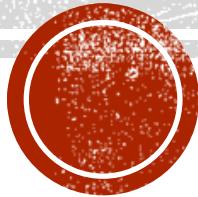


DJANGO QUERYSET



SQL



WHAT IS SQL?

- **SQL stands for Structured Query Language.**
- **SQL lets you access and manipulate databases.**
- **SQL is an ANSI(American National Standards Institute) standard.**



PYAYING WITH DBSHELL

```
# cd to the location where you store your project  
# activate the env  
(env) $ python manage.py dbshell
```

- This chapter uses sqlite3 as the database.



SELECT

- The **SELECT** statement is used to select data from a database.

```
sqlite> select id, name from books_author;
id          name
-----  -----
1           Scott Chacon
2           Zed A. Shaw

sqlite> select * from books_author;
```



WHERE

```
SQLite> select * from books_publisher where id=1;  
sqlite> select * from books_author where  
name='Zed A. Shaw';
```

- **SQL requires single quotes around text values (most database systems will also allow double quotes).**
- **However, numeric fields should not be enclosed in quotes**



INSERT

- The **INSERT INTO** statement is used to insert new records in a table.

```
sqlite> insert into books_author (name) values  
( 'Jeff Forcier' );
```

```
sqlite> select id, name from books_author;  
id          name  
-----  
1           Scott Chacon  
2           Zed A. Shaw  
3           Jeff Forcier
```



UPDATE

- The UPDATE statement is used to modify the existing records in a table.

```
sqlite> update books_author set intro='He is  
professionally specializing in Python and Ruby  
development' where id=3;
```



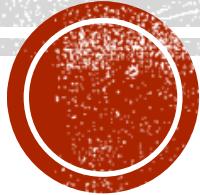
DELETE

- **The DELETE statement is used to delete existing records in a table.**

```
sqlite> delete from books_author where id=3;
```



QUERYSET



INTRODUCTION

- **Once you are creating your data models, Django automatically gives you a database-abstraction API that let you create, retrieve, update and delete objects.**



PLAYING WITH DJANGO SHELL

- **Open shell**
 - **Go to the direction of your project**
 - **Source to env**
 - **python manage.py shell**



RETRIEVING OBJECTS

- A **QuerySet** represents a collection of objects from your database. It can have zero, one or many *filters*. Filters narrow down the query results based on the given parameters.
- In SQL terms, a **QuerySet** equates to a **SELECT** statement, and a filter is a limiting clause such as **WHERE** or **LIMIT**.



RETRIEVING ALL OBJECTS

- **all()**
 - Returns a *copy* of the current QuerySet .

```
>>> from books.models import Author  
  
>>> authors = Author.objects.all()  
  
<QuerySet [<Author: Scott Chacon>, <Author: Zed A.  
Shaw>, <Author: Jeff Forcier>]>
```

- Translates into the following SQL:
 - **SELECT * FROM books_author;**



RETRIEVING ALL OBJECTS

- Print author id and name

```
>>> for author in authors:  
...     print(author.id, author.name)  
...  
1 Scott Chacon  
2 Zed A. Shaw  
3 Jeff Forcierv
```



RETRIEVING SPECIFIC OBJECTS

- **filter(**kwargs)**
 - Returns a new QuerySet containing objects that match the given lookup parameters.

```
>>> books = Book.objects.filter(pub_year=2013)
>>> books
<QuerySet [Book: Learn Python the Hard Way]>
```

- Translates into the following SQL:
 - **SELECT * FROM books_book where pub_year=2013;**



RETRIEVING SPECIFIC OBJECTS

```
>>> books = Book.objects.filter(pub_year__gt=2000)
>>> books
<QuerySet [Book: Pro Git, Book: Learn Python the Hard Way]>
```

- Translates into the following SQL:
 - **SELECT * FROM books_book where pub_year>2000;**



FIELD-LOOKUPS

- <https://docs.djangoproject.com/en/1.11/ref/models/querysets/#field-lookups>



FILTER

- Multiple parameters are joined via AND

```
>>> books = Book.objects.filter(pub_year__gt=2000,  
name__icontains='git')  
  
>>> books  
<QuerySet [Book: Pro Git]>
```

- Translates into the following SQL:
 - **SELECT * FROM books_book WHERE pub_year>2000 and name like '%git%';**



EXCLUDE

- **exclude(**kwargs)**

- Returns a new QuerySet containing objects that do *not* match the given lookup parameters.

```
>>> books = Book.objects.exclude(pub_year=2009)  
>>> books  
<QuerySet [Book: Learn Python the Hard Way]>
```



RETRIEVING A SINGLE OBJECT

- **get(**kwargs)**
 - Return the object matching the given lookup parameters.

```
>>> book = Book.objects.get(id=1)  
>>> book  
<Book: Pro Git>
```

- Translates into the following SQL:
 - **SELECT * FROM books_book WHERE id=1;**



RETRIEVING A SINGLE OBJECT

- Print book object

```
>>> book = Book.objects.get(id=1)
>>> print(book.id, book.name, book.pub_year)
1 Pro Git 2009
```



RETRIEVING A SINGLE OBJECT

- **get()** raises a **DoesNotExist** exception if an object wasn't found for the given parameters.
 - Try: `book = Book.objects.get(id=5555)`
- **get()** raises **MultipleObjectsReturned** if more than one object was found.



CREATING OBJECTS

- To create an object, instantiate it using keyword arguments to the model class, then call `save()` to save it to the database.
- Let's create a book object



CREATING OBJECTS

```
from books.models import Book, Author, Category,  
Publisher  
  
category = Category.objects.get(id=1)  
author = Author.objects.get(name='Jeff Forcier')  
publisher = Publisher.objects.filter(name='Addison-  
Wesley Professional').first()  
  
book = Book(name='Python Web Development with Django',  
            pub_year=2008,  
            serial_num='T0003',  
            available=True,  
            author=author,  
            category=category,  
            publisher=publisher)  
book.save()
```

CREATING OBJECTS

```
book = Book.objects.create(  
    name='Python Web Development with Django',  
    pub_year=2008,  
    serial_num='T0003',  
    available=True,  
    author=author,  
    category=category,  
    publisher=publisher)
```



CREATING A USER

```
from accounts.models import User  
  
user = User(username='lisa',  
             nickname='Lisa',  
             is_superuser=True)  
  
user.set_password('admin1234')  
  
user.save()
```

- **set_password() is used to hashing password**



CREATING A USER

- The password in database in encryption

	id	password	last_login	is_superuser	username
1	1	pbkdf2_sha256\$36000\$WB3D1...	2017-08-14 09:11:49.815887	1	admin
2	2	pbkdf2_sha256\$36000\$Wt3Ek...	<null>	1	lisa



CREATING A USER

create_user:

```
user = User.objects.create_user(  
    username='nancy',  
    nickname='Nancy',  
    is_superuser=True,  
    password='admin1234'  
)
```



UPDATE OBJECT

- How to update a field of a object?

```
>>> user = User.objects.get(id=2)
>>> user
<User: lisa>
>>> user.nickname
'Lisa'
>>> user.nickname='Lisa Wu'
>>> user.save()
>>> user.nickname
'Lisa Wu'
```



DELETING OBJECTS

- **The delete method is named `delete()`. This method immediately deletes the object and returns the number of objects deleted and a dictionary with the number of deletions per object type.**



DELETING OBJECTS

```
>>> User.objects.get(id=3).delete()  
(1, {'admin.LogEntry': 0, 'accounts.User_groups':  
0, 'accounts.User_user_permissions': 0,  
'accounts.User': 1})  
  
>>> User.objects.filter(nickname='Nancy').delete()
```



REFERENCE

- [https://docs.djangoproject.com/en/1.11/topics/
db/queries/](https://docs.djangoproject.com/en/1.11/topics/db/queries/)





Questions?

