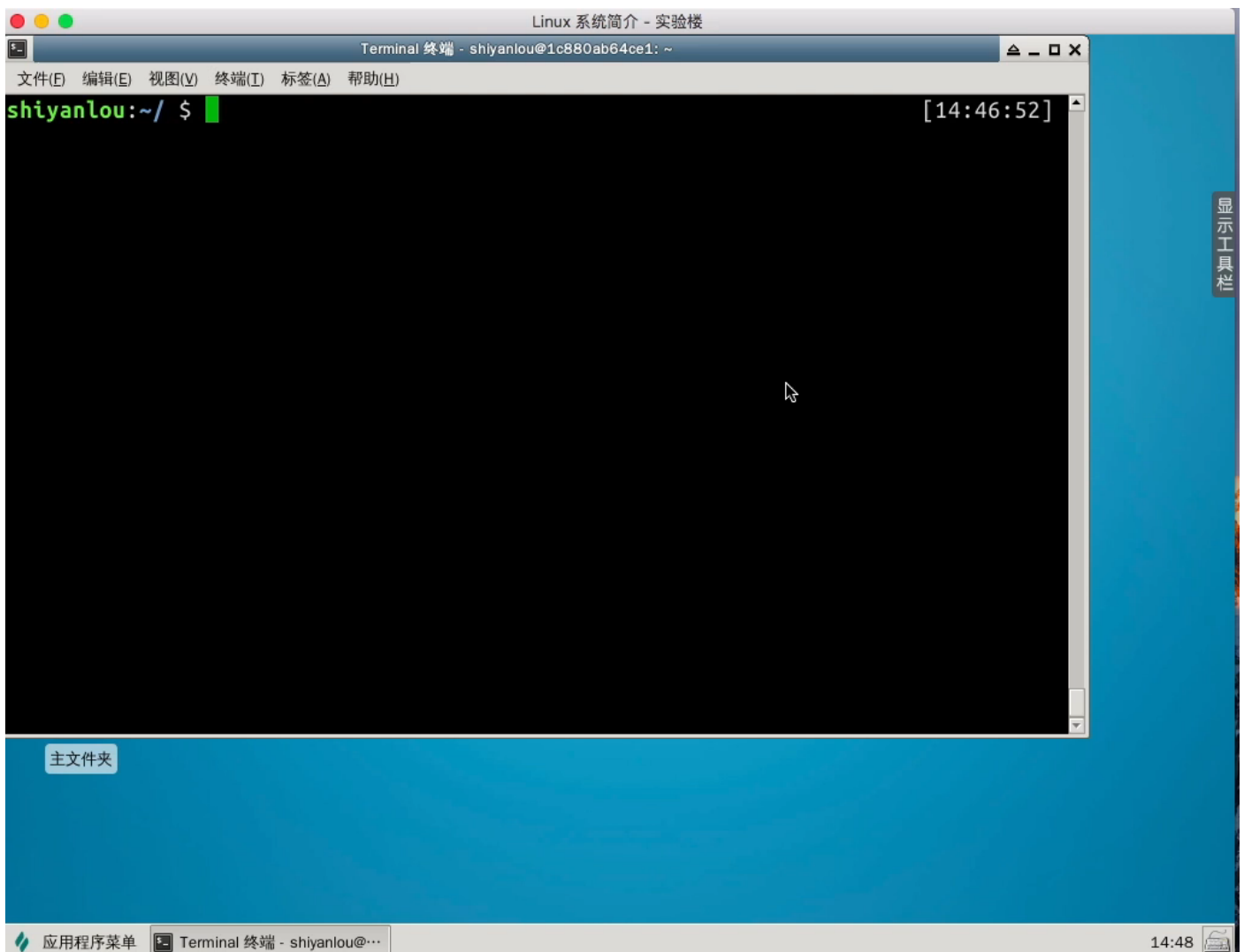
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

后面的所有 MySQL 命令都通过 MySQL 客户端输入(也就是以 `mysql>` 标识符开始的命令)。

MySQL 环境操作视频：



基本操作

楼+之Python实战第10期 (/courses/1190)

创建数据库非常简单，只需要输入 `create database <db_name>;` 就可以了，其中 `<db_name>` 代表数据库名称，查看所有数据库可以通过 `show databases;` (注意，这里多了一个 `s`) 命令，而删除一个数据库则可以通过 `drop database <db_name>;`，下面的例子演示了数据库的创建和删除过程：

```
mysql> create database shiyanlou;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| shiyanlou |
| sys |
+-----+
5 rows in set (0.00 sec)

mysql> drop database shiyanlou;
Query OK, 0 rows affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.00 sec)
```

需要注意的是 `show databases;` 的输出结果显示了多个数据库，这是因为 MySQL 服务器有几个默认的数据库，该命令输出结果在不同的 MySQL 版本下可能不一致。

数据库创建成功以后，就可以创建表了。在创建表之前，需要通过 `use shiyanlou;` 命令连接到 `shiyanlou` 数据库。

创建表的基本命令如下：

```
CREATE TABLE 表的名字
(
  列名a 数据类型(数据长度),
  列名b 数据类型(数据长度),
  列名c 数据类型(数据长度)
);
```

下面尝试创建一张表名为 `user` 的表，该表有 3 个字段：

🔗 楼+之Python实战第10期 (/courses/1190)

- `id` 编号，整数类型，使用 `int` 类型；
- `name` 用户名，字符串，使用 `varchar` 可变字符类型；
- `email` 邮箱，字符串，使用 `varchar` 可变字符型；

在 MySQL 客户端输入下面的命令：

```
mysql> create database shiyanlou;
mysql> use shiyanlou;
Database changed
mysql> create table user
-> (
-> id int(10),
-> name varchar(20),
-> email varchar(64)
-> );
Query OK, 0 rows affected (0.03 sec)
mysql> show tables;
+-----+
| Tables_in_shiyanlou |
+-----+
| user                  |
+-----+
1 row in set (0.00 sec)
```

通过以上命令就可以创建出 `user` 表，需要注意的是当输入 `create table user` 回车以后，客户端会自动识别出这是一个未完成的命令，所以会出现提示符 `->`。表创建成功以后，可以通过 `show tables;` 查看所有的表，删除表可以通过 `drop table <table_name>;`。

如果想查看一张表的字段信息，可以通过 `show create table <table_name>;` 或者 `describe <table_name>;` 指令，如下所示：

🔗 实验楼之Python实战第10期 (/courses/1190)

```
mysql> create table user (
  `id` int(10) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  `email` varchar(64) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
1 row in set (0.00 sec)

mysql> describe user;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(10)       | YES  |     | NULL    |       |
| name  | varchar(20)   | YES  |     | NULL    |       |
| email | varchar(64)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

MySQL 表的字段支持多种类型，如整数，浮点数，字符串，时间戳等，篇幅有限就不一一介绍了。

表创建成功以后，就可以插入数据了。可以使用 `insert` 指令插入数据，完整的命令格式如下：

```
INSERT INTO 表的名字(列名a,列名b,列名c) VALUES(值1,值2,值3);
```

当插入的值和表定义的字段数量一致且顺序一致时，可以忽略列名信息，通过以下命令往 `user` 表中插入数据：

```
mysql> insert into user(id, name, email) values(1, 'aiden', 'luojin@simplecloud.cn');
Query OK, 1 row affected (0.02 sec)

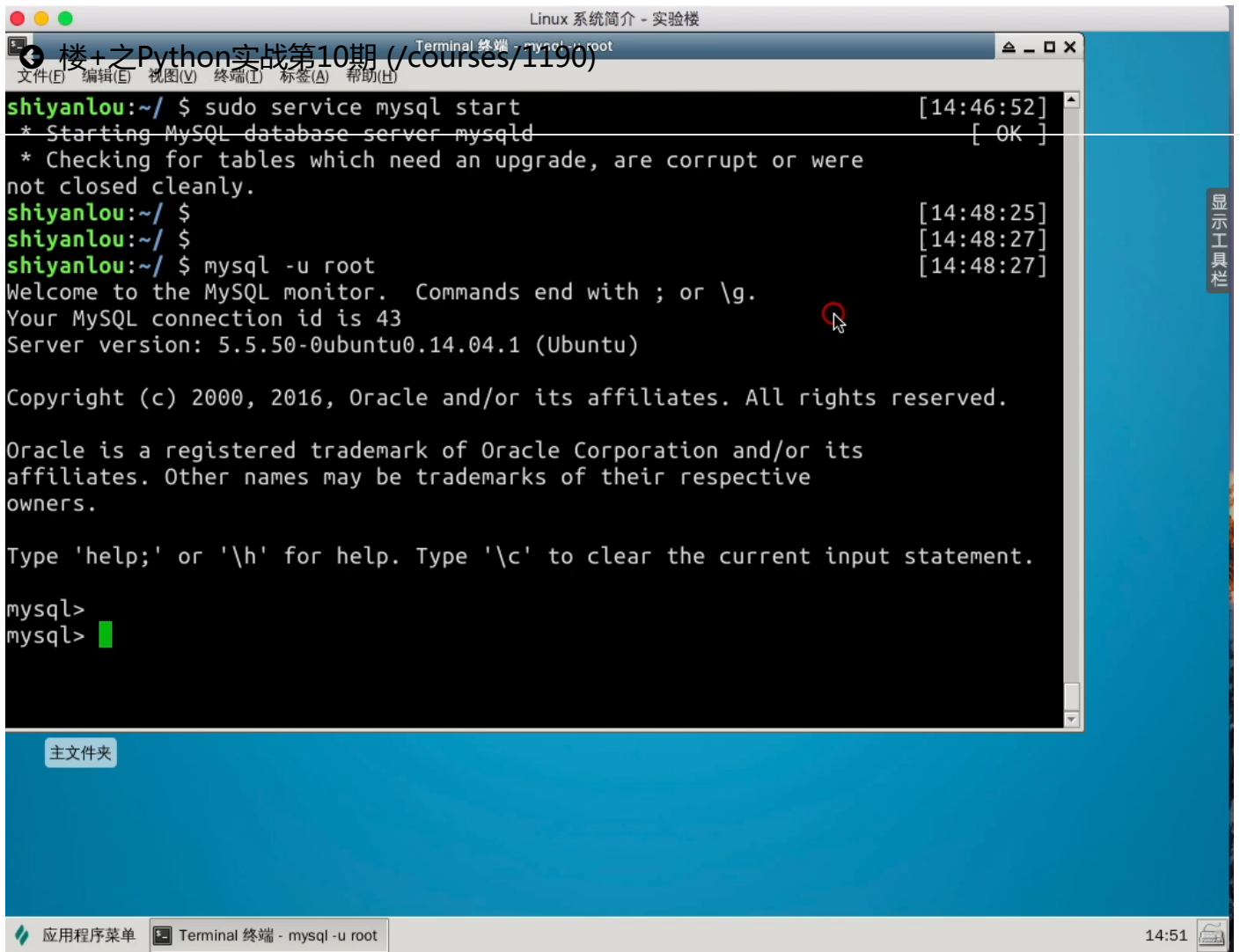
mysql> insert into user values(2, 'lxttx', 'lxttx@gmail.com');
Query OK, 1 row affected (0.00 sec)
```

成功的插入两条数据。查看某张表的数据可以通过 `select * from <table_name>;` 查看：

```
mysql> select * from user;
+-----+-----+-----+
| id  | name  | email                      |
+-----+-----+-----+
| 1   | aiden | luojin@simplecloud.cn      |
| 2   | lxttx | lxttx@gmail.com           |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

到这里我们已经学会了创建数据库和表，插入记录，查询记录等基本操作，是不是非常简单？

MySQL 基本操作视频：



The screenshot shows a terminal window titled "Terminal 终端 - mysql -u root" on a Linux system. The user 'shiyanolou' is at the prompt '~/' and runs 'sudo service mysql start'. The output shows the MySQL service starting successfully. Then, the user runs 'mysql -u root' and is prompted with the MySQL monitor welcome message. The terminal shows the MySQL version as 5.5.50-0ubuntu0.14.04.1 (Ubuntu). The terminal window is part of a desktop environment with a blue background and a taskbar at the bottom.

```
shiyanolou:~/ $ sudo service mysql start [14:46:52]
* Starting MySQL database server mysqld [ OK ]
* Checking for tables which need an upgrade, are corrupt or were
not closed cleanly.
shiyanolou:~/ $ [14:48:25]
shiyanolou:~/ $ [14:48:27]
shiyanolou:~/ $ mysql -u root [14:48:27]
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43
Server version: 5.5.50-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2016, 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>
mysql>
```

约束

MySQL 是一个关系数据库，可以通过约束限制一些数据操作。比如插入数据时，可以检查该数据是否满足关系约束，如果不满足则拒绝操作。下面通过一个简单的例子进行演示。

在前一小节中，我们创建了 `user` 表，且插入了下面的两条数据：

```
mysql> select * from user;
+-----+-----+-----+
| id    | name  | email                               |
+-----+-----+-----+
| 1     | aiden | luojin@simplecloud.cn              |
| 2     | lxttx | lxttx@gmail.com                   |
+-----+-----+-----+
```

现在重新再插入一条数据：

```
mysql> insert into user (id, name, email) values(3, 'lxttx', 'lxttx@gmail.com');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from user;
+-----+-----+-----+
| id    | name   | email                |
+-----+-----+-----+
| 1     | aiden  | luojin@simplecloud.cn |
| 2     | lxttx  | lxttx@gmail.com      |
| 3     | lxttx_1 | lxttx@gmail.com      |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

数据成功插入，但是现在有点问题，对于一个正式上线的数据库来说，一般不允许在一张用户表存在邮箱相同的用户，也就是说要求 email 字段具有唯一约束。这个时候怎么办呢？对于一个已经存在的表，可以通过 alter 指令修改表的字段，设置唯一约束：

```
mysql> alter table user modify email varchar(64) unique;
ERROR 1062 (23000): Duplicate entry 'lxttx@gmail.com' for key 'email'
mysql> delete from user where id = 3;
Query OK, 1 row affected (0.00 sec)

mysql> alter table user add constraint unique (email);
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> insert into user values(3, 'lxttx_1', 'lxttx@gmail.com');
ERROR 1062 (23000): Duplicate entry 'lxttx@gmail.com' for key 'email'
```

以上命令中，先尝试修改 email 字段设置唯一约束，但是数据库报错，因为 user 表中已存在重复的邮箱记录。接着使用 delete from user where id = 3; 删除了重复记录，该命令的含义是删除 id 为 3 的记录。接着成功设置了 email 的唯一索引，然后插入重复记录时已被禁止。添加唯一约束还有一种方式为 alter table user modify email varchar(64) unique;，这种方式实际上是通过修改字段添加唯一索引。

还有一种约束，是跨表的。在表中插入数据项时，要求该数据项的某一个字段值必须已经在其他表中存在，比如外键约束。插入数据时，如果不满足外键约束则不允许插入，删除数据如果破坏了外键约束也会禁止删除数据。外键约束创建时，必须要求另一张表中存在主键，主键在表中能唯一的确定某一行的值。下面尝试创建一张具有外键约束的 course 表，在创建 course 表之前，需要在 user 表中设置主键：

```
mysql> alter table user add constraint pk_id primary key (id);
Query OK, 0 rows affected (0.14 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> describe user;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(10)       | NO   | PRI | NULL    |       |
| name  | varchar(20)   | YES  | UNI | NULL    |       |
| email | varchar(64)   | YES  | UNI | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

通过 `alter` 指令成功在 `user` 中设置 `id` 字段为主键。通过 `describe user;` 命令输出确实发现 `id` 列变成了主键（因为 `Key` 列的值为 `PRI`，代表 `primary`）。接着创建 `course` 表：

```
mysql> create table course
-> (
-> id int(10) auto_increment,
-> name varchar(64),
-> teacher_id int(10),
-> primary key (id),
-> constraint fk_user foreign key (teacher_id) references user(id)
-> );
Query OK, 0 rows affected (0.04 sec)
```

以上命令相对于之前建表命令增加了一些内容，首先通过 `auto_increment` 指定了字段 `id` 为自增模式，这样每次插入数据时不用指定该字段，插入数据后这个字段会自动增加。接着通过 `primary key (id)` 语句指定了这个表的主键为 `id`，最后设置了字段 `teacher_id` 为外键，且关联到了 `user` 表的 `id` 字段。现在尝试往 `course` 表中插入一条 `teacher_id` 在 `user` 表中不存在的数据：

```
mysql> insert into course(name, teacher_id) values('Python基础', 100);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`shiyuanlou`.`course`, CONSTRAINT `fk_user` FOREIGN KEY (`teacher_id`) REFERENCES `user` (`id`))
```

环境中输入中文并不方便，需要点击右下角的小键盘才可以切换输入法，如果觉得太麻烦可以直接输入英文即可，实验操作都是一致的，并不要求每个内容都完全相同，但需要理解每一个步骤和操作参数的作用。

可以看到，数据插入失败。因为 `user` 表中不存在 `id` 为 100 的数据，不满足外键约束。删除数据时，如果违反了外键约束也将删除失败：

实验楼之Python实战第10期 (/courses/1190)

```
mysql> select * from user;
+----+-----+-----+
| id | name  | email                |
+----+-----+-----+
| 1  | aiden | luojin@simplecloud.cn |
| 2  | lxttx | lxttx@gmail.com      |
+----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> insert into course(name, teacher_id) values('Python 基础', 2);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select * from course;
+----+-----+-----+
| id | name      | teacher_id |
+----+-----+-----+
| 2  | Python 基础 | 2          |
+----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> delete from user where id = 2;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`shiyianlou`.`course`, CONSTRAINT `fk_user` FOREIGN KEY (`teacher_id`) REFERENCES `user` (`id`))
```

上面的例子中，为了演示删除失败的情况，先往 `course` 表中插入了一条数据，接着尝试从 `user` 表中删除 `id` 为 2 的数据，发现删除失败，这是因为如果删除该记录就违背了已存在的外键约束。

很多时候需要进行联合跨表查询，比如想知道课程的名称，课程老师的名称，邮箱信息就需要联合查询：

```
mysql> select * from course join user on course.teacher_id = user.id;
+----+-----+-----+----+-----+-----+
| id | name      | teacher_id | id | name  | email                |
+----+-----+-----+----+-----+-----+
| 2  | Python 基础 | 2          | 2  | lxttx | lxttx@gmail.com      |
+----+-----+-----+----+-----+-----+
1 row in set (0.00 sec)
```

例子中使用 `join` 指令进行联合查询，`on` 关键字指定了两张表的关联方式。

MySQL 约束操作视频：



```
mysql> insert into user(id, name, email) values(2,'lxttx','lxttx@gmail.com');
Query OK, 1 row affected (0.02 sec)

mysql> select * from user;
+-----+-----+-----+
| id    | name  | email                |
+-----+-----+-----+
| 1     | aiden | luojin@simplecloud.cn |
| 2     | lxttx | lxttx@gmail.com      |
+-----+-----+-----+
2 rows in set (0.01 sec)

mysql>
```

主文件夹

应用程序菜单

Terminal 终端 - mysql -u root

15:09

练习题：创建问答表

在 shiyanlou 数据库中创建新的一个表 question，用来存储课程问题，需要满足以下需求：

1. 包含 id，name，content，course_id 字段
2. 主键为 id，设置为自增
3. name，course_id 为必填
4. course_id 为外键

表创建之后，点击 下一步，系统将自动检测完成结果。

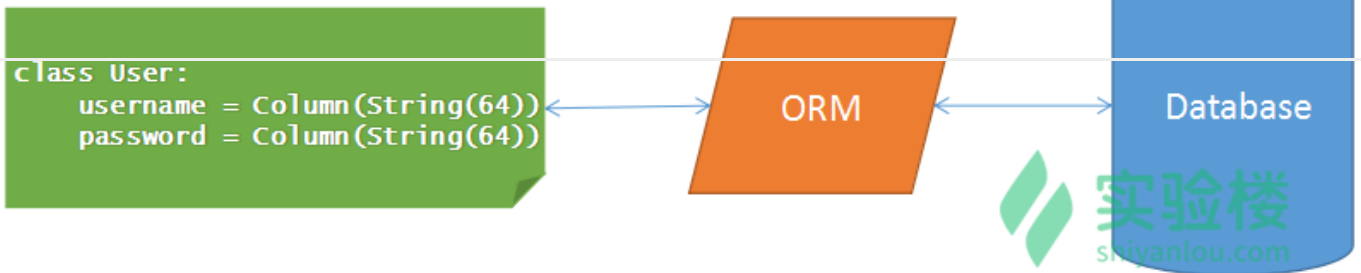
总结

学习完以上内容，我们已经能够进行 MySQL 的基本操作了。MySQL 博大精深，涉及的知识点非常多，比如上面的内容就没有涉及索引管理和性能优化方面的内容。如果想学习更多的 MySQL 内容，一方面可以在实验楼学习其他课程，另一方面要多查阅 MySQL 文档 (<https://dev.mysql.com/doc/>)。

SQLAlchemy

在实际项目中，一般都不直接写 SQL 语句访问数据库，而是通过 ORM 工具。ORM 全称是对象关系映射（Object Relational Mapping），有了 ORM 以后可以将 Python 对象映射到数据库中，这样就不用再编写各种 SQL 语句了。而在 Python 语言中，SQLAlchemy 是非常强大的 ORM 包，非常值得学习。同时 SQLAlchemy 支持多种关系数据，如果项目后期需要切换到其他类型的数据库，通过 SQLAlchemy 也比较容易。

楼+之Python实战第10期 (/courses/1190)



环境准备

本节实验环境主要基于 `virtualenv` 建立。`virtualenv` 的作用很重要，可以理解为一个隔离的虚拟环境，安装在 `virtualenv` 下的包不会影响到整个系统的包，避免不同版本包之间的影响，不需要太深入只需要知道如何创建及激活、退出（`deactivate`命令）就可以。

使用 `virtualenv` 的时候需要注意，当在 `virtualenv` 下安装了包之后，需要先用 `deactivate` 命令退出 `virtualenv` 后再重新激活 `virtualenv` 才可以用这个包。

打开桌面上的终端后，依次输入下面的命令建立学习环境（\$ 是 shell 提示符，之后的内容才是真正需要输入的命令）：

```
$ cd ~/Code
$ sudo pip install virtualenv
$ virtualenv -p /usr/bin/python3.5 env
$ source env/bin/activate
$ pip install sqlalchemy ipython mysqlclient
$ deactivate
```

以上命令在 `Code` 目录创建一个 `virtualenv` 环境，接着在这个虚拟环境中安装了本节实验需要的软件包，比如 `sqlalchemy` 和 `ipython`，然后使用 `deactivate` 退出 `env`。后续的交互式命令都通过 `IPython` 终端输入。可以通过以下命令启动 `IPython` 终端：

```
$ source env/bin/activate
$ ipython
```

`IPython`是一个交互式计算系统，类似于我们之前使用的 `Python 3` 交互式环境，但是它是一个具备更多便利功能的增加的交互式“`Python shell`”。主要优点包括：

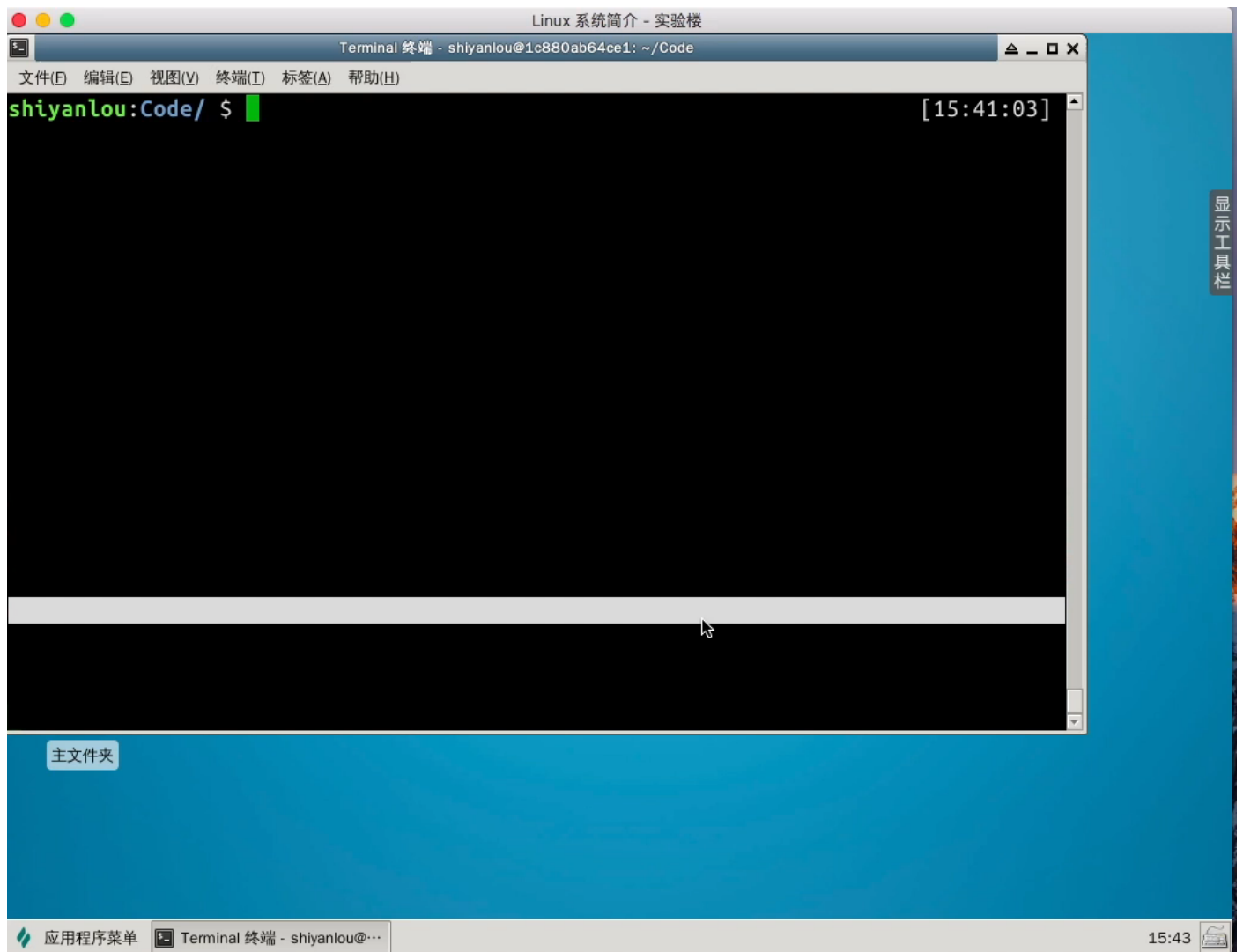
1. 可以非常方便获取代码中对象的各种信息
2. 可以使用叹号直接在其中调用 `Linux` 的命令
3. 支持 `TAB` 自动补全代码
4. 输入的代码都保存到历史记录数据库里

因此，如果你对 `IPython` 熟悉的话，会逐步替代先前使用的 `Python Shell`。

后文出现的代码中，In [1] 类似样式的字符是 IPython 的提示符，不需要输入。

🔗 楼+之Python实战第10期 (/courses/1190)

virtualenv 及 IPython 操作视频：



后续的 SQLAlchemy 基本操作都提供了详细的操作代码和预期的结果，不涉及到难以理解的操作内容，操作过程中需要细心理解 sqlalchemy 模块中每个类及函数的作用。

注意：在 ipython 中连续键入两个回车换行就会进入一个新的输入，这个时候如果你的类还没有写完，那么不要连续键入两次回车。

连接数据库

使用 SQLAlchemy 连接数据库主要通过 Engine 对象进行，在 IPython 终端中输入以下代码：

```
In [1]: from sqlalchemy import create_engine

In [2]: engine = create_engine('mysql://root:@localhost/shiyanlou')

In [3]: engine.execute('select * from user').fetchall()
Out[3]: [(1, 'aiden', 'luojin@simplecloud.cn'), (2, 'lxttx', 'lxttx@gmail.com')]
```

首先导入了 `create_engine`，该方法用于创建 Engine 实例，传递给 `create_engine` 的参数定义了 MySQL 服务器的访问地址，其格式为 `mysql://<user>:<password>@<host>/<db_name>`，例子中访问的正是上文中创建的 `shiyanolou` 数据库。

接着通过 `engine.execute` 方法执行了一条 SQL 语句，查询了 `user` 表中的所有用户，非常容易。

对象关系映射

如果想使 Python 类映射到数据库表中，需要基于 SQLAlchemy 的 declarative base class，也就是声明基类创建类。当基于此基类，创建 Python 类时，就会自动映射到相应的数据库表上。创建声明基类，可以通过 `declarative_base` 方法进行，如下代码：

```
In [12]: from sqlalchemy.ext.declarative import declarative_base

In [13]: Base = declarative_base()
```

创建基类以后，在 IPython 终端中输入下面代码，创建 `User` 类，该类将映射到上文中创建的 `user` 表：

```
In [19]: from sqlalchemy import Column, Integer, String

In [20]: class User(Base):
...:     __tablename__ = 'user'
...:     id = Column(Integer, primary_key=True)
...:     name = Column(String(50))
...:     email = Column(String(50))
...:     def __repr__(self):
...:         return "<User(name=%s)>" % self.name
...:

In [21]:
```

以上代码执行以后就成功定义了 `User` 类，注意 `__repr__` 前后各有两个下划线，这种前后有两个下划线的函数表示一个特殊的函数，称为 Python 类的魔法方法，`__init__` 也是一个魔法方法，这里 `__repr__` 方法会在直接调用实例对象的时候被调用。其中，`__tablename__` 对应着表名。

此时 `User` 有一个 `__table__` 属性，记录了定义的表信息，该属性如下所示：

```
In [27]: User.__table__
Out[27]: Table('user', MetaData(bind=None), Column('id', Integer(), table=<user>, primary_key=True, nullable=False), Column('name', String(), table=<user>), Column('email', String(), table=<user>), schema=None)
```

如果想通过 `User` 查询数据库该怎么办呢？需要先引入 `Session`。`Session` 是映射类和数据库沟通的桥梁，包含事务管理功能。通过以下代码创建 `Session`：

🔍 楼+之Python实战第10期 (courses/1190)

```
In [31]: Session = sessionmaker(bind=engine)
```

```
In [32]: session = Session()
```

这里的代码先从 sqlalchemy.orm 中导入 sessionmaker，然后创建了 sessionmaker 对象 Session，其中 Session 对象中有一个魔法方法（__call__），这个魔法方法让 Session 对象可以像函数那样调用，从而使用 Session() 获得了 session 对象。

Session 创建成功以后，就可以查询用户了，主要通过 session.query 方法：

```
In [63]: session.query(User).all()
Out[63]: [<User(name=aiden)>, <User(name=lxctx)>]

In [65]: session.query(User).filter(User.name=='aiden').first()
Out[65]: <User(name=aiden)>
```

可以看到查询成功，而且可以直接使用 User 类的字段进行过滤查询。

如果在上文中创建 Engine 时，通过 echo 参数开启了显示 SQL 语句（engine = create_engine('mysql://root:@localhost/shiyanlou', echo=True)），则使用 session 查询时，可以看到相应的 SQL 输出：

```
In [69]: session.query(User).filter(User.name=='aiden').first()
2017-08-30 18:02:37,878 INFO sqlalchemy.engine.base.Engine SHOW VARIABLES LIKE 'sql_mode'
2017-08-30 18:02:37,878 INFO sqlalchemy.engine.base.Engine ()
2017-08-30 18:02:37,883 INFO sqlalchemy.engine.base.Engine SELECT DATABASE()
2017-08-30 18:02:37,883 INFO sqlalchemy.engine.base.Engine ()
2017-08-30 18:02:37,885 INFO sqlalchemy.engine.base.Engine show collation where `Charset` =
'utf8' and `Collation` = 'utf8_bin'
2017-08-30 18:02:37,885 INFO sqlalchemy.engine.base.Engine ()
2017-08-30 18:02:37,889 INFO sqlalchemy.engine.base.Engine SELECT CAST('test plain returns'
AS CHAR(60)) AS anon_1
2017-08-30 18:02:37,889 INFO sqlalchemy.engine.base.Engine ()
2017-08-30 18:02:37,893 INFO sqlalchemy.engine.base.Engine SELECT CAST('test unicode return
s' AS CHAR(60)) AS anon_1
2017-08-30 18:02:37,893 INFO sqlalchemy.engine.base.Engine ()
2017-08-30 18:02:37,895 INFO sqlalchemy.engine.base.Engine SELECT CAST('test collated return
s' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin AS anon_1
2017-08-30 18:02:37,895 INFO sqlalchemy.engine.base.Engine ()
2017-08-30 18:02:37,897 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)
2017-08-30 18:02:37,898 INFO sqlalchemy.engine.base.Engine SELECT user.id AS user_id, user.n
ame AS user_name, user.email AS user_email
FROM user
WHERE user.name = %s
LIMIT %s
2017-08-30 18:02:37,899 INFO sqlalchemy.engine.base.Engine ('aiden', 1)
Out[69]: <User(name=aiden)>
```

在上面的查询过程中我们使用到了 filter 对数据进行筛选。filter 中的运算符可以填入 SQL WHERE 子句中的运算符（像 =, !=, >, < 等），或者是 AND, OR 等运算符。另外，这里的 filter 也可以更换为 filter_by 函数，后者相对于前者没有那么灵活，一般会推荐使用 filter。

关于 SQL 语句，可以参考：

- 拓展阅读 《W3school 教程 - SQL WHERE 子句》
(http://www.w3school.com.cn/sql/sql_where.asp)

创建数据库表

可以基于 SQLAlchemy 定义的类中生成数据库表，下面尝试创建一个新的实验表 lab，一个课程将对应于多个实验。所以课程表 course 和 lab 表是 1:M 一对多的关系。在定义 Lab 类之前，需要先将创建 Course 类，使其映射到之前定义的 course 表：

```
In [80]: from sqlalchemy.orm import relationship

In [81]: from sqlalchemy import ForeignKey

In [84]: class Course(Base):
...:     __tablename__ = 'course'
...:     id = Column(Integer, primary_key=True)
...:     name = Column(String(50))
...:     teacher_id = Column(Integer, ForeignKey('user.id'))
...:     teacher = relationship('User')
...:     def __repr__(self):
...:         return '<Course(name=%s)>' % self.name
...:
```

上面的代码引入了一些新的东西。前文中创建的 course 表有外键 teacher_id，在 SQLAlchemy 中可以使用 ForeignKey 设置外键。设置外键后，如果能够直接从 Course 的实例上访问到相应的 user 表中的记录会非常方便，而这可以通过 relationship 实现。上面的代码通过 relationship 定义了 teacher 属性，这样就可以直接通过 course.teacher 获取相应的用户记录。Course 类定义后，接着定义 Lab 类：

```
In [89]: class Lab(Base):
...:     __tablename__ = 'lab'
...:     id = Column(Integer, primary_key=True)
...:     name = Column(String(64))
...:     course_id = Column(Integer, ForeignKey('course.id'))
...:     course = relationship('Course', backref='labs')
...:     def __repr__(self):
...:         return '<Lab(name=%s)>' % self.name
...:
```

以上代码定义了 Lab 类，需要注意的地方是定义 course 属性时，使用了 relationship 的 backref 参数，该参数使得可以在 Course 实例中，通过 course.labs 访问关联的所有实验记录。

定义 Lab 类以后，就可以通过以下命令在 MySQL 中创建相应的 lab 表：

🔗 楼+之Python实战第10期 (/courses/1190)

```
In [90]: Base.metadata.create_all(engine)
```

然后通过 MySQL 客户端可以看到，相应的表已经创建成功：

```
mysql> show tables;
+-----+
| Tables_in_shiyanlou |
+-----+
| course               |
| lab                  |
| user                 |
+-----+
3 rows in set (0.00 sec)

mysql> describe lab;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id         | int(11)       | NO   | PRI | NULL    | auto_increment |
| name      | varchar(64)   | YES  |     | NULL    |                |
| course_id | int(11)       | YES  | MUL | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

练习题：SQLAlchemy 创建新表

在 shiyanlou 数据库中创建新的一个新的表 path，用来存储路径信息，需要满足以下需求：

1. 包含 id，name，config 字段
2. 主键为 id，设置为自增
3. name 为必填
4. config 为字符串（企业应用中通常为 JSON 或 YAML 字符串），可以按照字符串（varchar（128））来进行存储

继续在之前的 ipython 中使用 SQLAlchemy 来创建 Path 类，对应到 MySQL 中的 path 表，表创建之后，点击 下一步，系统将自动检测完成结果。

简单的 CRUD 操作

基于前面创建的 User，Course 和 Lab 类，我们进一步学习创建，查询，更新，删除操作。有了 ORM 映射以后了，创建数据库记录就非常简单，比如想创建一个实验，该实验关联到前文插入的课程上，只需要创建 Lab 的实例就可以了：

楼+之Python实战第10期 (courses/1190)

```
In [37]: course = Course(name='ORM 基础', course_id=2)

In [38]: lab1 = Lab(name='ORM 基础', course_id=course.id)

In [39]: lab2 = Lab(name='关系数据库', course=course)

In [41]: session.add(lab1)

In [42]: session.add(lab2)

In [43]: session.commit()

In [44]: course.labs
Out[44]: [<Lab(name=关系数据库)>, <Lab(name=ORM 基础)>]
```

上面的代码中，首先查询出需要管理的课程对象，接着创建了两个实验，并通过 `session.commit()` 操作提交到了数据库中，在将数据变更提交到数据库中前，需要将数据通过 `session.add` 方法添加到 `session` 中。还可以看到，创建实验时，关联到课程有两种办法，一种是直接赋值给 `course_id`，另外一种办法是赋值给定义的关系属性 `course`。当数据成功插入数据库后，就可以通过 `course.labs` 获取这两个实验。

MySQL 客户端中可以查询出 `lab` 表中成功的插入了两条数据：

```
mysql> select * from lab;
+----+-----+-----+
| id | name      | course_id |
+----+-----+-----+
|  2 | 关系数据库 |          2 |
|  3 | ORM 基础  |          2 |
+----+-----+-----+
2 rows in set (0.00 sec)
```

更新操作也非常简单，只需要更新对象的属性，然后通过 `session.commit()` 提交到数据库即可，下面的代码演示了如何更新课程名称：

🔍 楼之Python实战第10期 (/courses/1190)
In [58]: lab1.course
Out[58]: 'Python 基础'

```
In [59]: lab1.course
Out[59]: <Course(name=Python 基础)>

In [60]: course.name = 'Python 数据分析'

In [61]: session.add(course)

In [62]: session.commit()

In [63]: lab1.course
Out[63]: <Course(name=Python 数据分析)>

In [64]: session.query(Course).all()
Out[64]: [<Course(name=Python 数据分析)>]
```

删除数据时通过 `session.delete` 方法删除相应的对象即可：

```
In [65]: session.delete(lab1)

In [66]: session.commit()

In [67]: course.labs
Out[67]: [<Lab(name=关系数据库)>]
```

删除记录以后，通过课程实例获取到的实验数量同样减少。

可以看到有了 SQLAlchemy 以后，对于数据库的 CRUD 操作全部变成了相应的 Python 对象操作，非常方便。

练习题：SQLAlchemy 增加数据到 Path 表

向之前创建的 `path` 表中增加技术路径信息，需要增加以下两个路径（也就是增加两条数据到数据表中）：

1. 路径名称：Python，配置信息 {'description': 'Python path'}
2. 路径名称：BigData，配置信息 {'description': 'BigData path'}

继续在之前的 ipython 中使用 SQLAlchemy 的 Path 类向 `path` 表中添加这两个路径数据，然后点击 下一步，系统将自动检测完成结果。

1:1 和 M:M 的关系建立

在关系数据库中，关系类型有三种：

- 1:1, 两张表中的数据项是一一对应的关系, 假设用户只有一项附属信息, 这个时候用户表和附属信息表就是一对一的关系;
- 1:M, 比如上文中课程和实验的关系, 一个课程对应多个实验;
- M:M, 多对多的关系, 比如课程和标签的关系, 一个课程有多个标签, 而一个标签可以关联到多个课程;

我们已经知道 1:M 的关系创建方法, 主要通过外键进行。其实 1:1 的关系也可以通过外键进行创建。通过以下代码创建 UserInfo 用户附属信息表:

```
In [86]: class UserInfo(Base):
...:     __tablename__ = 'userinfo'
...:     user_id = Column(Integer, ForeignKey('user.id'), primary_key=True)
...:     addr = Column(String(512))
...:

In [87]: Base.metadata.create_all(engine)
```

以上代码中, 成功创建了 UserInfo 表, 它和 User 表的关系就是 1:1 的关系, 因为 UserInfo 表的主键和外键都是 user_id, 且依赖于 User 表的主键。

对于 M:M 多对多的关系, 可以由两个 1:M 的关系构造出来。如果两张表对于同一张表 (假如表名是 T) 都是 1:M 的关系, 那么就可以把 T 当做中间表, 创建出两张表的多对多关系。比如课程表和标签表的关系, 可以由以下代码创建:

```
In [90]: from sqlalchemy import Table, Text

In [91]: course_tag = Table('course_tag', Base.metadata,
...:     Column('course_id', ForeignKey('course.id'), primary_key=True),
...:     Column('tag_id', ForeignKey('tag.id'), primary_key=True)
...: )

In [92]: class Tag(Base):
...:     __tablename__ = 'tag'
...:     id = Column(Integer, primary_key=True)
...:     name = Column(String(64))
...:     courses = relationship('Course',
...:                             secondary=course_tag,
...:                             backref='tags')
...:     def __repr__(self):
...:         return '<Tag(name=%s)>' % self.name

In [93]: Base.metadata.create_all(engine)
```

1. Base.metadata 是 sqlalchemy.schema.MetaData 对象, 表示所有 Table 对象集合, create_all() 会触发 CREATE TABLE 语句创建所有的表。
2. course_tag 是双主键, 双主键的目的就是为了约束避免出现重复的一对主键记录, 大部分情况都是应用在这种多对多的中间表中。
3. secondary 指的是中间表, backref 指向自己的这个表。

4. session 内部的实现都是调用 engine 的各种接口，相当于 session 是 engine 的一个封装，比如 session.commit 的时候会先调用 engine.connect() 去连接数据库，再调用执行 sql 相关的接口。

上面的代码中，首先通过 Table 类创建了一张中间表 course_tag，因为这张表不需要映射到任何 Python 对象上，所以直接创建了表。可以看到 Course，Tag 表和 course_tag 表都是 1:M 的关系。通过 course_tag 这张中间表成功的建立了 M:M 的关系。还需要注意的是，Tag 表中定义了 courses 属性，该属性通过 relationship 函数的 secondary 参数告诉 SQLAlchemy 通过表 course_tag 关联到 Course 表的对象上。下面演示下多对多关系的 CRUD 操作：

```
In [126]: session.close()
In [130]: course = session.query(Course).first()

In [131]: course.tags
Out[131]: []

In [132]: tag1 = Tag(name='tag_1')

In [133]: tag2 = Tag(name='tag_2')

In [136]: course.tags.append(tag1)

In [137]: course.tags.append(tag2)

In [138]: session.add(course)

In [139]: session.commit()

In [140]: course.tags
Out[140]: [<Tag(name=tag_1)>, <Tag(name=tag_2)>]

In [141]: engine.execute('select * from tag').fetchall()
Out[141]: [(1, 'tag_1'), (2, 'tag_2')]

In [142]: engine.execute('select * from course_tag').fetchall()
Out[142]: [(2, 1), (2, 2)]
```

上面的代码中，通过 course.tags.append(tag1) 向课程中添加标签，就像操作 Python 列表一样方便，最后还通过 engine.execute 执行 SQL 查询验证了标签确实插入了数据库中，而且 course_tag 表中也生成了相应的关系记录。下面尝试向 Tag 对象中添加课程：

```
In [153]: teacher = User.query.filter(User.name=='aiden').first()
楼+之Python实战第10期 (courses/1190)

In [154]: course1 = Course(name='Linux 基础', teacher=teacher)

In [155]: session.add(course1)

In [156]: session.commit()

In [157]: tag1.courses
Out[157]: [<Course(name=Python 数据分析)>]

In [159]: tag1.courses.append(course1)

In [160]: session.add(tag1)

In [161]: session.commit()

In [163]: tag1.courses
Out[163]: [<Course(name=Python 数据分析)>, <Course(name=Linux 基础)>]

In [164]: engine.execute('select * from course_tag').fetchall()
Out[164]: [(2, 1), (3, 1), (2, 2)]
```

首先创建了另外一门课程 `course1`，然后成功的将 `course1` 添加到了标签 `tag1` 中。

相信通过以上例子，你已经基本学会了 SQLAlchemy 中创建各种关系的方法。

- 拓展阅读：SQLAlchemy 官方文档（英文）(<https://docs.sqlalchemy.org/en/latest/>)
- 拓展阅读：SQLAlchemy 查询语句 (<https://www.jianshu.com/p/196b7892cf38>)

总结

本节实验主要讲解了 MySQL 和 SQLAlchemy 的基础知识，其中 MySQL 主要包含以下知识点：

- 数据库，表的创建方式；
- 基本的 CRUD 操作；
- 基本约束的建立方式；

SQLAlchemy 主要包含以下知识点：

- 连接数据库的方式；
- 映射类（表）的定义方式；
- 通过 Python 对象进行 CRUD 操作；
- 常用关系的定义方式；

本节实验涉及的代码比较多，但是请务必都实际写一遍，这样才能真正掌握本节内容。

**本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

🔍 上一节之挑战：Python 实战第19期资讯网站 (/courses/1190/labs/8535/document)

下一节：挑战：从数据库中读取内容 (/courses/1190/labs/8537/document)