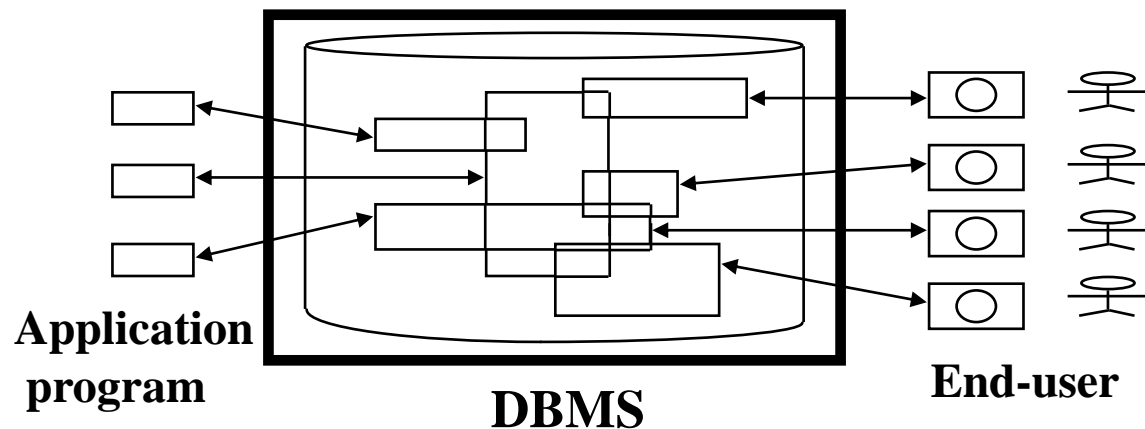


Object to Relational Mapping: Django ORM

Dr. Shaheen Khatoon



How to install Django

- ❑ Separate Virtual Environment
- ❑ **Globally**

Django Requirements

- ❑ Python 3.0 or Higher

`python --version`

- ❑ PIP

`pip --version`

- ❑ Text/Code Editor/IDE – Notepad++, **VS Code**, ATOM, PyCharm

- ❑ Web Browser – Google Chrome, Mozilla Firefox, Edge

Check Django is installed or Not

django-admin --version

How to Install Django

pip install django – This is used to install Django.

For Version Specific

pip install django==2.0

How to Uninstall Django

pip uninstall django

Django Project

A Django Project may contain multiple Project Application, which means a group of Application and files is called as Django Project.

An **Application** is a Part of **Django Project**.

SchoolProject

- Registration App
- Fees App
- Exam App
- Attendance App
- Result App

Create Django Project

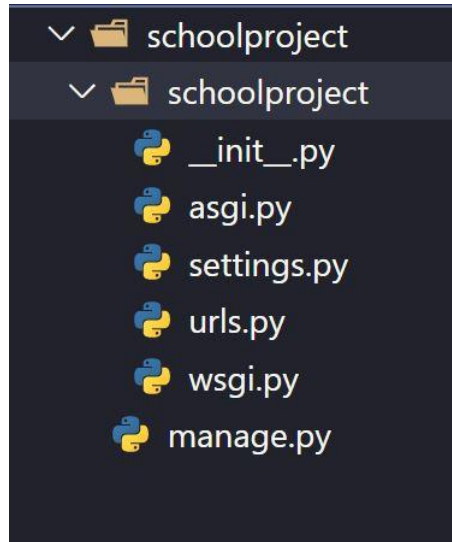
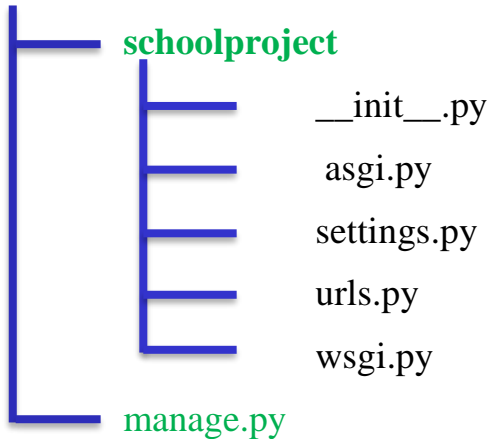
Syntax: -

*django-admin startproject **projectname***

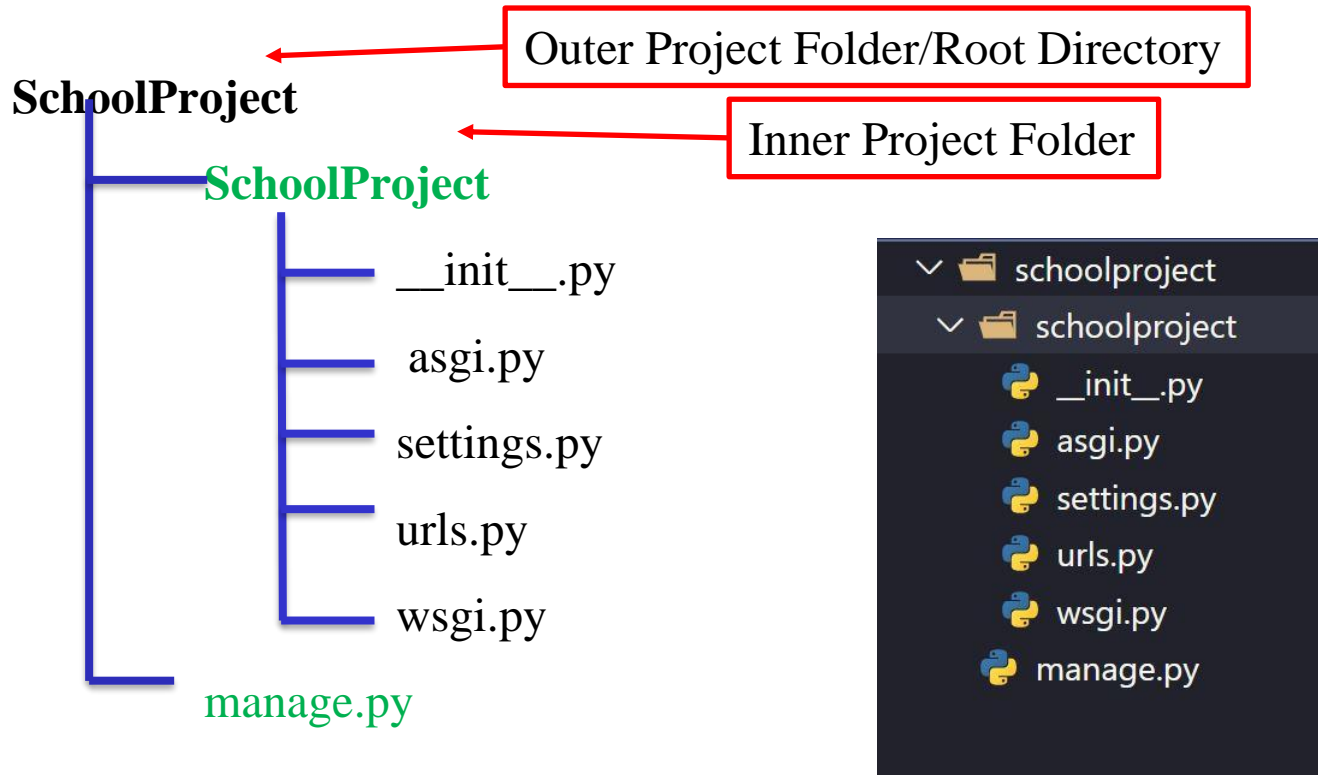
Example:-

*django-admin startproject **schoolproject***

schoolproject



Django Project Directory Structure



Django Project Directory Structure

__init__.py – The folder which contains __init__.py file is considered as python package.

wsgi.py – WSGI (Web Server Gateway Interface) is a specification that describes how a web server communicates with web applications, and how web applications can be chained together to process one request. WSGI provided a standard for synchronous Python apps.

asgi.py – ASGI (Asynchronous Server Gateway Interface) is a spiritual successor to WSGI, intended to provide a standard interface between async-capable Python web servers, frameworks, and applications. ASGI provides standard for both asynchronous and synchronous apps (**Provide interface for web application and web server**)

settings.py – This file contains all the information or data about project settings.

E.g.:- Database Config information, Template, Installed Application, Validators etc.

urls.py – This file contains information of url attached with application.

manage.py – manage.py is automatically created in each Django project. It is Django's command-line utility also sets the DJANGO_SETTINGS_MODULE environment variable so that it points to your project's settings.py file. Generally, when working on a single Django project, it's easier to use manage.py than django-admin.

How to run server

Django provides built-in server which we can use to run our project.

runserver – This command is used to run built-in server of Django.

Steps:-

- ❑ Go to Project Folder
- ❑ Then run command *python manage.py runserver*
- ❑ Server Started
- ❑ Visit *http://127.0.0.1:8000* or *http://localhost:8000*

- ❑ You can specify Port number *python manage.py runserver 5555*
- ❑ Visit *http://127.0.0.1:5555* or *http://localhost:5555*

How to Stop server

ctrl + *c* is used to stop Server

Note – Sometime when you make changes in your project you may need to restart the server.

How to Start/Create Application

A Django project contains one or more applications which means we create application inside Project Folder.

Syntax:- `python manage.py startapp appname`

Creating One Application inside Project:-

- ❑ Go to Project Folder
- ❑ Run Command *`python manage.py startapp course`*

Creating Multiple Applications inside Project:-

- ❑ Go to Project Folder
- ❑ *`python manage.py startapp course`*
- ❑ *`python manage.py startapp fees`*
- ❑ *`python manage.py startapp result`*

How to Install Application in Our Project

As we know a Django project can contain multiple application so just creating application inside a project is not enough we also have to install application in our project.

We install application in our project using **settings.py** file.

settings.py file is available at Project Level which means we can install all the application of project.

This is compulsory otherwise Application won't be recognized by Django.

Open **settings.py** file

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'application_name1',  
    'application_name2',  
]
```

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'course',  
    'fees',  
    'result',  
]
```

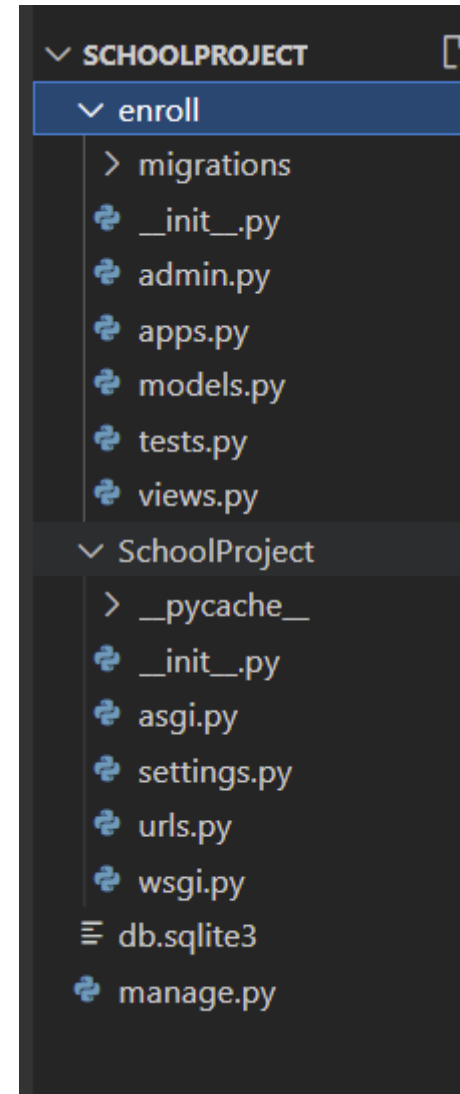
Save **settings.py** File

Steps in our Project

- ❑ Create Django Project: `django-admin startproject SchoolProject`
- ❑ Change Directory to Django Project: `cd SchoolProject`
- ❑ Create Django Application: `python manage.py startapp enroll`
- ❑ Add/Install Application to Django Project (enroll to SchoolProject)
 - Open `settings.py`
 - Add `enroll`

`INSTALLED_APPS`
`['django.contrib.admin', 'course',]`

- Save `settings.py`



Application Directory Structure

- ❑ Go to Project Folder
- ❑ Run Command *python manage.py startapp enroll*

enroll

migrations

__init__.py

__init__.py

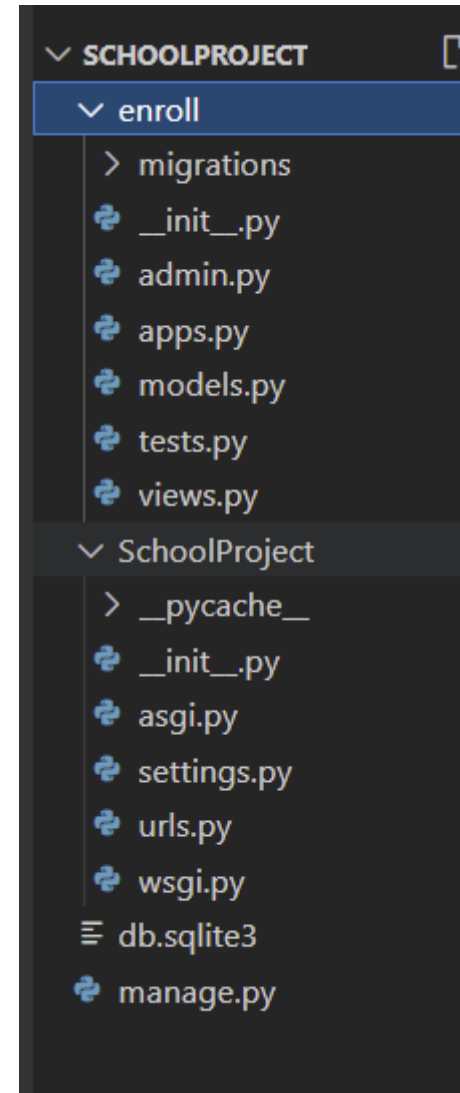
admin.py

apps.py

models.py

tests.py

views.py



Application Directory Structure

migrations – This folder contains `__init__.py` file which means it's a python package. It also contains all files which are created after running **makemigration** command .

__init__.py – The folder which contains `__init__.py` file is considered as Python Package.

admin.py – This file is used to register sql tables so we could perform CRUD operation from Admin Application. Admin Application is provided by Django to perform CRUD operation.

apps.py – This file is used to config app.

models.py – This file is used to create our own **model class** later these classes will be converted into database table by Django for our application.

tests.py – This is files is used to create tests.

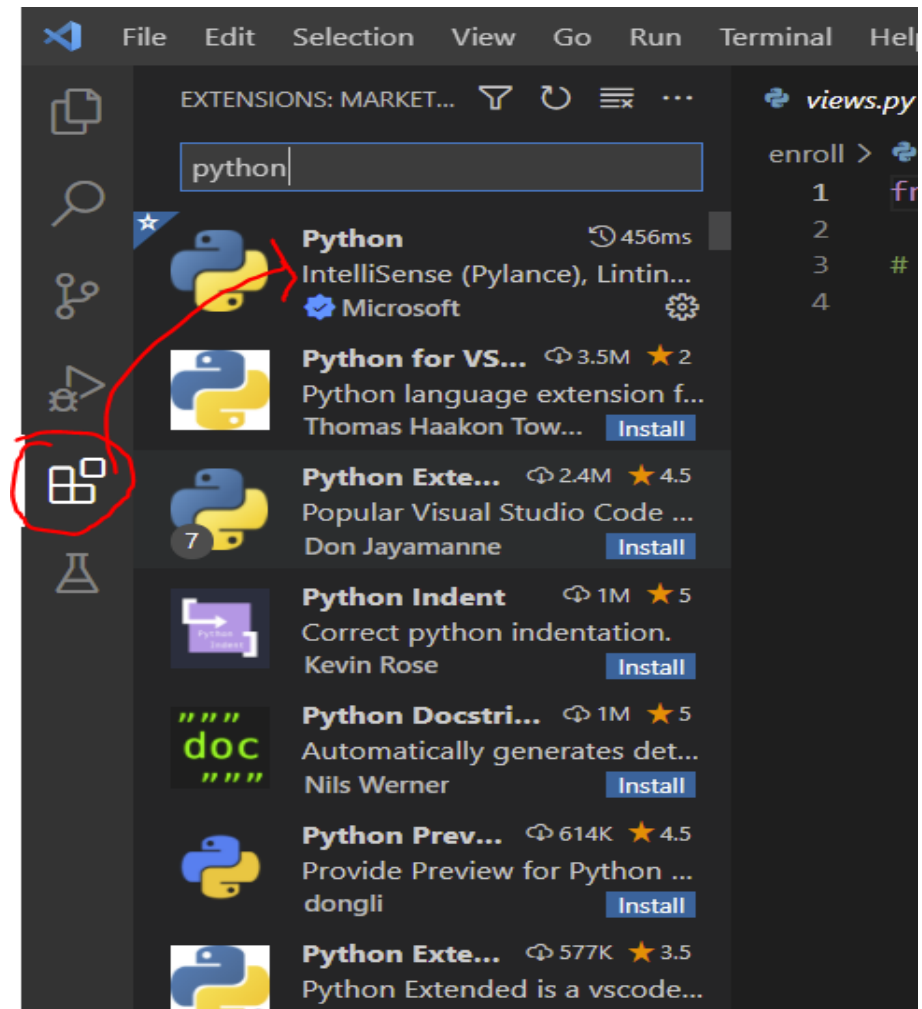
views.py – This file is used to create view. We write all the business logic related code in this file.

Note for Python support in Visual Studio Code (VSC)

- ❑ To get support of Python in VSC please install python package in VSC

Go to package icon at the left panel.

Type Python and click install



Object Relational Mapper (ORM)

Object-Relational Mapper (ORM), which enables application to interact with database such as SQLite, MySQL, PostgreSQL, Oracle.

ORMs automatically create a database schema from defined classes or models. It generate SQL from Python code for a particular database which means developer do not need to write SQL Code.

ORM maps objects attributes to respective table fields.

It is easier to change the database if we use ORMs hence project becomes more portable.

Django's ORM is just a way to create SQL to query and manipulate your database and get results in a pythonic fashion.

ORMs use connectors to connect databases with a web application.


Object Relation Mapper (ORM)



Object Relation Mapper (ORM)

```
class Student(models.Model):  
    stuid=models.IntegerField()  
    stuname=models.CharField(max_  
length=70)  
    stuemail=models.EmailField(max_  
_length=70)  
    stupass=models.CharField(max_l  
ength=70)
```

```
CREATE TABLE "enroll_student" (  
    "id" integer NOT NULL  
    PRIMARY KEY  
    AUTOINCREMENT,  
    "stuid" integer NOT NULL,  
    "stuname" varchar(70) NOT  
NULL,  
    "stuemail" varchar(70) NOT  
NULL,  
    "stupass" varchar(70) NOT  
NULL  
);
```



id	stuid	stuname	stuemail	stupass

QuerySet

A QuerySet can be defined as a list containing all those objects we have created using the Django model.

QuerySets allow you to read the data from the database, filter it and order it.

Model

A model is the single, definitive source of information about your data.

It contains the essential fields and behaviors of the data you're storing.

Generally, each model maps to a single database table.

Model Class

Model class is a class which will represent a table in database.

Each model is a Python class that subclasses `django.db.models.Model`

Each attribute of the model represents a database field.

With all of this, Django gives you an automatically-generated database-access API Django provides built-in database by default that is sqlite database.

We can use other database like MySQL, Oracle SQL etc.

Create Our Own Model Class

models.py file which is inside application folder, is required to create our own model class.

Our own model class will inherit Python's Model Class.

Syntax:-

```
class ClassName(models.Model):  
    field_name=models.FieldType(arg, options)
```

Example:-

models.py

```
class Student(models.Model):
```

```
    stuid=models.IntegerField()
```

```
    stuname=models.CharField(max_length=70)
```

```
    stuemail=models.EmailField(max_length=70)
```

```
    stupass=models.CharField(max_length=70)
```

Length is required in
CharField Type

- This class will create a table with columns and their data types
- Table Name will be ApplicationName_ClassName, in this case it will be enroll_student
- Field name will become table's Column Name, in this case it will be stuid, stuname, stuemail, stupass with their data type.
- As we have not mentioned primary key in any of these columns so it will automatically create a new column named 'id' Data Type Integer with primary key and auto increment.

Create Our Own Model Class

models.py

```
class Student(models.Model):
```

```
    stuid=models.IntegerField()
```

```
    stuname=models.CharField(max_length=70)
```

```
    stuemail=models.EmailField(max_length=70)
```

```
    stupass=models.CharField(max_length=70)
```

```
CREATE TABLE "enroll_student" (
```

```
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT
```

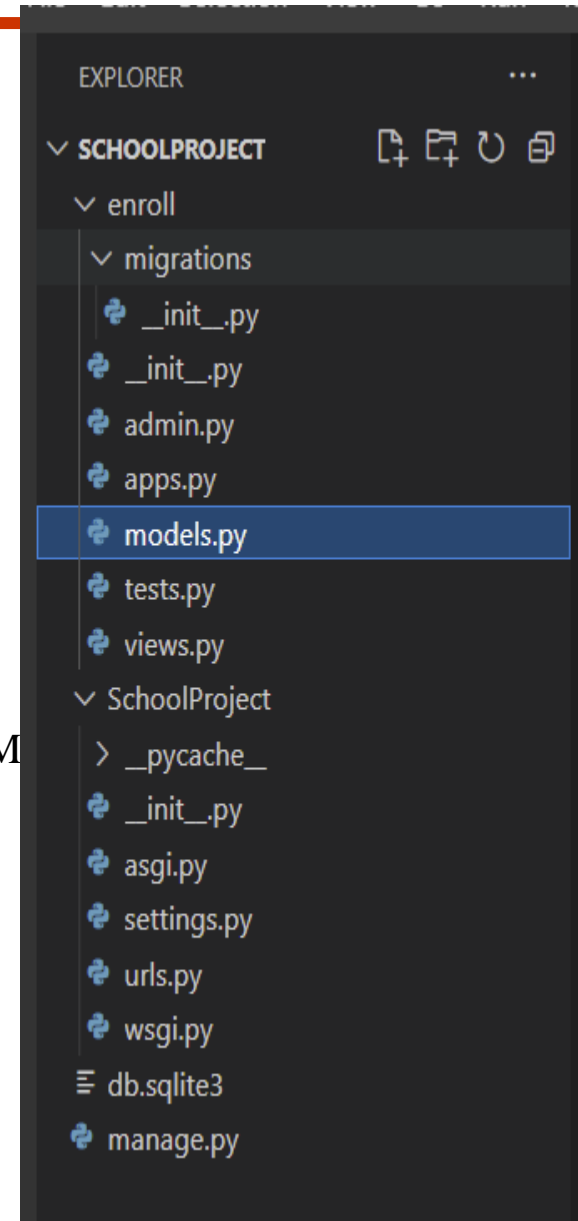
```
    "stuid" integer NOT NULL,
```

```
    "stuname" varchar(70) NOT NULL,
```

```
    "stuemail" varchar(70) NOT NULL,
```

```
    "stupass" varchar(70) NOT NULL
```

```
);
```



Rules

- ❑ Field Name instantiated as a class attribute and represents a particular table's Column name.
- ❑ Field Type is also known as Data Type.
- ❑ A field name cannot be a Python reserved word, because that would result in a Python syntax error.
- ❑ A field name cannot contain more than one underscore in a row, due to the way Django's query lookup syntax works.
- ❑ A field name cannot end with an underscore.

Migrations

Migrations are Django's way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema.

- ❑ **makemigrations** – This is responsible for creating new migrations based on the changes you have made to your models.
- ❑ **migrate** – This is responsible for applying and unapplying migrations.
- ❑ **sqlmigrate** – This displays the SQL statements for a migration.
- ❑ **showmigrations** – This lists a project's migrations and their status.

makemigrations and migrate

- ❑ **makemigrations** is used to convert model class into sql statements. This will also create a file which will contain sql statements. This file is located in Application's migrations folder.

Syntax:- `python manage.py makemigrations`

- ❑ **migrate** is used to execute sql statements generated by makemigrations. This command will execute All Application's (including built-in applications) SQL Statements if available. After execution of sql statements table will be created.

Syntax:- `python manage.py migrate`

Note – If you make any change in your own model class you are required to run makemigrations and migrate command only then you will get those changes in your application.

Display SQL Statement

We can retrieve SQL Statement by using below command:-

Syntax:-

```
python manage.py sqlmigrate application_name dbfile_name
```

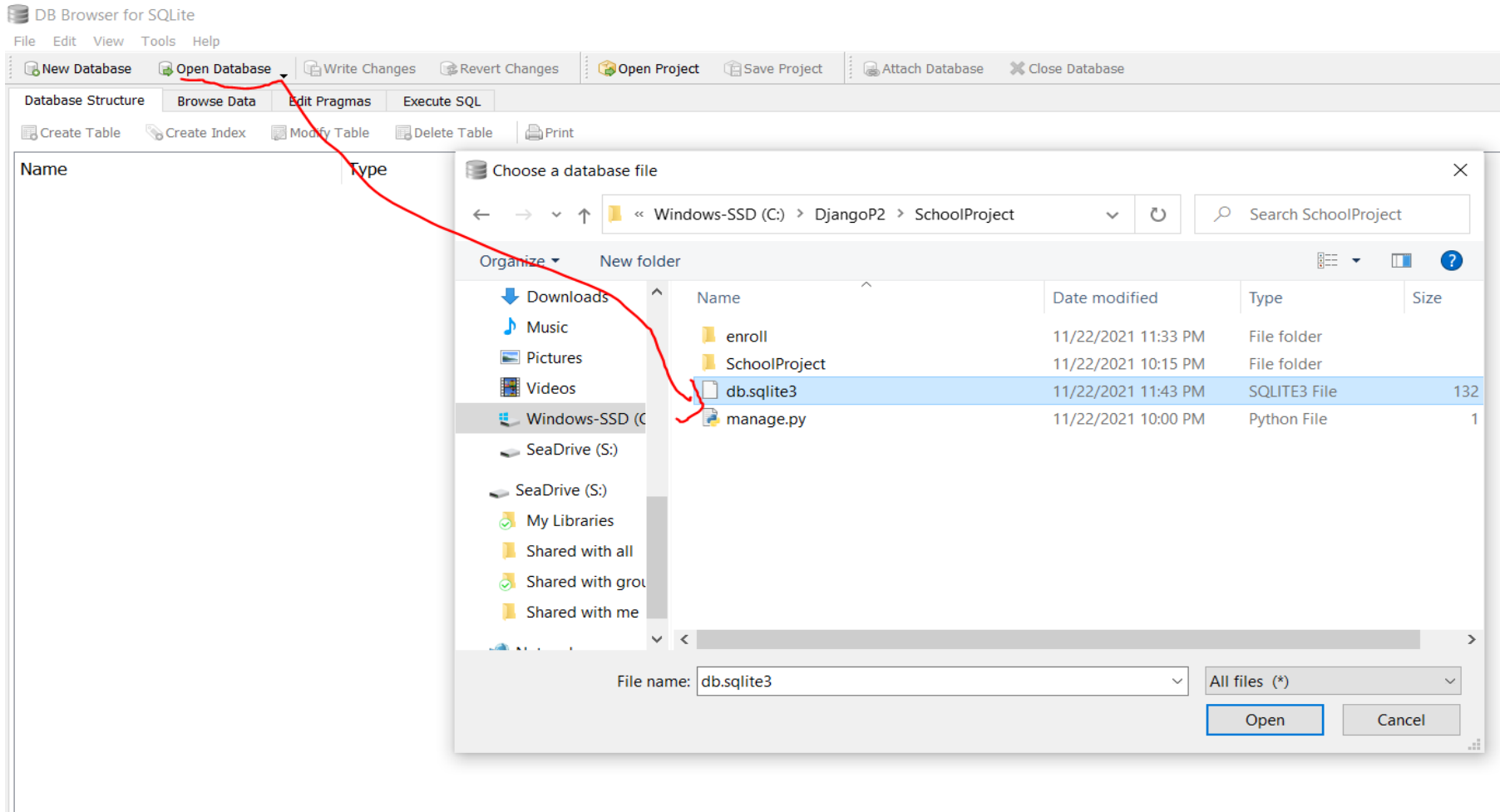
Example:-

```
python manage.py sqlmigrate enroll 0001
```

Note – File name can be found inside Application's migrations folder.

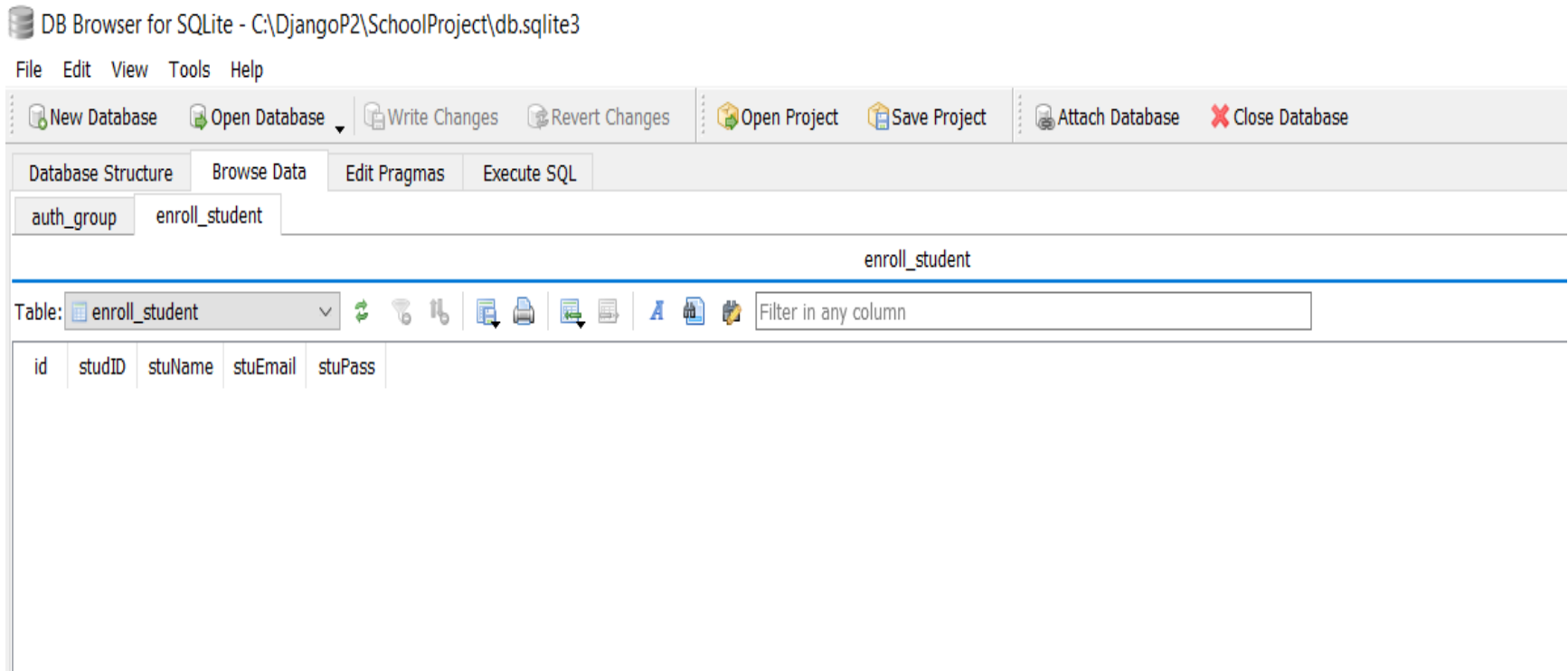
Verify tables in SQLite

- ❑ Open sqlite
- ❑ Open database by locating the Django project folder



Name	Type	Schema
▼ Tables (12)		
> auth_group		CREATE TABLE "auth_group" ("id" integer NOT NULL PRIMARY KEY AUTOINCR
> auth_group_permissions		CREATE TABLE "auth_group_permissions" ("id" integer NOT NULL PRIMARY KE
> auth_permission		CREATE TABLE "auth_permission" ("id" integer NOT NULL PRIMARY KEY AUTO
> auth_user		CREATE TABLE "auth_user" ("id" integer NOT NULL PRIMARY KEY AUTOINCRE
> auth_user_groups		CREATE TABLE "auth_user_groups" ("id" integer NOT NULL PRIMARY KEY AUT
> auth_user_user_permissions		CREATE TABLE "auth_user_user_permissions" ("id" integer NOT NULL PRIMAR
> django_admin_log		CREATE TABLE "django_admin_log" ("id" integer NOT NULL PRIMARY KEY AUT
> django_content_type		CREATE TABLE "django_content_type" ("id" integer NOT NULL PRIMARY KEY A
> django_migrations		CREATE TABLE "django_migrations" ("id" integer NOT NULL PRIMARY KEY AUT
> django_session		CREATE TABLE "django_session" ("session_key" varchar(40) NOT NULL PRIMA
> enroll_student		CREATE TABLE "enroll_student" ("id" integer NOT NULL PRIMARY KEY AUTOIN
> sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
▼ Indices (15)		
> auth_group_permissions_gr...		CREATE INDEX "auth_group_permissions_group_id_b120cbf9" ON "auth_group.
> auth_group_permissions_gr...		CREATE UNIQUE INDEX "auth_group_permissions_group_id_permission_id_0cd
> auth_group_permissions_pe...		CREATE INDEX "auth_group_permissions_permission_id_84c5c92e" ON "auth_c
> auth_permission_content_ty...		CREATE INDEX "auth_permission_content_type_id_2f476e4b" ON "auth_permis
> auth_permission_content_ty...		CREATE UNIQUE INDEX "auth_permission_content_type_id_codename_01ab37
> auth_user_groups_group_id...		CREATE INDEX "auth_user_groups_group_id_97559544" ON "auth_user_groups
> auth user groups user id ...		CREATE INDEX "auth user groups user id 6a12ed8b" ON "auth user groups"

❑ Right click enroll_student table and select browse:



Lab Assignment: ORM for Online Course APP

