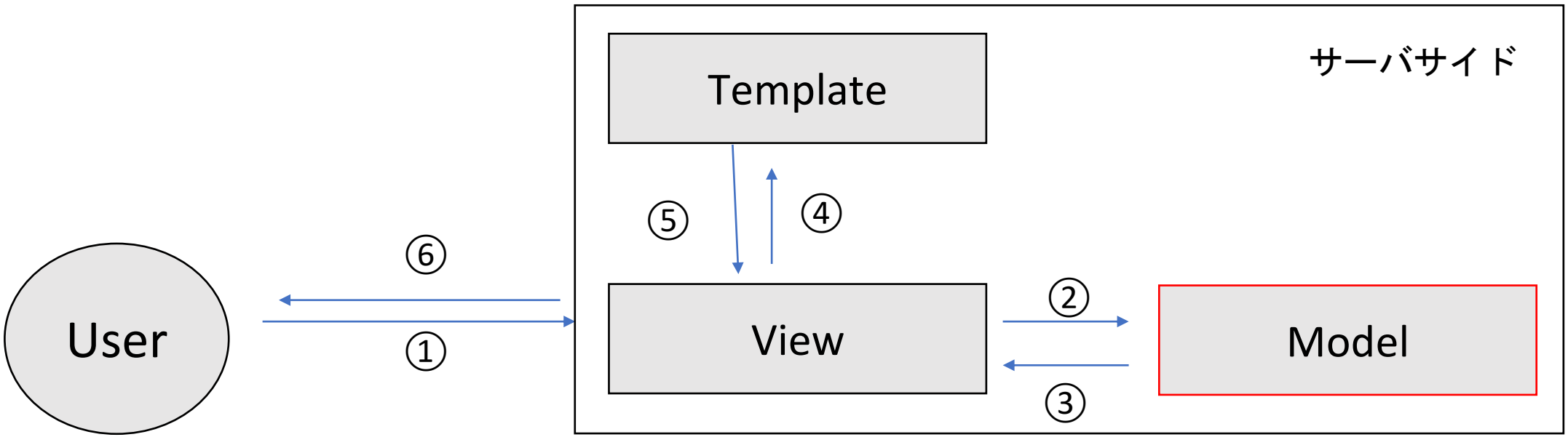
Django基本講座２
(modelの利用)

Modelとは

DBにアクセスして、テーブルを作成したり、データの挿入・更新・取得を行います。
テーブルの型と合わせたプロパティを持って、データの挿入を行えます。



Modelの使用

DBへの接続情報は、settings.py内の**DATABASES**を用います。

アプリケーションを作成すると**models.py**と言うファイルが作成されています。

この中に、モデルの設定を書いて行きます。

・ModelはDjangoのクラス、django.db.models.Modelを継承します。

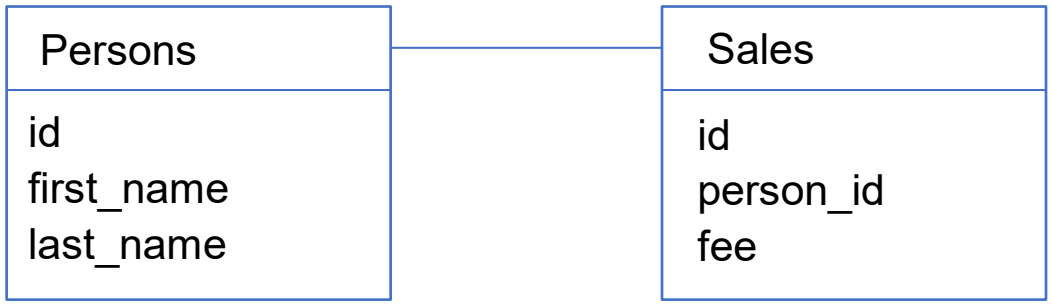
テーブルの定義

```
from django.db import models

class Persons(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)


class Sales(models.Model):
    person = models.ForeignKey(Person, on_delete=models.CASCADE) # 外部キー
    fee = models. IntegerField()
```

以下のテーブルが自動的に作成される

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

*) ForeignKeyというのは外部キーを表す。



models.Modelには
id = models.AutoField(primary_key=True)というフィールドを持っていて、
idは、自動的にPKとして追加されます。
自動的に追加したくない場合には、別のフィールドにmodels.CharField(primary_key=True)として主キーを付与します

DBのテーブル作成（マイグレーション）

models.pyにテーブルの定義を記述したらマイグレーションを行います。

makemigrations・・・models.pyに加えた変更点をマイグレーションするために、変更点を記録したファイルを作成する。
例) python manage.py makemigrations (アプリケーション名) (--name マイグレーションの名前)

migrate・・・マイグレーションをして、テーブルの定義の変更をDBに反映させる。
例) python manage.py migrate (アプリケーション名)

showmigrations・・・Djangoのプロジェクト内で過去に実行されたマイグレーションのリストを表示する。
例) python manage.py showmigrations (アプリケーション名)

マイグレーションを特定の地点まで戻したい場合
python manage.py migrate アプリケーション名 マイグレーションの名前

マイグレーションを実施していない状態に戻す
python manage.py migrate アプリケーション zero

テーブルのフィールドの定義

フィールドには以下のような型があります。
*) 参考: https://docs.djangoproject.com/ja/3.1/ref/models/fields/#model-field-types

BooleanField	論理型。TRUE、Falseの２値を入れられる
CharField	文字列型。VARCHARとしてカラムが作成される。max_lengthで長さを指定する
DateField, DatetimeField	日付型、タイムスタンプ型のカラムが作成される
EmailField	メールを格納する。デフォルトでVARCHAR(254)として作成される
FileField	ファイルのアップロードの際に用いられる。デフォルトでVARCHAR(100)作成される
DecimalField	正確な数値を格納したい場合に用いられる。
FloatField	浮動小数点数を格納する。
IntegerField	数値型。INTEGERとしてカラムが作成される
SlugField	文字、数値、ハイフン、アンダースコアの文字列。デフォルトでVARCHAR(50)として作成される
TextField	文字列を入れる。デフォルトでTEXT型としてカラムが作成される
URLField	URLを入れる。デフォルトでVARCHAR(200)としてカラムが作成される。
UUIDField	UUIDを入れる。デフォルトでCHAR(30)としてカラムが作成される。

カラムに制約を追加

モデルでテーブルを定義する際に、カラムに制約、インデックスなどを追加することができます。

以下のようにカラム宣言の際にオプションを追加します

オプション	制約	使用例
primary_key	主キー制約 (ユニーク+NOT NULL + インデックス)	models.Field(db.Integer, primary_key=True)
unique	ユニーク制約 (同じ値を入れられない)	models.Field(db.Integer, unique=True)
null	NOT NULL 制約 (Falseの場合NULL値を入れられない)	models.Field(db.Integer, null=False)
db_index	インデックスを作成 (索引。検索の際に高速化できる)	models.Field(db.Text, db_index=True)
default	デフォルト値の追加	models.Field(db.Text, default='A')
blank	Trueの場合、空を許す (デフォルトはFalse)	models.Field(db.Text, blank=True)

管理画面の利用

Djangoでは管理画面をデフォルトで用意されていて、テーブル内のデータの確認やデータの挿入などをすることができます。
(参考: https://docs.djangoproject.com/ja/3.1/ref/contrib/admin/)

管理画面にログインするには、まずスーパーユーザーを作成します。
python manage.py createsuperuser

そのあと、サイトを立ち上げ以下のURLにアクセスすると管理画面にログインできます。

http://127.0.0.1:8000/admin

アプリケーションのフォルダ内にあるadmin.pyに以下のように管理対象のモデルを追加すると、そのモデルを管理画面上で扱えるようになります。

```
from .models import Person


admin.site.register(Person)
```

ModelのMetaオプション

Metaオプションを追加することで、Model全体の設定を変更します。
(参照: https://docs.djangoproject.com/ja/3.1/ref/models/options/)

class ModelName(models.Model):
    class Meta: **# この中に記載**

オプション	**制約**	**使用例**
abstract	クラスを抽象クラスとして定義する	Class Meta: abstract = True
db_table	DBに登録するテーブル名を指定する	db_table = 'table_a'

ordering	DBからレコードを取り出す場合のデフォルトのOrderを指定する	Ordering = ['pub_date'] # pub_dateカラムで昇順
		Ordering = ['-pub_date'] # pub_dateカラムで降順

unique_together	セットでユニーク(一意)でないといけないフィールドを指定する	unique_together = [['driver', 'restaurant']]
index_together	複合インデックスを作成する	index_together = [['driver', 'restaurant']]
constraints	チェック制約などを設ける	constraints = [models.CheckConstraint(check=models.Q(age__gte=18), name='age_gte_18'),]

Model（データの追加）

DjangoのModelからDBへデータを追加するには、以下のような方法があります。
1. 対象のModelのインスタンスを作成してsaveを実行します。

```
web_site = WebSite(
      url = "www.sample.com", name = “sample"
)
web_site.save()
```

2. クラスのcreateメソッドを用います。

```
WebSite.objects.create(
    name=‘sample’,
    url=‘www.sample.com’
)
```

3. get_or_createメソッドを実行する。

```
obj, created = Person.objects.get_or_create(first_name='Jiro', last_name=‘Sato')
```

この時、first_nameがJiro, last_nameがSatoの人が存在する場合は、objにはPersonクラスのインスタンスが、createdにはTrueが入ります。

```
obj.save() # 更新する
```

Modelからデータの取得

DjangoのModelからデータを取得するには、以下のような方法があります。
1. getメソッドで絞り込んでデータを取得
entry = Entry.objects.get(pk=1) **# Entryクラスから主キーが1のものを取得（取得できない場合はエラー）**
person = Person.objects.get(first_name='taro') **# Personクラスからfirst_nameがtaroのものを取得（取得できない場合と複数取得した場合はエラー）**
p = Person.objects.get(first_name='taro', last_name='sato') **# Personクラスからfirst_nameがtaro, last_nameがsatoのものを取得（取得できない場合はエラー）**

2. 値を全て取得(allメソッド)
persons = Person.objects.all() **# Personクラスからレコードを全て取得する。**

3. filterで特定の条件で絞り込んで、allで取得

p = Person.objects.filter(first_name='taro').all() **# Personクラスからfirst_nameがtaroのもののみを取得（取得できない場合もエラーにはならない）**

Modelからデータを更新

DjangoのModelからデータを更新するには、以下のような方法があります。
1. getメソッド等でデータを取得して、直接書き換えsaveで更新
person = Person.objects.get(first_name='taro') **# Personクラスからfirst_nameがtaroのものを取得（取得できない場合はエラー）**
person.first_name = 'jiro' **# personのfirst_nameをjiroに変更**
person.save() **# 変更した内容で更新**

2. updateメソッドを使用して更新（一度に更新したい場合に利用する）
Event.objects.filter(id=4).update(event_date=event_date) **# id=4のレコードに対して、event_dateを更新する**

Modelからデータを削除

DjangoのModelからデータを削除するには、以下のような方法があります。

1. filterで、データを絞り込んで、deleteメソッドを実行
Person.objects.filter(first_name='taro').delete() **# Personクラスからfirst_nameがtaroのものを削除**

2. allで、全件取得してdeleteメソッドを実行し、レコードを全て削除
Person.objects.all().delete() **# Personクラスの内容を全て削除**

テーブル間を紐づける（外部キー）

外部キーを用いて、テーブル間を紐づける場合には以下のように記述します。
(参照: https://docs.djangoproject.com/ja/3.1/ref/models/fields/#module-django.db.models.fields.related)

```
class Car(models.Model):
    manufacturer = models.ForeignKey(
        'Manufacturer',
        on_delete=models.CASCADE,
    )
    # ...

class Manufacturer(models.Model):
    # ...
    pass
```

on_deleteには、レコード削除時の動作を定義します。
models.CASCADE: 参照先が削除されたとき、強制的にレコードを削除する。

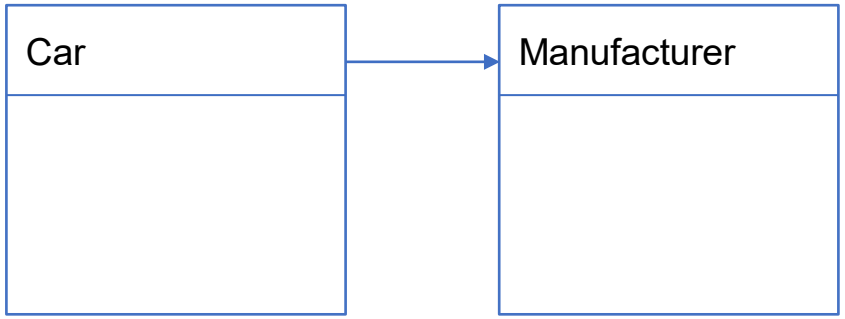
models.PROTECT: 参照先を削除する際にProtectedErrorを発生させて、保護する

models.RESTRICT:参照先を削除する際にRestictedErrorを発生させて、保護する。ただし、参照先の参照先が削除される際に、CASCADEでその参照先に紐づけられいた場合には、保護されず削除される。

models.SET_NULL: 参照先が削除された場合、NULLが入る

models.SET_DEFAULT:参照先が削除された場合、デフォルト値が入る

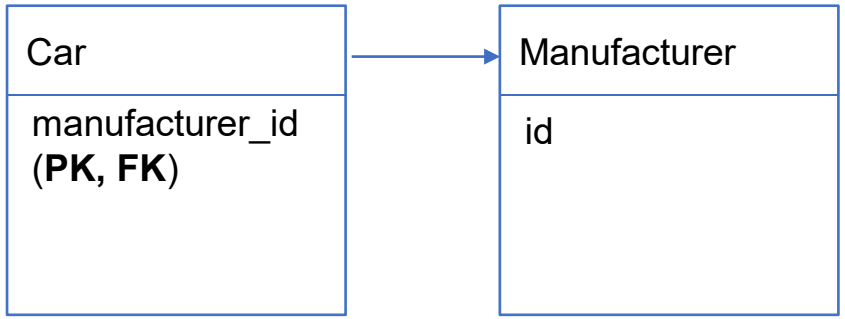
models.SET(): SETで指定した値を設定する



テーブル間を紐づける（One-to-one）

前回は、テーブルの紐づけとしてデフォルトの1対多で紐づけをしました。ここでは、1対1の紐づけを行います。
(参照: https://docs.djangoproject.com/ja/3.1/topics/db/examples/one_to_one/)

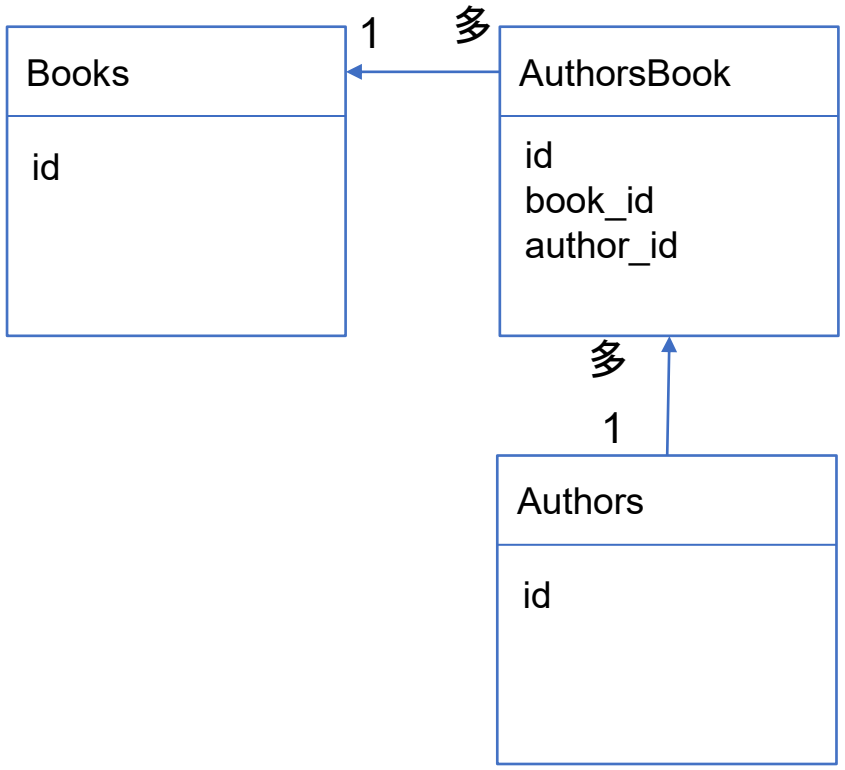
```
class Car(models.Model):
    manufacturer = models. OneToOneField(
        Manufacturer,
        on_delete=models.CASCADE,
        primary_key=True,
    )
    # ...

class Manufacturer(models.Model):
    # ...
    pass
```



テーブル間を紐づける（Many-to-many）

ここでは、多対多の紐づけを行います。多対多の紐づけでは、間にテーブルをはさむことで実現します。
(参照: https://docs.djangoproject.com/ja/3.1/topics/db/examples/many_to_many/)

```
class Authors(models.Model):
    pass
    # ...

class Books(models.Model):
     books = models. ManyToManyField(
         Authors,
    )

# インスタンスを追加する場合、
book1.authors.add(author1)
book1.authors.add(author2, author3)
```



結合先のテーブルからデータを取得する

【1対多での結合】

ForeignKeyフィールドがある側
インスタンス名.フィールド名　で取得する

ForeignKeyフィールドがない側
インスタンス名.フィールド名_set.all()　で取得する

【1対1での結合】

ForeignKeyフィールドがある側
インスタンス名.フィールド名　で取得する

ForeignKeyフィールドがない側
インスタンス名.フィールド名　で取得する

【多対多での結合】

ForeignKeyフィールドがある側
インスタンス名.フィールド名.all()　で取得する

ForeignKeyフィールドがない側
インスタンス名.フィールド名_set.all()　で取得する

Modelのクエリ作成詳細

Modelから様々な方法でレコードを取得します。
（参考: https://docs.djangoproject.com/ja/3.1/topics/db/queries/）

【レコードを一件取得】
Model.objects.first()

【要素数を制限する】
Model.objects.all()[:5] # LIMIT 5で最初の5件を取得する。

【レコードを絞り込む】
Model.objects.filter(field='〇〇') # fieldが○○のレコードのみに絞り込む
Model.objects.filter(field1='〇〇', field2='××') # field1が○○かつ、field2が××のレコードのみに絞り込む
Model.objects.filter(field**__startswith**='A') # fieldがAで始まるものだけに絞り込む
Model.objects.filter(field**__endswith**='A') # fieldがAで終わるものだけに絞り込む
Model.objects.filter(age**__gt**=20) # ageが20より大きいものだけに絞り込む
Model.objects.filter(age**__lt**=20) # ageが20より小さいものだけに絞り込む
Model.objects.filter(age**__gte**=20) # ageが20以上ものだけに絞り込む
Model.objects.filter(age**__lte**=20) # ageが20以下ものだけに絞り込む

or条件(from django.db.models import Q)
Model.objects.filter(**Q**(field1='〇〇') | **Q**(field2='××')) # field1が○○または、field2が××のレコードのみに絞り込む

queryフィールド # SQLを取得する

Modelのクエリ作成詳細

【レコードを絞り込む】
Model.objects.filter(field__**in**=[‘〇’, ‘×’, …]) # fieldの値が○, ×, … に該当するレコードのみに絞り込む
Model.objects.filter(field__**contains**=‘〇〇’) # fieldに○○を含むレコードのみに絞り込む(大文字小文字を区別)
Model.objects.filter(field__**icontains**=‘〇〇’) # fieldに○○を含むレコードのみに絞り込む(大文字小文字を区別しない)
Model.objects.filter(field__**isnull=True**) # fieldがNULLのレコードのみに絞り込む

【レコードを取り除く】
Model.objects.exclude(field=‘〇〇’) # fieldが○○のレコードのもの以外を取り出す。

【一部のカラムのみを取り出す】
Model.objects.values(‘column1’, ‘column2’).all(): # カラムcolumn1とcolumn2だけを取り出す。

【順番を並び替える】
Model.objects.order_by(‘column1’).all() # column1で昇順に並び替え
Model.objects.order_by(‘-column1’).all() # column1で降順に並び替え
Model.objects.order_by(‘column1’, ‘column2’).all() #column1で昇順に並び替えた後、column2で昇順に並び替え
Model.objects.order_by(‘column1’, ‘-column2’).all() #column1で昇順に並び替えた後、column2で降順に並び替え

Modelのクエリ作成詳細

【集計をする】

```
Model.objects.count() # 件数をカウントする(intで返される)

from django.db.models import Max, Min, Avg, Count, Sum
Model.objects.aggregate(Max('column')) # columnの最大値を求める
Model.objects.aggregate(Min('column')) # columnの最小値を求める
Model.objects.aggregate(Avg('column')) # columnの平均値を求める
Model.objects.aggregate(Count('column')) # 件数を求める
Model.objects.aggregate(count_column=Count('column')) # Countに対して別名をつける

# column1でGroup Byして、column2のMaxを計算する
Model.objects.values('column1').annotate(
  Max('column2')
)
Model.objects.values('column1').annotate( # 別名をつける
  max_column=Max('column2')
)
```

Modelのクエリ作成詳細

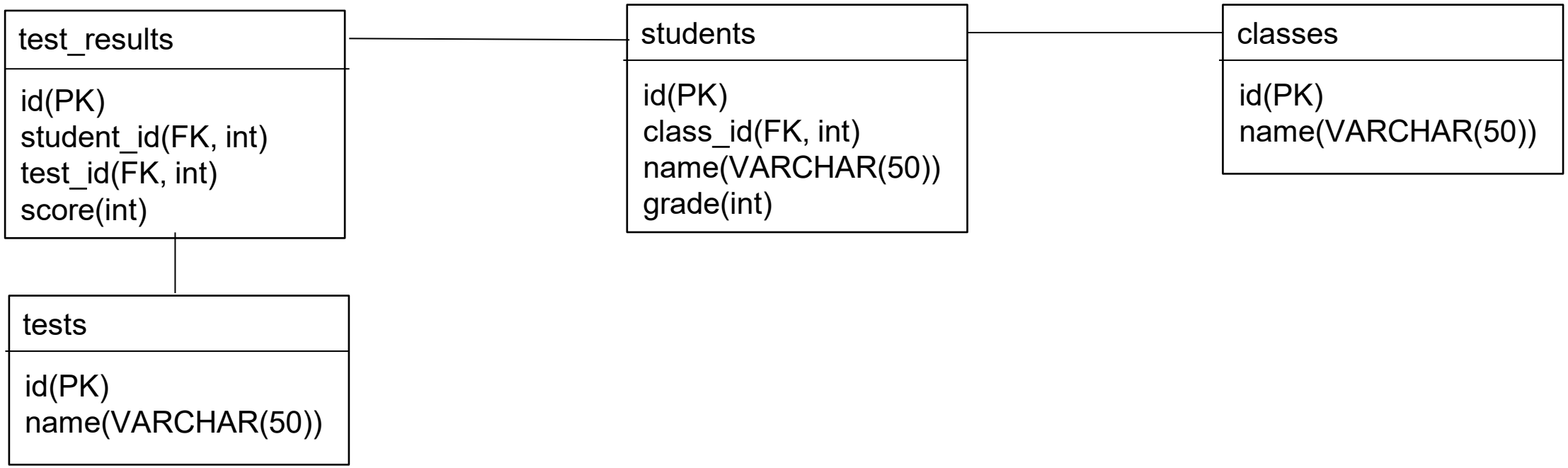
【外部テーブルの情報を使用する】
Model.objects.filter(外部テーブル__カラム='○○') # 外部テーブルのカラムを指定して絞込み
Model.objects.filter(外部テーブル__外部テーブル__カラム='○○') # 外部テーブルの外部テーブルのカラムを指定して絞込み
Model.objects.order_by('外部テーブル__カラム') # 外部テーブルのカラムで並び替え（昇順）
Model.objects.order_by('-外部テーブル__カラム') # 外部テーブルのカラムで並び替え（降順）

Model.objects.values('外部テーブル__カラム').annotate(Count('id')) # 外部テーブルのカラムでGROUP BYして集計

問題

簡単にデータを挿入して取り出す演習を行います。

1. ModelExamというプロジェクトを作成しましょう
2. ModelAppというアプリケーションを作成しましょう
3. migrateを行って、Djangoのデフォルトのデータベースを作成しましょう
4. ModelAppの中に以下のテーブルを作成します。



問題

5. 各テーブルにデータを入れましょう。classesには10件(classA …. classJ)。studentsには各クラスに10人(classA studentA … class J studentJ)ずつ(gradeは1)。testsには、数学、英語、国語の3教科。test_resultsには、50~100点のランダム値を各生徒に対して、数学、英語、国語の3教科に対して付けましょう。
6. idが1のstudentsの名前とテストの科目と点数、を表示する。
7. クラス毎の、各科目のテストの合計、平均、最大、最小を表示する。