DATABASE DESIGN VIA DJANGO FRAMEWORK

Yaroslav Makhenko,
Assistant of the Department of Computer Sciences and Information Technologies
Oksana Yatsenko,
Assistant of the Department of Computer Sciences and Information Technologies
Yanina Stelmashenko,
Academic Degree: Master's
Kateryna Nekhaienko,
Academic Degree: Master's
Zhytomyr Ivan Franko State University, Ukraine

In today's realities, a website or web application that doesn't store data for further processing is impossible to imagine. Therefore, a database is important for convenient information management, storage, and processing to enhance the capabilities of a website.

Django is a Python web development framework that provides powerful tools for working with databases. Django has its own modelling language that allows developers to easily create database structures for their applications. Therefore, the goal is an overview of some general methods and recommendations for helping to create efficient and scalable databases.

To begin with, the list of databases supported by the Django framework is worth considering:

- PostgreSQL is an object-relational database management system that is not controlled by any one company, but rather is developed through the collaboration of many people and companies interested in using this database management system and implementing the latest developments.
- MariaDB is a relational database management system created in early 2009 as a fork of MySQL.
- MySQL is a free relational database management system that was developed by TcH to increase the speed of processing large databases. The open source database management system (DBMS) was created as an alternative to commercial systems. MySQL was originally quite similar to mSQL, however over time it has expanded and now MySQL is one of the most widely used database management systems. Primarily, the system is used to create dynamic web pages, as it has excellent support from a variety of programming languages.
- Oracle is an object-relational database management system by Oracle Corporation.

SQLite is a simplified relational database management system. The system is implemented in the form of a library containing many of the features of the SQL-92 standard.

Databases in Django are represented by model classes. A model is a description of a table structure in a database. It determines the names of the table fields, the data types of the fields, and their restrictions. The Model class from the django.db.models package is used for creating a model. As an example, the creation of a bookstore model will look like this (Figure 1):

```
from django.db import models

class Shop(models.Model):
    name = models.CharField('Name', max_length=200)
    address = models.TextField('Address',)
    opening_time = models.TimeField('Opening time')
    closing_time = models.TimeField('Closing time')

    def __str__(self):
        return f'{self.name} / {self.opening_time} - {self.closing_time}'

    class Meta:
        verbose_name = 'Shop'
```

Figure 1. Creating a bookshop model

The next step is importing django.db.models and creating a Shop model with 4 fields:

- name - the store's name, a field of CharField type that can contain small text, limiting the field size to 200 characters.
- address – the store address, a field of TextField type that can contain large text.
- opening_time and closing_time – the store's opening and closing times, fields of the TimeField type that contain an instance of datetime.time, namely, the time.

For further work with the site at this stage, the "Genre" model should be created (Fig. 2), which should contain only one field of the CharField type - the name of the genre:

```
class Genre(models.Model):
    name = models.CharField('Name', max_length=200)

    def __str__(self):
        return f'{self.name}'

    class Meta:
        verbose_name = 'Genre'
```

Figure 2. Creating the Genre model.

The next step is to create the last model - "Book" (Fig. 3). This model should contain the following fields:

- name - the book title, a field of CharField type that can contain small text, limit the field size to 200 characters.
- genre - the book genre, a field of ForeignKey type referring to the Genre model.
- quantity - the book quantity, a field of IntergerField type that can contain an integer.
- price - the book price, a field of FloatField type which may contain any number.

```
class Book(models.Model):
    name = models.CharField('Name', max_length=200)
    genre = models.ForeignKey(Genre, models.PROTECT, verbose_name='Жанр')
    quantity = models.IntergerField('Quantity')
    price = models.FloatField('Price')

    def __str__(self):
        return f'{self.name} / {self.price}'

    class Meta:
        verbose_name = 'Book'
```

Figure 3. Creating the "Book" model.

The convenience of database design using models means the code is universal for any of the databases presented.

Django creates and updates databases using migrations. A migration is a file describing the changes that need to be made to a database.

A migration is created using the makemigrations command. For example, the command python manage.py makemigrations generates a migration to create the Shop, Genre, and Book tables in the database. Django runs the migrate command to make the changes described in the migration. For example, the python manage.py migrate command applies the migration and creates the tables in the database.

The database design process with Django is based on certain rules:

- Use the right field types. Select field types appropriate for the type of data stored.

- Exploit restrictions. Restrictions help prevent errors and ensure data integrity. It is very important to properly validate data, set maximum and minimum values for numeric fields, and sizes for text fields.
- Create a connection. Model connections help to link multiple tables, as in our example, we linked the table with the book list to the genre table.

Database design is an important stage in the development of any website or web application. Django provides powerful tools for designing and building databases. The tools allow to create efficient and scalable databases that meet the needs of modern trends.

References**:**

1. Databases. Django documentation. Django: website. URL: https://docs.djangoproject.com/en/4.2/ref/databases/#postgresql-notes (application date 23.10.2023)

2. PostgreSQL: The world's most advanced open source database: website. URL: https://www.postgresql.org/ (application date 23.10.2023)

3. MariaDB Foundation - MariaDB.org: website. URL: https://mariadb.org/ (application date 23.10.2023)

4. Oracle Ukraine| Cloud Applications and Cloud Platform: website. URL: https://www.oracle.com/ua/ (application date 23.10.2023)