МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БГТУ им. В.Г.ШУХОВА»

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

КУРСОВАЯ РАБОТА

по дисциплине: «Базы данных»

тема: Разработка веб-приложения для использования базы данных Хоккейной лиги с помощью фреймворка Django

Автор работы Донцов Александр Алексеевич, ПВ-31

(подпись)

Руководитель проекта Панченко Максим Владимирович (подпись)

Оценка

Белгород 2019г.

|  |  |  |
| --- | --- | --- |
|  | Оглавление |  |
| [**Глава 1**](#_bookmark0) |  | [3](#_bookmark0) |
| [**Постановка задачи**](#_bookmark1) |  | [3](#_bookmark1) |
| [**Основная информация**](#_bookmark2) |  | [3](#_bookmark2) |
| [**Глава 2**](#_bookmark3) |  | [3](#_bookmark3) |
| [**Языки SQL и NoSQL**](#_bookmark4) |  | [3](#_bookmark4) |
| [**Структура Языка SQL**](#_bookmark5) |  | [4](#_bookmark5) |
| [**Выбор БД нужного типа**](#_bookmark6) |  | [4](#_bookmark6) |
| [**Глава 3**](#_bookmark7) |  | [4](#_bookmark7) |
| [**Используемые технологии**](#_bookmark8) |  | [4](#_bookmark8) |
| **Диаграмма сущность-связь** |  | [5](#_bookmark9) |
| [**С**](#_bookmark10)**труктура [базы данных](#_bookmark10)** |  | [6](#_bookmark10) |
| [**Код программы**](#_bookmark11) |  | [6](#_bookmark11) |
| [**Скриншоты**](#_bookmark12) |  | 18 |
| [**Заключение**](#_bookmark13) |  | 20 |

**Глава 1**

**Постановка задачи:**

Цель – создание веб-приложения с использованием базы данных, содержащую основную информацию о лигах, клубах, игроках, тренерах, матчах и спонсорах. Должна быть реализована возможность просмотра, добавления и удаления основной информации о элементах базы.

**Основная информация:**

Главная страница сайта отображает приветствие. По ссылке просмотр, открываются списки таблиц, каждый элемент можно открыть в отдельной странице, где его можно удалить или посмотреть информацию о нем. Также в таблицах можно добавлять элементы, переходя по ссылке создания элемента.

**Глава 2**

**Языки SQL и NoSQL**

Когда необходимо выбрать СУБД, главный вопрос обычно заключается в выборе реляционной (SQL) или нереляционной (NoSQL) структуры. У обоих вариантов есть свои преимущества, а также несколько ключевых особенностей, которые стоит иметь в виду при выборе.

**Реляционные базы данных** используют структурированный язык запросов (Structured Query Language, SQL) для определения и обработки данных. С одной стороны, это открывает большие возможности для разработки: SQL один из наиболее гибких и распространённых языков запросов, так что его выбор позволяет минимизировать ряд рисков, и будет особенно кстати, если

предстоит работа с комплексными запросами. С другой стороны, в SQL есть ряд ограничений. Построение запросов на этом языке обязывает предопределять структуру данных, а последующее изменение структуры данных может быть губительным для всей системы.

**Нереляционные базы данных**, в свою очередь, предлагают динамическую структуру данных, которые могут храниться несколькими способами:

ориентированно по колонкам, документо-ориентированно, в виде графов или на основе пар «ключ-значение». Такая гибкость означает следующее:

* Можно создавать документы, не задавая их структуру заранее;
* Каждый документ может обладать собственной структурой;
* У каждой базы данных может быть собственный синтаксис;
* Можно добавлять поля прямо во время работы с данными.

**Структура Языка SQL**

В реляционных СУБД данные представлены в виде таблиц, в то время как в нереляционных — в виде документов, пар «ключ-значение». Это делает SQL базы данных лучшим выбором для приложений, которые предполагают транзакции с несколькими записями — как, например, система учётных записей — или для устаревших систем, которые были построены для реляционных структур.

**Выбор БД нужного типа**

Реляционная БД — верный выбор для любого проекта, который может положиться на предопределённую структуру и заданные схемы. С другой стороны, документо-ориентированная — отличный вариант для быстрорастущих проектов без определённой схемы данных. В особенности если невозможно определить схему для своей базы данных изначально.

**Глава 3**

**Используемые технологии**

При разработке приложения использовалась СУБД PostgreSQL, а также фреймворк Django.

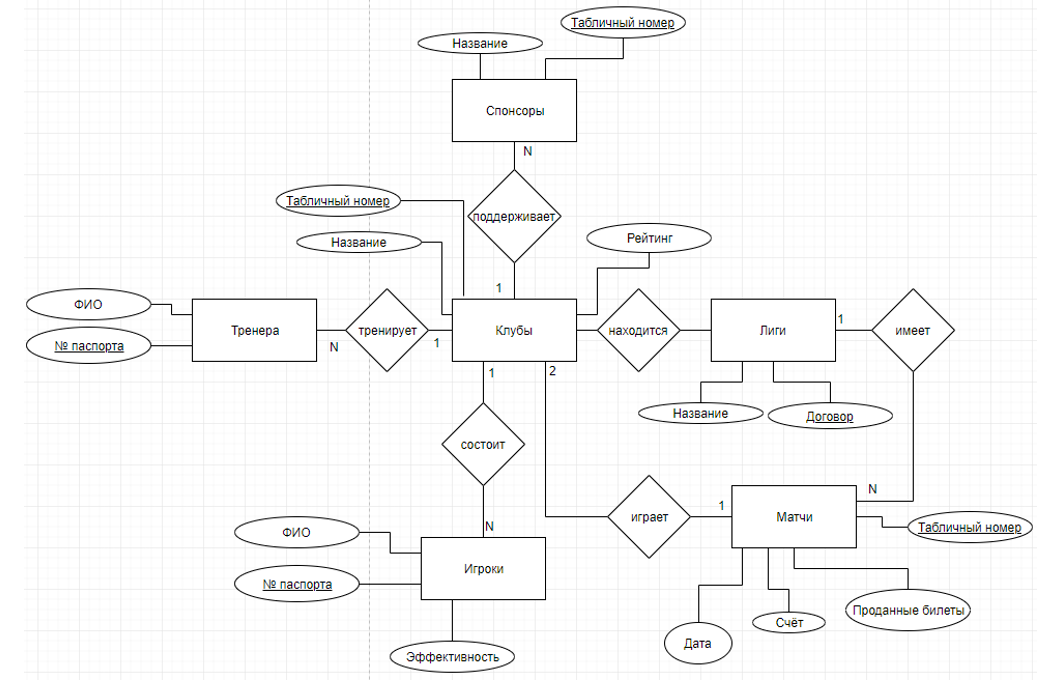
**Django** — [свободный](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) на языке [Python](https://ru.wikipedia.org/wiki/Python" \o "Python), использующий шаблон проектирования [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller). Проект поддерживается организацией [Django Software Foundation](https://ru.wikipedia.org/wiki/Django_Software_Foundation" \o "Django Software Foundation).

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других. Один из основных принципов фреймворка — [DRY](https://ru.wikipedia.org/wiki/DRY) (*Don't repeat yourself*).

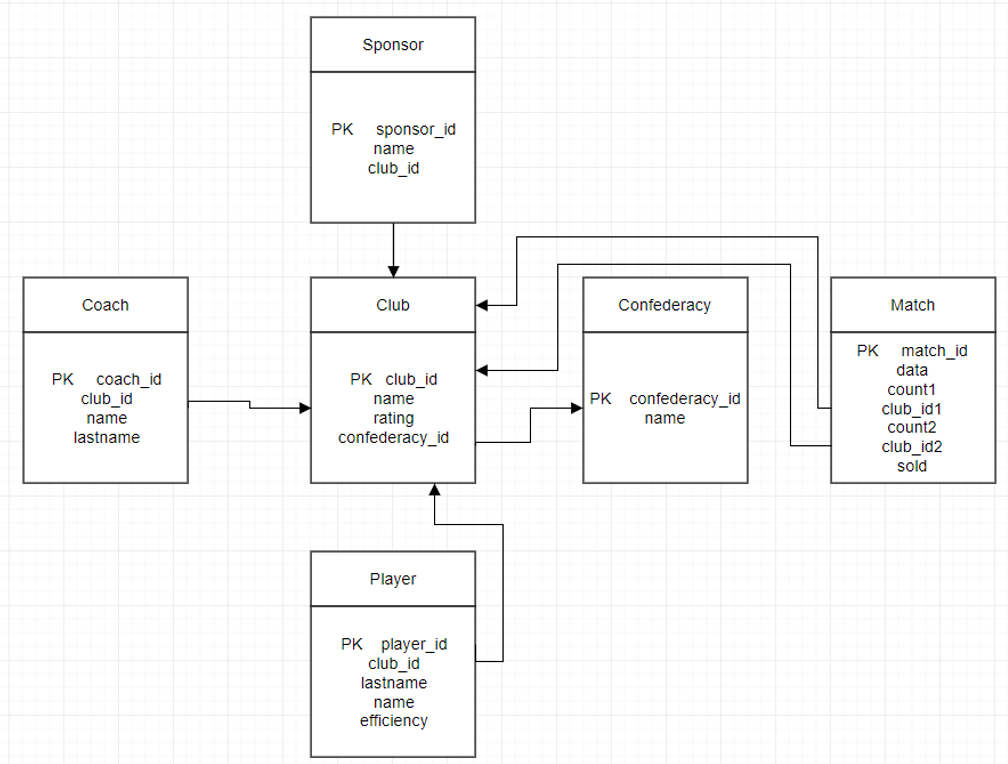
Также, в отличие от других фреймворков, обработчики [URL](https://ru.wikipedia.org/wiki/URL) в Django конфигурируются явно при помощи [регулярных выражений](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F).

Для работы с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) Django использует собственный [ORM](https://ru.wikipedia.org/wiki/ORM), в котором модель данных описывается классами Python, и по ней генерируется схема [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%94).

**Диаграмма сущность-связность:**

****

**Структура базы данных:**

****

Для подключения существующей БД к проекту Django добавляется следующий код в файл *settings.py* (настройки проекта):

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql\_psycopg2',

'NAME': 'Databases',

'USER' : 'postgres',

'PASSWORD' : 'admin',

'HOST' : 'hostlocal',

'PORT' : '5432',

}

}

Ниже представлен код основных файлов проекта – *models.py* (модели проекта), *views.py* (представления) и *urls.py* (url-адреса, связывающие представления с html-шаблонами):

**Код программы:**

***models.py***

from django.db import models

class AuthGroup(models.Model):

name = models.CharField(unique=True, max\_length=150)

class Meta:

managed = False

db\_table = 'auth\_group'

class AuthGroupPermissions(models.Model):

group = models.ForeignKey(AuthGroup, models.DO\_NOTHING)

permission = models.ForeignKey('AuthPermission', models.DO\_NOTHING)

class Meta:

managed = False

db\_table = 'auth\_group\_permissions'

unique\_together = (('group', 'permission'),)

class AuthPermission(models.Model):

name = models.CharField(max\_length=255)

content\_type = models.ForeignKey('DjangoContentType', models.DO\_NOTHING)

codename = models.CharField(max\_length=100)

class Meta:

managed = False

db\_table = 'auth\_permission'

unique\_together = (('content\_type', 'codename'),)

class AuthUser(models.Model):

password = models.CharField(max\_length=128)

last\_login = models.DateTimeField(blank=True, null=True)

is\_superuser = models.BooleanField()

username = models.CharField(unique=True, max\_length=150)

first\_name = models.CharField(max\_length=30)

last\_name = models.CharField(max\_length=150)

email = models.CharField(max\_length=254)

is\_staff = models.BooleanField()

is\_active = models.BooleanField()

date\_joined = models.DateTimeField()

class Meta:

managed = False

db\_table = 'auth\_user'

class AuthUserGroups(models.Model):

user = models.ForeignKey(AuthUser, models.DO\_NOTHING)

group = models.ForeignKey(AuthGroup, models.DO\_NOTHING)

class Meta:

managed = False

db\_table = 'auth\_user\_groups'

unique\_together = (('user', 'group'),)

class AuthUserUserPermissions(models.Model):

user = models.ForeignKey(AuthUser, models.DO\_NOTHING)

permission = models.ForeignKey(AuthPermission, models.DO\_NOTHING)

class Meta:

managed = False

db\_table = 'auth\_user\_user\_permissions'

unique\_together = (('user', 'permission'),)

class Confederacy(models.Model):

confederacy\_id = models.AutoField(primary\_key=True)

name = models.CharField(max\_length=50)

class Meta:

managed = False

db\_table = 'confederacy'

def \_\_str\_\_(self):

return self.name

class DjangoAdminLog(models.Model):

action\_time = models.DateTimeField()

object\_id = models.TextField(blank=True, null=True)

object\_repr = models.CharField(max\_length=200)

action\_flag = models.SmallIntegerField()

change\_message = models.TextField()

content\_type = models.ForeignKey('DjangoContentType', models.DO\_NOTHING, blank=True, null=True)

user = models.ForeignKey(AuthUser, models.DO\_NOTHING)

class Meta:

managed = False

db\_table = 'django\_admin\_log'

class DjangoContentType(models.Model):

app\_label = models.CharField(max\_length=100)

model = models.CharField(max\_length=100)

class Meta:

managed = False

db\_table = 'django\_content\_type'

unique\_together = (('app\_label', 'model'),)

class DjangoMigrations(models.Model):

app = models.CharField(max\_length=255)

name = models.CharField(max\_length=255)

applied = models.DateTimeField()

class Meta:

managed = False

db\_table = 'django\_migrations'

class DjangoSession(models.Model):

session\_key = models.CharField(primary\_key=True, max\_length=40)

session\_data = models.TextField()

expire\_date = models.DateTimeField()

class Meta:

managed = False

db\_table = 'django\_session'

class Sponsor(models.Model):

sponsor\_id = models.AutoField(primary\_key=True)

name = models.CharField(max\_length=50)

class Meta:

managed = False

db\_table = 'sponsor'

def \_\_str\_\_(self):

return self.name

class Club(models.Model):

club\_id = models.AutoField(primary\_key=True)

name = models.CharField(max\_length=50)

rating = models.IntegerField(blank=True, null=True)

confederacy = models.ForeignKey(Confederacy, models.CASCADE, blank=True, null=False)

sponsor = models.ForeignKey(Sponsor, models.SET\_NULL, blank=True, null=True)

class Meta:

managed = False

db\_table = 'club'

def \_\_str\_\_(self):

return self.name

class Match(models.Model):

match\_id = models.AutoField(primary\_key=True)

playdata = models.DateTimeField(blank=True, null=True)

count1 = models.IntegerField()

club\_id1 = models.ForeignKey(Club, models.CASCADE, related\_name = 'club\_id1', db\_column='club\_id1', blank=True, null=True)

count2 = models.IntegerField()

club\_id2 = models.ForeignKey(Club, models.CASCADE, related\_name = 'club\_id2', db\_column='club\_id2', blank=True, null=True)

sold = models.IntegerField()

class Meta:

managed = False

db\_table = 'match'

class Coach(models.Model):

coach\_id = models.AutoField(primary\_key=True)

club = models.ForeignKey(Club, models.CASCADE, blank=True, null=True)

name = models.CharField(max\_length=50)

lastname = models.CharField(max\_length=50)

class Meta:

managed = False

db\_table = 'coach'

def \_\_str\_\_(self):

return '%s %s' % (self.name, self.last\_name)

class Player(models.Model):

player\_id = models.AutoField(primary\_key=True)

club = models.ForeignKey(Club, models.CASCADE, blank=True, null=False)

name = models.CharField(max\_length=50)

lastname = models.CharField(max\_length=50)

efficiency = models.IntegerField(blank=True, null=True)

class Meta:

managed = False

db\_table = 'player'

def \_\_str\_\_(self):

return '%s %s' % (self.name, self.lastname)

***views.py***

from django.shortcuts import render, get\_object\_or\_404

from database.models import Sponsor, Confederacy, Club, Coach, Match, Player

from django.shortcuts import redirect

from database.reservation import csv\_writer

from .forms import ConfederacyForm, ClubForm, PlayerForm, CoachForm, MatchForm, SponsorForm

from django.contrib.auth.forms import UserCreationForm

from django.views.generic.edit import FormView

# Вариант регистрации на базе класса FormView

class MyRegisterFormView(FormView):

# Укажем какую форму мы будем использовать для регистрации наших пользователей, в нашем случае

# это UserCreationForm - стандартный класс Django унаследованный

form\_class = UserCreationForm

# Ссылка, на которую будет перенаправляться пользователь в случае успешной регистрации.

# В данном случае указана ссылка на страницу входа для зарегистрированных пользователей.

success\_url = "/login/"

# Шаблон, который будет использоваться при отображении представления.

template\_name = "register/register.html"

def form\_valid(self, form):

form.save()

# Функция super( тип [ , объект или тип ] )

# Возвратите объект прокси, который делегирует вызовы метода родительскому или родственному классу типа .

return super(MyRegisterFormView, self).form\_valid(form)

def form\_invalid(self, form):

return super(MyRegisterFormView, self).form\_invalid(form)

def home(request):

return render(request, 'database/home.html', {})

def view(request):

return render(request, 'database/view.html', {})

def allclub(request):

club=Club.objects.all()

return render(request, 'database/allclub.html', {'clubs':club})

def allcoach(request):

coach=Coach.objects.all()

return render(request, 'database/allcoach.html', {'coachs':coach})

def allconfederacy(request):

confederacy=Confederacy.objects.all()

return render(request, 'database/allconfederacy.html', {'confederacys':confederacy})

def allmatch(request):

match=Match.objects.all()

return render(request, 'database/allmatch.html', {'matchs':match})

def allplayer(request):

player=Player.objects.all()

return render(request, 'database/allplayer.html', {'players':player})

def allsponsor(request):

sponsor = Sponsor.objects.all()

return render(request, 'database/allsponsor.html', {'sponsors':sponsor})

def club(request, pk):

club = get\_object\_or\_404(Club, club\_id=pk)

return render(request, 'database/club.html', {'club':club})

def coach(request, pk):

coach = get\_object\_or\_404(Coach, coach\_id=pk)

return render(request, 'database/coach.html', {'coach':coach})

def confederacy(request, pk):

confederacy = get\_object\_or\_404(Confederacy, confederacy\_id=pk)

return render(request, 'database/confederacy.html', {'confederacy':confederacy})

def match(request, pk):

match = get\_object\_or\_404(Match, match\_id=pk)

return render(request, 'database/match.html', {'match':match})

def player(request, pk):

player = get\_object\_or\_404(Player, player\_id=pk)

return render(request, 'database/player.html', {'player':player})

def sponsor(request, pk):

sponsor = get\_object\_or\_404(Sponsor, sponsor\_id=pk)

return render(request, 'database/sponsor.html', {'sponsor':sponsor})

def club\_new(request):

if request.method == "POST":

form = ClubForm(request.POST)

if form.is\_valid():

error=''

flag=False

if (form.cleaned\_data.get('rating') < 0):

error = 'Отрицательное значение рейтинга'

flag=True

if flag:

return render(request, 'register/club\_edit.html', {'error':error, 'form':form})

post = form.save(commit=False)

post.save()

return redirect('club', pk=post.pk)

else:

form = ClubForm()

return render(request, 'register/club\_edit.html', {'form': form})

def coach\_new(request):

if request.method == "POST":

form = CoachForm(request.POST)

if form.is\_valid():

post = form.save(commit=False)

post.save()

return redirect('coach', pk=post.pk)

else:

form = CoachForm()

return render(request, 'register/coach\_edit.html', {'form': form})

def confederacy\_new(request):

if request.method == "POST":

form = ConfederacyForm(request.POST)

if form.is\_valid():

post = form.save(commit=False)

post.save()

return redirect('confederacy', pk=post.pk)

else:

form = ConfederacyForm()

return render(request, 'register/confederacy\_edit.html', {'form': form})

def match\_new(request):

if request.method == "POST":

form = MatchForm(request.POST)

if form.is\_valid():

error1 = ''

error2 = ''

error3 = ''

flag = False

if(form.cleaned\_data.get('count1') < 0):

error1="Отрицательное значение счёта первой команды"

flag = True

if(form.cleaned\_data.get('count2') < 0):

error2="Отрицательное значение счёта второй команды"

flag = True

if(form.cleaned\_data.get('sold') < 0):

error3="Отрицательное значение продаж"

flag = True

if(flag):

return render(request, 'register/match\_edit.html', {'error1':error1,'error2':error2,'error3':error3, 'form':form})

post = form.save(commit=False)

post.save()

return redirect('match', pk=post.pk)

else:

form = MatchForm()

return render(request, 'register/match\_edit.html', {'form': form})

def player\_new(request):

if request.method == "POST":

form = PlayerForm(request.POST)

if form.is\_valid():

error=''

flag=False

if (form.cleaned\_data.get('efficiency') < 0):

error = 'Отрицательное значение эффективности'

flag=True

if(form.cleaned\_data.get('club') == 0):

error += '<br /> Не выбран клуб'

flag=True

if flag:

return render(request, 'register/player\_edit.html', {'form': form, 'error':error})

post = form.save(commit=False)

post.save()

return redirect('player', pk=post.pk)

else:

form = PlayerForm()

return render(request, 'register/player\_edit.html', {'form': form})

def sponsor\_new(request):

if request.method == "POST":

form = SponsorForm(request.POST)

if form.is\_valid():

post = form.save(commit=False)

post.save()

return redirect('sponsor', pk=post.pk)

else:

form = SponsorForm()

return render(request, 'register/sponsor\_edit.html', {'form': form})

def clubdelete(request, id):

obj = get\_object\_or\_404(Club, club\_id=id)

obj.delete()

return redirect('allclub')

def sponsordelete(request, id):

obj = get\_object\_or\_404(Sponsor, sponsor\_id=id)

obj.delete()

return redirect('allsponsor')

def confederacydelete(request, id):

obj = get\_object\_or\_404(Confederacy, confederacy\_id=id)

obj.delete()

return redirect('allconfederacy')

def matchdelete(request, id):

obj = get\_object\_or\_404(Match, match\_id=id)

obj.delete()

return redirect('allmatch')

def coachdelete(request, id):

obj = get\_object\_or\_404(Coach, coach\_id=id)

obj.delete()

return redirect('allcoach')

def playerdelete(request, id):

obj = get\_object\_or\_404(Player, player\_id=id)

obj.delete()

return redirect('allplayer')

def confederacyreservation(request, id):

obj = get\_object\_or\_404(Confederacy, confederacy\_id=id)

clubs = Club.objects.filter(confederacy = id)

data = []

for club in clubs:

path=Match.objects.filter(club\_id1 = club.club\_id)

for p in path:

#data.append(p.playdata)

match = []

match.append(p.club\_id1.name)

match.append(p.count1)

match.append(p.club\_id2.name)

match.append(p.count2)

data.append(match)

path=Match.objects.filter(club\_id2 = club.club\_id)

for p in path:

#data.append(p.playdata)

match = []

match.append(p.club\_id1.name)

match.append(p.count1)

match.append(p.club\_id2.name)

match.append(p.count2)

data.append(match)

csv\_writer(data, obj.name + '.csv')

confederacy=Confederacy.objects.all()

return render(request, 'database/allconfederacy.html', {'confederacys':confederacy, 'reserv':obj.name + ' зарезервировано'})

***forms.py***

from django import forms

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth.models import User

from django.contrib.admin.widgets import AdminDateWidget

from .models import Sponsor, Player, Coach, Club, Match, Confederacy

class MyForm(UserCreationForm):

email = forms.EmailField(max\_length=200, help\_text='Required')

class Meta:

model = User

fields = ('username', 'email', 'password1', 'password2')

class ConfederacyForm(forms.ModelForm):

class Meta:

model = Confederacy

fields = ('name',)

labels = {

'name': ('Название'),

}

class SponsorForm(forms.ModelForm):

class Meta:

model = Sponsor

fields = ('sponsor\_id', 'name')

labels = {

'sponsor\_id': ('ID Спонсора'),

'name': ('Название'),

}

class PlayerForm(forms.ModelForm):

class Meta:

model = Player

fields = ('player\_id', 'club', 'name', 'lastname', 'efficiency')

labels = {

'player\_id': ('ID Игрока'),

'club': ('Клуб'),

'name': ('Имя'),

'lastname': ('Фамилия'),

'efficiency': ('Эффективность'),

}

class CoachForm(forms.ModelForm):

class Meta:

model = Coach

fields = ('club', 'name', 'lastname')

labels = {

'club': ('Клуб'),

'name': ('Имя'),

'lastname': ('Фамилия'),

}

class ClubForm(forms.ModelForm):

class Meta:

model = Club

fields = ('club\_id', 'name', 'rating', 'confederacy', 'sponsor')

labels = {

'club\_id': ('Клуб'),

'name': ('Имя'),

'rating': ('Рейтинг'),

'confederacy': ('Лига'),

'sponsor': ('Спонсор'),

}

class MatchForm(forms.ModelForm):

class Meta:

model = Match

fields = ('match\_id', 'playdata', 'count1', 'club\_id1', 'count2', 'club\_id2', 'sold')

labels = {

'match\_id': ('ID Матча'),

'playdata': ('Дата игры'),

'count1': ('Счёт первой команды'),

'club\_id1': ('Первая команда'),

'count2': ('Счёт второй команды'),

'club\_id2': ('Вторая команда'),

'sold': ('Количество проданных билетов'),

}

#widgets = {

#'playdata': DateInput(attrs={'type': 'date'}),

#'playdata': forms.DateTimeInput(

# attrs={

# 'type': 'date',

# })

#}

***urls.py***

from django.urls import path

from . import views

from django.db import models

urlpatterns = [

path('', views.home, name='home'),

path('view', views.view, name='view'),

path('allclub', views.allclub, name='allclub'),

path('allcoach', views.allcoach, name='allcoach'),

path('allconfederacy', views.allconfederacy, name='allconfederacy'),

path('allmatch', views.allmatch, name='allmatch'),

path('allplayer', views.allplayer, name='allplayer'),

path('allsponsor', views.allsponsor, name='allsponsor'),

path('accounts/register/', views.MyRegisterFormView.as\_view(), name="register"),

path('allclub/<int:pk>/', views.club, name='club'),

path('allcoach/<int:pk>/', views.coach, name='coach'),

path('allconfederacy/<int:pk>/', views.confederacy, name='confederacy'),

path('allmatch/<int:pk>/', views.match, name='match'),

path('allplayer/<int:pk>/', views.player, name='player'),

path('allsponsor/<int:pk>/', views.sponsor, name='sponsor'),

path('allclub/new/', views.club\_new, name='club\_new'),

path('allcoach/new/', views.coach\_new, name='coach\_new'),

path('allconfederacy/new/', views.confederacy\_new, name='confederacy\_new'),

path('allmatch/new/', views.match\_new, name='match\_new'),

path('allplayer/new/', views.player\_new, name='player\_new'),

path('allsponsor/new/', views.sponsor\_new, name='sponsor\_new'),

path('allclub/delete/<int:id>/', views.clubdelete, name='clubdelete'),

path('allcoach/delete/<int:id>/', views.coachdelete, name='coachdelete'),

path('allconfederacy/delete/<int:id>/', views.confederacydelete, name='confederacydelete'),

path('allmatch/delete/<int:id>/', views.matchdelete, name='matchdelete'),

path('allplayer/delete/<int:id>/', views.playerdelete, name='playerdelete'),

path('allsponsor/delete/<int:id>/', views.sponsordelete, name='sponsordelete'),

path('confederacyreservation/<int:id>/', views.confederacyreservation, name='confederacyreservation'),

]

**reservation.py**

import csv

def csv\_writer(data, path):

"""

Write data to a CSV file path

"""

print(data)

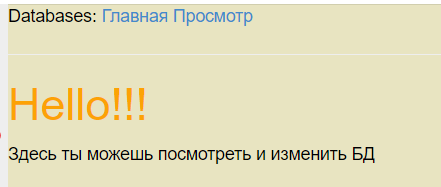
with open(path, "w", newline='') as csv\_file:

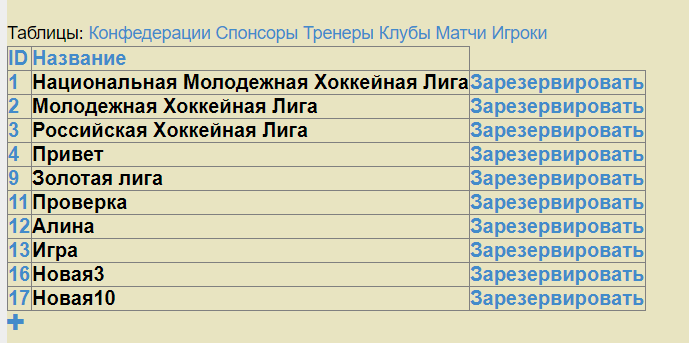
writer = csv.writer(csv\_file, delimiter=',')

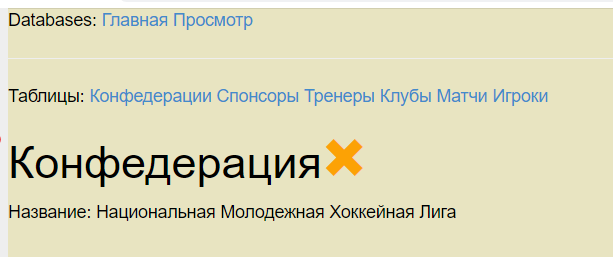
for line in data:

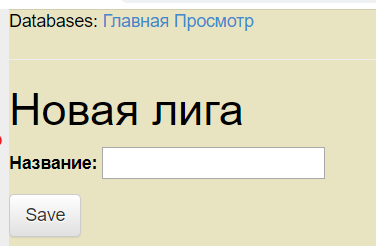
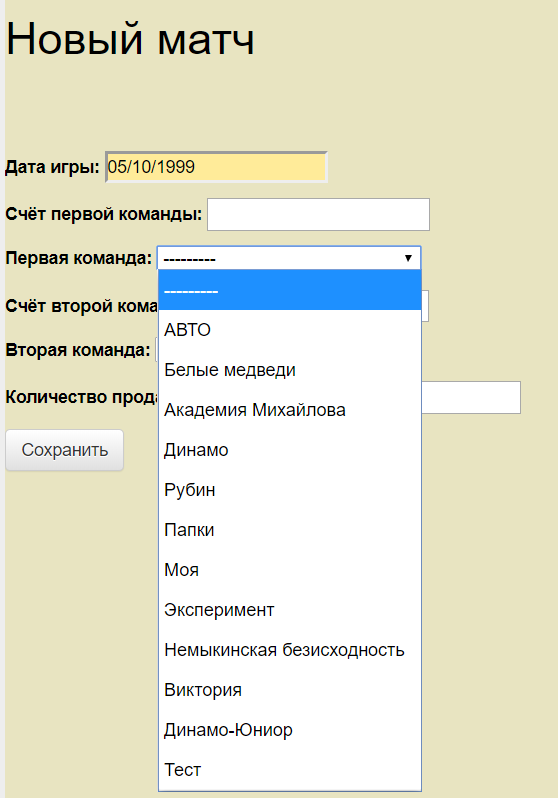
writer.writerow(line)

**Результат выполнения**









**Заключение:** В ходе выполнения курсовой работы были получены навыки программирования на языке Python, была освоена СУБД PostgreSQL, фреймворк Django. Получен опыт работы с веб-приложениями, html и css, csv.