1.DB Normalization:
- Introduction:
  - a process that helps organize and structure relational DB efficiently
  - a set of guidelines or rules are applied:
    1.1. designing the DB schema
    1.2. minimizing data redundancy(no repetitive data)
    1.3. ensuring data integrity
    1.4. eliminating data anomalies
    1.5. improving the efficiency and maintainability
- Key Benefits

2.Relationships in Django Models:
- Introduction
  - Model relationships allow defining how different tables(models) relate to each other
- Foreign Key
- ManyToManyField
- Related Name: when defining a FK from one model to another you can specify a related name for the reverse relationship
  The related name allows you to access related objects from the other model conveniently
  By default, the related name is generated automatically by appending '_set' to the lowercase name of the model that defines the FK

3.Types of Relationships:
- One-to-One:
  2required positional arguments:
  - the class to which the model is related
  - on_delete opiton

  Most useful on the primary key of an object when that object 'extends' another object in some way
- One-to-Many:
  2required positional arguments:
  - the class to which the model is related
  - required on_delete option
    - related_name is optional

    ***

```
class Lecturer(models.Model):
    first_name = models.CharField(
        max_length=100,
    )
    last_name = models.CharField(
        max_length=100,
    )
    def __str__(self):
        return f'{self.first_name} {self.last_name}'


class Subject(models.Model):
    name = models.CharField(
        max_length=100,
    )
    code = models.CharField(
        max_length=10,
    )
    # a subject has only one lecturer, FK is many-to-one
    lecturer = models.ForeignKey(
        to='Lecturer',
        on_delete=models.SET_NULL,
```

```
            null=True,
      )
      def __str__(self):
            return self.name

# print(lecturer1.subject_set.all())

class Student(models.Model):
      student_id = models.CharField(
            max_length=10,
            primary_key=True,
      )
      first_name = models.CharField(
            max_length=100,
      )
      last_name = models.CharField(
            max_length=100,
      )
      birth_date = models.DateField()
      email = models.EmailField(
            unique=True,
      )
      subjects = models.ManyToManyField(
            to='Subject',
            through='StudentEnrollment',
      )

class StudentEnrollment(models.Model):
      GRADES = (
            ('A', 'A'),
            ('B', 'B'),
            ('C', 'C'),
            ('D', 'D'),
            ('F', 'F'),
      )
      student = models.ForeignKey(
            to='Student',
            on_delete=models.CASCADE,
      )
      subject = models.ForeignKey(
            to='Subject',
            on_delete=models.CASCADE,
      )
      enrollment_date = models.DateField(
            default=date.today,
      )
      grade = models.CharField(
            max_length=1,
            choices=GRADES,
      )
class LecturerProfile(models.Model):
      lecturer = models.OneToOneField(
            to='Lecturer',
            on_delete=models.CASCADE,
      )
      email = models.EmailField(
            unique=True,
      )
      bio = models.TextField(
```

```
                              null=True,
                              blank=True,
                        )
                        office_location = models.CharField(
                              max_length=100,
                              null=True,
                              blank=True,
                        )
                  ***
```

- Many-to-Many
  requires only 1 positional argument(the class to which the model is related)
  -through option

3.1.:
  ON DELETE:
  - SET_NULL -> only possible if null is True
  - CASCADE
  - PROTECT -> Protected error
  - RESTRICT
  - SET
  - DO_NOTHING

TODO: install ipython