Phitron

Outlines

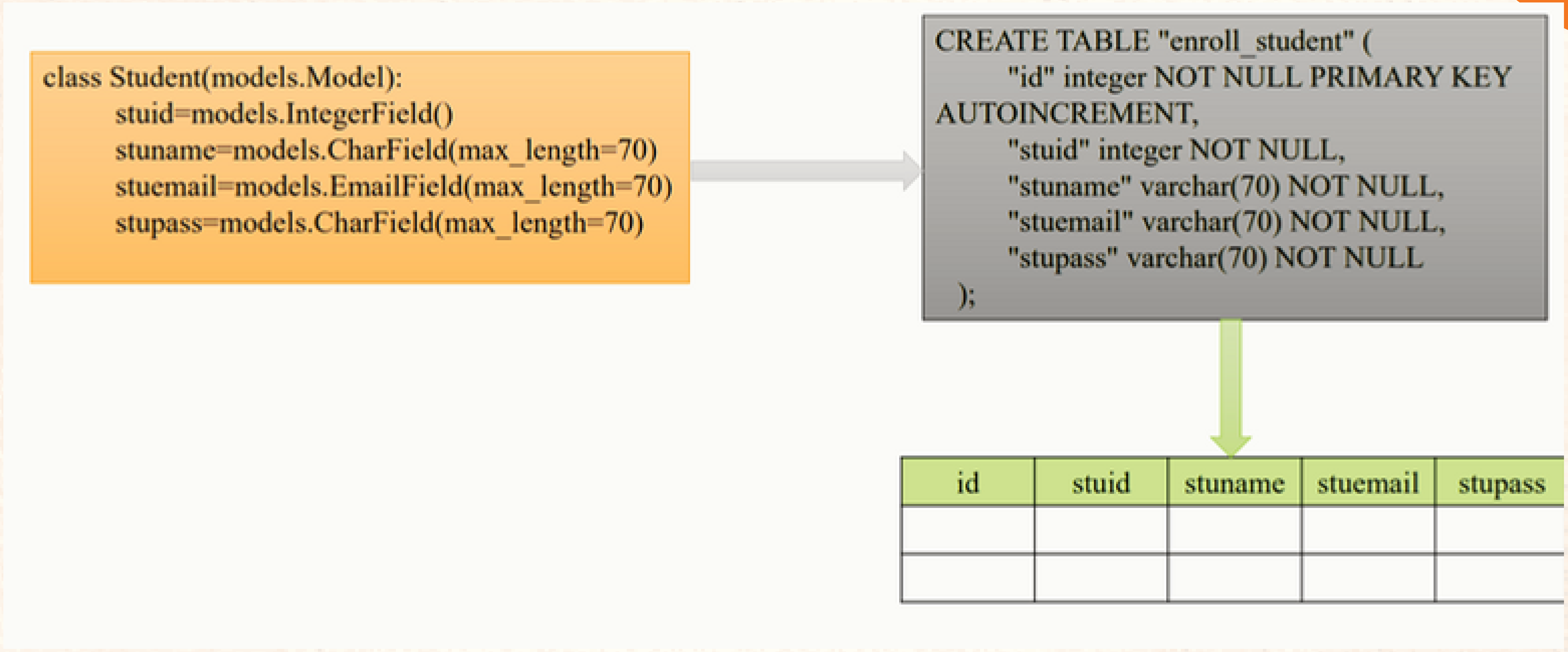
- ORM
- Model

ORM

Object-Relational Mapper is a programming technique that helps application to interact with database such as SQLite, MySQL, PostgreSQL, Oracle.

- create a database schema from defined classes or models.
- generate SQL from Python code for a particular database which means developer do not need to write SQL Code.
- helps to change the database easily
- use connectors to connect databases with a web application.

ORM



```
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
  );
```

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 helps us

- read the data from the database
- filter it
- order it.

Model

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

It contains the

- essential fields and behaviors of the data.
- 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 represents a database field.
- Django gives automatically-generated database-access API
- Django provides sqlite database by default.
- We can use other database like MySQL, Oracle SQL etc.

Model Class

```python
from django.db import models

# Create your models here.

class Movie(models.Model):
    movie_title = models.CharField(max_length=150)
    release_year = models.IntegerField()
    director = models.CharField(max_length=100)
    movie_plot = models.TextField()
```

Phitron

Migrations

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

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

```
python manage.py makemigrations
```

migrate : is used to execute sql statements generated by makemigrations

```
python manage.py migrate
```

showmigrations : This lists a project's migrations

Phitron

Built-in Field Options

null :- contain either True or False. If True, Django will store empty values as NULL in the database. Default is False.

blank :– contain either True or False. If True, the field is allowed to be blank.

Note : *null is purely database-related, whereas blank is validation-related.*

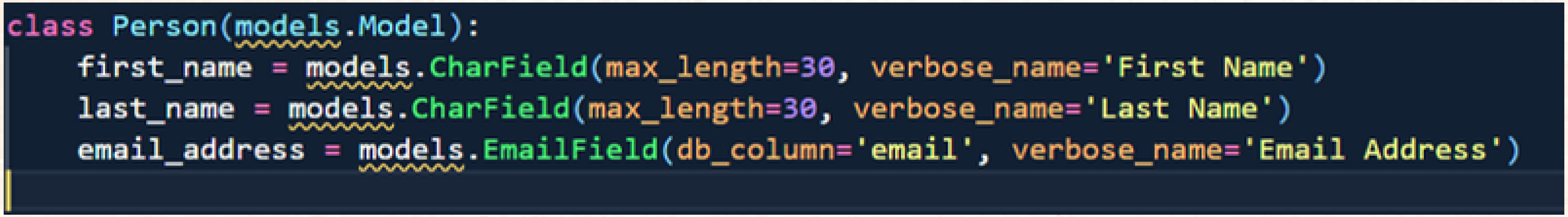
default :- default value for the field.

verbose_name :- A human-readable name for the field. If the verbose name isn’t given, Django will automatically create it using the field’s attribute name, converting underscores to spaces.

Built-in Field Options

db_column :- The name of the database column to use for this field. If this isn't given, Django will use the field's name.

primary_key :– If True, that field will be the primary key for the model.



```
class Person(models.Model):
    first_name = models.CharField(max_length=30, verbose_name='First Name')
    last_name = models.CharField(max_length=30, verbose_name='Last Name')
    email_address = models.EmailField(db_column='email', verbose_name='Email Address')
```

Built-in Field Options

unique :- If True, this field must be unique throughout the table. This is enforced at the database level and by model validation.

Some More fields :

- IntegerField
- AutoField
- FloatField
- TextField
- CharField
- BooleanField
- EmailField
- URLField

Phitron