Database access using Django. Questions.

1. Indicate an example of database where it is better to use a relational database better than a NoSQL database and explain why.

   In an inventory system, the data tends to follow a well-defined structure with clear relationships between entities, such as products, categories, suppliers, and stock. A relational database is perfectly suited for this type of model, as it can organize the data into tables that are related through foreign keys (for example, a product belongs to a category, and a supplier provides products).

   The tabular structure of relational databases simplifies the representation and management of the relationships between these entities. In contrast, with NoSQL, you would have to handle these relationships in a less structured and more complex manner, often duplicating data. This can lead to less efficient data management and potential inconsistencies.

2. If we have the following tables in a relational database:

   student: idStudent, idAddress (foreign key to table address), name, age.

   address: idAddress, street, number, postalCode, idProvince (foreign key to table province)

   province: idProvince, description.

   - Indicate the equivalence in an Object Oriented Model.

     ```
     class Province:
         idProvince (Integer)
         description (String)

     class Address:
         idAddress (Integer)
         street (String)
         number (Integer)
         postalCode (String)
         idProvince (Province)

     class Student:
         idAddress (Address)
         idStudent (Integer)
         name (String)
         age (Integer)
     ```

- Indicate the equivalence in a documental database that uses JSON files.

```
{
"idStudent": 1,
   "name": "John Doe",
   "age": 20,
   "address": {
      "idAddress": 101,
      "street": "123 Main St",
      "number": 5,
      "postalCode": "12345",
      "province": {
         "idProvince": 10,
         "description": "California"
      }
   }
}
```

- Indicate the equivalence using a xml file.

```
<student>
  <idStudent>1</idStudent>
   <name>John Doe</name>
   <age>20</age>
   <address>
      <idAddress>101</idAddress>
      <street>123 Main St</street>
      <number>5</number>
      <postalCode>12345</postalCode>
      <province>
         <idProvince>10</idProvince>
         <description>California</description>
      </province>
   </address>
</student>
```

3. Indicate three advantages of using a database better than a file to store information.

   Data Integrity: Databases enforce integrity constraints (e.g., primary and foreign keys), ensuring the consistency of the data.

   Concurrency: Databases allow multiple users to access and modify data simultaneously, while ensuring proper isolation between transactions.
   Querying: Databases offer advanced querying capabilities, enabling you to search, filter, and aggregate data efficiently with SQL, unlike file systems which require manual parsing.

4. Indicate the SQL code to perform the following actions:
    - Create a table with name student and fields: idStudent (integer, PK), name (string), age (number)

      ```
      CREATE TABLE student (
          idStudent   INTEGER  PRIMARY KEY,
          name        VARCHAR(100),
          age         INTEGER,
      );
      ```

    - Add two rows of data in the previous table.

      ```
      INSERT INTO student (idStudent, name, age) VALUES (1, Alejandro, 22);
      INSERT INTO student (idStudent, name, age) VALUES (2, Nicolás, 20);
      ```

    - Modify the second row of data that you added.

      ```
      UPDATE student SET name = ‘Adrián’, age = 21 WHERE idStudent = 1;
      ```

    - Delete the first row of data in that table.

      ```
      DELETE FROM student WHERE idStudent = 2;
      ```

5. What is a Django Model?

   A Django model is a Python class that defines the structure of a database table. It represents a single table in a database and provides an abstraction layer to interact with the data using Python code instead of raw SQL. Django models are used to define fields (columns) and behaviors of the data.

6. What does “object-relational mapping” mean?

   Object-relational mapping (ORM) is a technique that allows you to interact with a relational database using object-oriented programming principles. Django's ORM translates Python class instances into database rows and fields into columns, enabling developers to query, insert, update, and delete records using Python code instead of SQL.

7. Define the Django model that is equivalent to the table in the exercise 4. Indicate which actions must be performed so that this model is mapped to a relational database.

```
from django.db import models


class Student(models.Model):
    idStudent = models.IntegerField(primary_key=True)
    name = models.CharField(max_length=100)
    age = models.IntegerField()
```

After adding the model, we must do the migrations.

8. What is the purpose of the field validators that we can specify in the Django models?

Field validators are used to enforce rules on the data that is saved to the database. For example, you can specify that an email field must contain a valid email address or that a number field must be within a specific range. Validators help maintain data integrity by ensuring that data adheres to specific formats or conditions before it is stored.

9. If I have a student with name “Luisa” in the database. Write the Django code that we must execute in order to modify it.

```
student = Student.objects.get(name='Luisa')
student.name = 'Luisa Updated'
student.save()
```

10. Write the code of the filters that are equivalent to the following SQL sentences:
    - select * from student where age between 20 and 70;

      ```
      Student.objects.filter(age__gte=20, age__lte=70)
      ```

    - select * from student where name like ‘%Gomez%’ and (register = 1 or previousRegister = 1);

      ```
      Student.objects.filter(name__icontains='Gomez', register=1 | previousRegister=1)
      ```

    - select * from customer where age >= 18 or city = ‘B%’;

      ```
      Customer.objects.filter(models.Q(age__gte=18) | models.Q(city__startswith='B'))
      ```

11. Describe with your own words the following concepts:
    - Persistent Data.

      Data that is stored and remains intact even after the application or system is shut down (e.g., data in a database).
    - Temporary Data.

      Data that exists only during the runtime of an application and is discarded after the application ends (e.g., session variables or cache).
    - Semi-Persistent Data.

      Data that exists for some time, often between different application runs, but is not guaranteed to be kept forever (e.g., data stored in cache or log files).

12. Write the django code to perform the following tasks:
    - Get all registers from the table ‘student’. Show only the 4 first registers.

      Students.objects.all()[:4]
    - Get from the previous table the register with id=3.

      Students.objects.get(idStudent=3)

13. We can choose between two ways to operate on the database:

    Case 1:

    ```
    bestsellers = Book.objects.filter(is_bestselling=True)
    amazing_bestsellers = bestsellers.filter(rating__gt=4)
    print (bestsellers)
    print (amazing_bestsellers)
    ```

    Case 2:

    ```
    print (Book.objects.filter(is_bestselling=True))
    print (Book.objects.filter(is_bestselling=True, rating__gt=4))
    ```

    - How many times does Django access the database in “Case 1”? And how many times in “Case 2”?

      Case 1 access twice and case 2 access once
    - Which case gives a better performance?

      Case 2 is better because it minimizes the number of database queries.

14. Explain with your own words what is the “Cache” and what is its purpose.

A "cache" is a temporary storage that holds frequently used data for faster access. Its purpose is to improve performance by avoiding slower retrieval from other sources.