

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ  
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ  
(ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**МНОГОПОЛЬЗОВАТЕЛЬСКОЕ ПРИЛОЖЕНИЕ  
ДЛЯ КОММУНИКАЦИИ**

Курсовая работа

по дисциплине «Современные концепции организации баз данных»

Пояснительная записка

Студент гр. 581-М

\_\_\_\_\_ А.Д. Андреянов

«\_\_\_» \_\_\_\_\_ 2022 г.

Руководитель

к.т.н. Доцент

каф. КСУП ТУСУР

\_\_\_\_\_ Н.Ю. Хабибулина

«\_\_\_» \_\_\_\_\_ 2022 г.

Томск 2022

## **Реферат**

Курсовой проект содержит 31 с., 8 рис., 3 табл.

**МНОГОПОЛЬЗОВАТЕЛЬСКОЕ ПРИЛОЖЕНИЕ ДЛЯ КОММУНИКАЦИИ, ЧАТ, СОЦИАЛЬНАЯ СЕТЬ.**

Пояснительная записка содержит концептуальную модель процесса «Многопользовательское приложение для коммуникации» и описание структуры реляционной базы данных, предназначенной для его информационной поддержки. Концептуальное моделирование выполнено с использованием методологии IDEF1X. Приложения содержат описания хранимых таблиц БД и формулировки типовых запросов к данным на SQL.

Функциональные схемы разработаны в приложении Erwin, приложение написано в IDE PyCharm, курсовой проект выполнен в текстовом редакторе Microsoft Word 2019

## Оглавление

1 Введение.....	4
2 Постановка задачи.....	5
2.1 Описание предметной области .....	5
2.2 Формализованное описание задачи.....	5
3 Концептуальная модель данных.....	6
3.1 Сущности и связи (ER - уровень) .....	6
3.2 Логика взаимосвязей данных (КВ-уровень) .....	6
3.3 Атрибуты и сущности (ФА-уровень).....	7
4 Глоссарий модели .....	8
5 Реализация приложения .....	14
6 Заключение .....	19
Приложение А .....	20
Приложение Б .....	26

## 1 Введение

Курсовой проект выполнен с целью практического освоения основных приемов и правил методологии информационного моделирования IDEF1X. В качестве предметной области разрабатываемой базы данных (БД) выбрано приложение для коммуникации. Оно нуждается в базе данных, которая будет хранить информацию о пользователях, регистрационные данные, диалоги, друзей сообщения и путь к фотографиям.

Предложенный в настоящей курсовой работе проект направлен на достижение указанных целей.

Основная часть пояснительной записки содержит описание компонентов, процессов и правил бизнеса. Концептуальная модель данных представлена в виде IDEF1X-диаграмм данных, показывающих сущности предметной области и выявляющих обусловленную правилами бизнеса логику связей между ними. Диаграммы сопровождаются глоссарием, содержащим формальные определения имен всех сущностей и хранимых элементов данных.

Рекомендуется следующий порядок чтения пояснительной записки:

- ознакомиться с описанием предметной области;
- внимательно изучить диаграмму ER-уровня и приведенные в глоссарии определения имен сущностей;
- изучить логику взаимосвязей сущностей, показанную на диаграмме KB-уровня;
- ознакомиться со структурами хранимых таблиц БД, представленными на диаграмме FA-уровня.

Приложение А содержит тексты команд создания хранимых таблиц. Синтаксис команд соответствует стандарту языка SQL с точностью до типов данных.

В Приложении Б приведены формулировки типовых запросов к данным.

## **2 Постановка задачи**

### **2.1 Описание предметной области**

Приложение должно хранить регистрационные данные пользователей, и использовать их для авторизации.

Пользователь, после регистрации может заполнить свои личные данные (в том числе добавлять фотографии), и в дальнейшем изменить их, может добавлять в друзья других пользователей, может отправлять сообщения, в рамках диалога, и прикреплять фотографии к сообщениям (в базе данных хранится путь к сообщению).

### **2.2 Формализованное описание задачи**

Задача: создать базу данных, удовлетворяющую потребностям и реализовать интерфейс, реализующий коммуникацию между пользователями и систему регистрации и авторизации.

Цель деятельности: создание многопользовательского приложения для коммуникации.

Выполняемые функции: регистрация, авторизация, получение, запись, хранение данных пользователей, а также данных о коммуникации.

Бизнес-правила: на одну электронную почту можно зарегистрировать одного пользователя, ограничение длины сообщений, определенная сложность пароля, ограниченная длина каждого поля личных данных, уникальный идентификатор пользователя.

Хранимые данные: личная информация о пользователе, данные для системы авторизации, данные о коммуникации.

Предполагаемые пользователи системы: пользователи интернета, заинтересованные в коммуникации между собой.

### 3 Концептуальная модель данных

#### 3.1 Сущности и связи (ER - уровень)

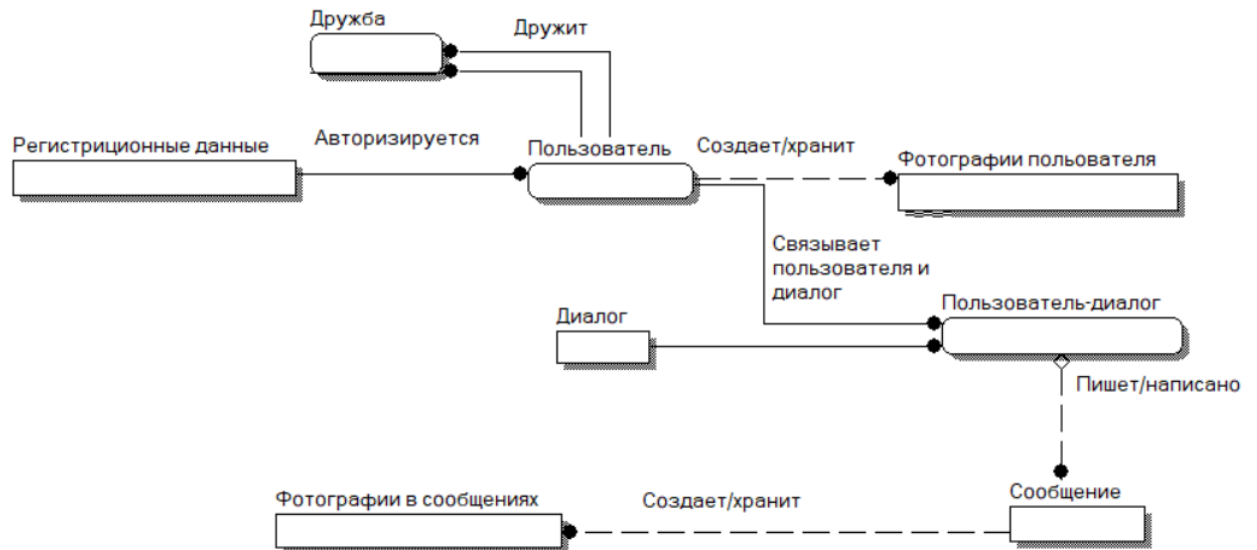


Рисунок 3.1 - ER диаграмма

#### 3.2 Логика взаимосвязей данных (КВ-уровень)

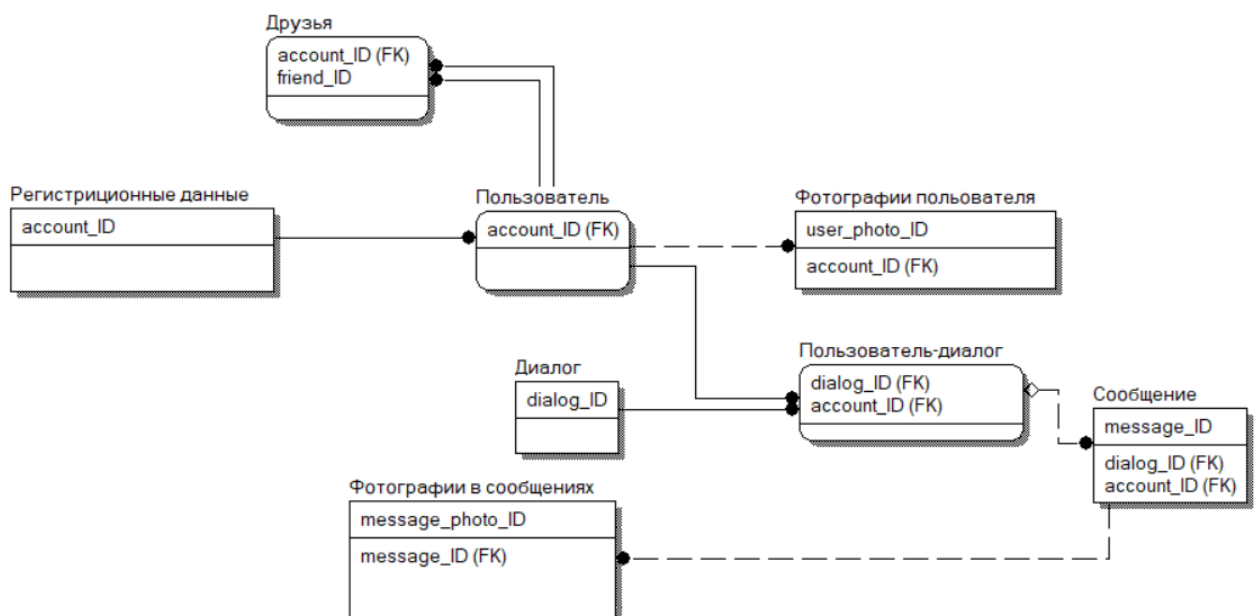


Рисунок 3.2 - КВ диаграмма

### 3.3 Атрибуты и сущности (ФА-уровень)

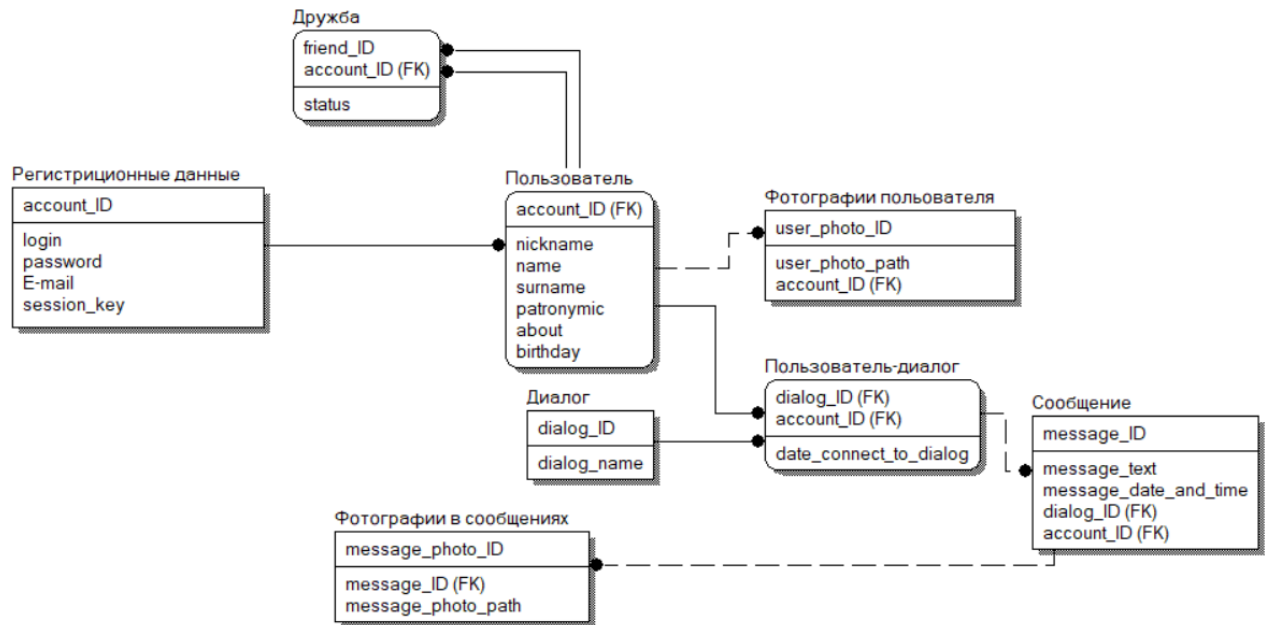


Рисунок 3.3 - ФА диаграмма

#### 4 Глоссарий модели

В таблице 1 представлены сущности БД ” Многопользовательское приложение для коммуникации”.

В таблице 2 представлены атрибуты БД ” Многопользовательское приложение для коммуникации”.

В таблице 3 представлены домены БД ” Многопользовательское приложение для коммуникации”.

Таблица 1 - Сущности

Имя	Определение
Регистрационные данные	Хранит РЕГИСТРАЦИОННЫЕ ДАННЫЕ пользователя, используются для авторизации
Пользователь	Хранит данные о ПОЛЬЗОВАТЕЛЕ
Дружба	Хранит список друзей данного ПОЛЬЗОВАТЕЛЯ
Фотографии пользователя	Хранит данные о фотографиях ПОЛЬЗОВАТЕЛЯ
Пользователь-диалог	Сущность, реализующая связь многие ко многим для ПОЛЬЗОВАТЕЛЯ и ДИАЛОГа
Диалог	ДИАЛОГ между двумя пользователями
Сообщение	Содержит данные сообщения в ДИАЛОГе от ПОЛЬЗОВАТЕЛЯ
Фотографии в сообщениях	Содержит путь к фотографии в СООБЩЕНИИ



Таблица 2 - Атрибуты

<b>Имя</b>	<b>Домен</b>	<b>Определение</b>	<b>Владелец</b>
account_ID	идентификационн ые номера	ID РЕГИСТРАЦИИ	РЕГИСТРАЦИОН НЫЕ ДАННЫЕ
login	логин	Логин ПОЛЬЗОВАТЕЛ я при РЕГИСТРАЦИИ	РЕГИСТРАЦИОН НЫЕ ДАННЫЕ
password	пароль	Пароль ПОЛЬЗОВАТЕЛ я при РЕГИСТРАЦИИ	РЕГИСТРАЦИОН НЫЕ ДАННЫЕ
E-mail	E-mail	E-mail ПОЛЬЗОВАТЕЛ я при РЕГИСТРАЦИИ	РЕГИСТРАЦИОН НЫЕ ДАННЫЕ
user_ID	идентификационн ые номера	ID ПОЛЬЗОВАТЕЛ я	ПОЛЬЗОВАТЕЛЬ
nickname	никнейм	Никнейм ПОЛЬЗОВАТЕЛ я	ПОЛЬЗОВАТЕЛЬ
name	имена	Имя ПОЛЬЗОВАТЕЛ я	ПОЛЬЗОВАТЕЛЬ
surname	имена	Фамилия ПОЛЬЗОВАТЕЛ я	ПОЛЬЗОВАТЕЛЬ

Продолжение таблицы 2 – Атрибуты

<b>Имя</b>	<b>Домен</b>	<b>Определение</b>	<b>Владелец</b>
patronymic	имена	Отчество ПОЛЬЗОВАТЕЛ я	ПОЛЬЗОВАТЕЛЬ
about	текст	Информация о ПОЛЬЗОВАТЕЛе	ПОЛЬЗОВАТЕЛЬ
status	статус	Статус ДРУжбы	ДРУЖБА
friendID	идентификационн ые номера	Ссылка на ДРУга	ДРУЖБА
birthday	даты	День рождения ПОЛЬЗОВАТЕЛ я	ПОЛЬЗОВАТЕЛЬ
user_photo_ID	идентификационн ые номера	ID ФОТОГРАФИИ ПОЛЬЗОВАТЕЛ я	ФОТОГРАФИИ ПОЛЬЗОВАТЕЛЯ
user_photo_path	путь	Путь к ФОТОГРАФИИ ПОЛЬЗОВАТЕЛ я	ФОТОГРАФИИ ПОЛЬЗОВАТЕЛЯ
date_connect_to_ dialog	дата и время	Дата вступления в диалог	ПОЛЬЗОВАТЕЛЬ- ДИАЛОГ
dialog_ID	идентификационн ые номера	ID ДИАЛОГа	ДИАЛОГ
dialog_name	имена	Имя ДИАЛОГа	ДИАЛОГ
message_ID	идентификационн ые номера	ID СООБЩЕНИЯ	СООБЩЕНИЕ

Продолжение таблицы 2 – Атрибуты

<b>Имя</b>	<b>Домен</b>	<b>Определение</b>	<b>Владелец</b>
message_text	текст	Текст СООБЩЕНИЯ	СООБЩЕНИЕ
message_date_and_time	дата и время	Дата и время отправки СООБЩЕНИЯ	СООБЩЕНИЕ
message_photo_ID	идентификационные номера	ID ФОТОГРАФИИ В СООБЩЕНИЯХ	ФОТОГРАФИИ В СООБЩЕНИЯХ
message_photo_path	путь	Путь к ФОТОГРАФИИ В СООБЩЕНИЯХ	ФОТОГРАФИИ В СООБЩЕНИЯХ

Таблица 3 - Домены

<b>Имя</b>	<b>Тип(&lt;длина&gt;)</b>	<b>Определение</b>
идентификационные номера	NUMERIC(10)	Последовательность цифр
имена	Char(30)	Последовательности букв русского алфавита, возможно, содержащие пробелы и дефисы.
логин	Char(30)	Последовательности букв английского алфавита, возможно, содержащие различные знаки, не содержит пробелы. Не может совпадать с ранее внесенными логинами.

Продолжение таблицы 3 - Домены

Имя	Тип(<длина>)	Определение
пароль	Char(60)	Последовательности букв английского алфавита, возможно, содержащие различные знаки, не содержит пробелы. Не может быть меньше 6 символов.
E-mail	Char(60)	Последовательности букв английского алфавита, возможно, содержащие различные знаки, не содержит пробелы. Обязательное содержит символ @.
nickname	Char(30)	Последовательности букв английского и (или) русского алфавита, может содержать пробелы и различные символы. Не может совпадать с ранее внесенными никнеймами.
текст	Char(1000)	Последовательности букв английского и (или) русского алфавита, может содержать пробелы и различные символы
дата и время	DATETIME(14)	Специальный числовой тип, интерпретируемый как <день><месяц><год><час><минута><секунда>.
Дата	DATE(8)	Специальный числовой тип, интерпретируемый как <день><месяц><год>.
путь	Char(100)	Последовательности букв английского алфавита, может содержать различные символы, используем в URL.

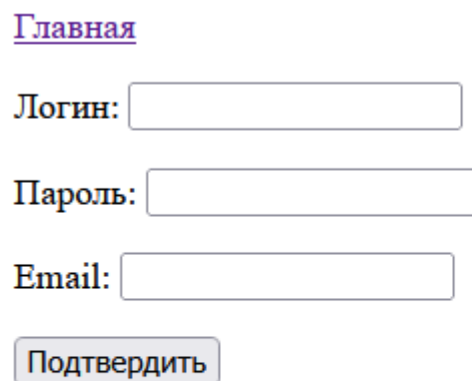
статус	BOOLEAN	Если заявка в друзья не принята – False, если принята - True
--------	---------	-----------------------------------------------------------------

## 5 Реализация приложения

Приложение реализовано при помощи языка программирования python, фреймворка Django, базы данных PostgreSQL. В данном приложении есть возможность регистрироваться на сайте, авторизоваться, а также отправлять сообщения и фотографии другим пользователям. Скриншоты приложения представлены на рисунках 1-6.

[Главная](#)  
[Регистрация](#)  
[Войти](#)

Рисунок 5.1 – Стартовая страница.



[Главная](#)

Логин:

Пароль:

Email:

Рисунок 5.2 – Окно для регистрации

[Главная](#)

Логин:

Пароль:

Рисунок 5.3 – Окно для авторизации

[Главная](#)

[Профиль](#)

Никнейм:

Имя:

Фамилия:

Отчество:

Информация о пользователе:

День рождения:

Рисунок 5.4 – просмотр и редактирование профиля пользователя

Друзья

[Никита](#)

Фото



Диалоги

[Никита и Николай](#)

[Выйти](#)

Рисунок 5.5 – друзья пользователя, фотографии и диалоги.



---

[Главная](#)

[Профиль](#)

Сообщения

**Nikita:** asas

*Dec. 30, 2021, 12:02 a.m.*

**Nikita:** asas

*Dec. 30, 2021, 12:02 a.m.*



**Nikita:** asas

*Dec. 30, 2021, 12:03 a.m.*

**Nikolai:** a

*Dec. 30, 2021, 12:07 a.m.*

**Nikolai:** s

*Dec. 30, 2021, 12:07 a.m.*



**Nikolai:** Привет

*Dec. 30, 2021, 1:34 a.m.*

MessageText:

отправить

[Выйти](#)

Рисунок 5.6 – диалог с другим пользователем.

Фреймворк Django позволяет напрямую не писать запросы к базе данных, а взаимодействовать с базой данных при помощи моделей. После создания

модели, Django автоматически создает API для работы с базой данных, который позволяет вам создавать, получать, изменять и удалять объекты.

Исходный код логики приложения представлен в приложении Б.

## **6 Заключение**

В результате выполнения курсового проекта была построена концептуальная модель БД «Многопользовательское приложение для коммуникации» и описана структура реляционной базы данных. Также были построены таблицы с детальным описанием всех сущностей, атрибутов и доменов и реализовано веб-приложение с использованием данной базы данных.

## Приложение А

(обязательное)

Описание таблиц

### А.1 Таблица соответствия логических и физических имен

<b>Физическое имя</b>	<b>Логическое имя</b>
RegistrationData	Регистрационные данные
accountID	account ID
login	login
password	password
Email	E-mail
User	Пользователь
nickname	nickname
name	name
surname	surname
patronymic	patronymic
about	about
birthday	birthday
friendID	friendID
UserPhoto	Фотографии пользователя
userPhotoID	user_photo_ID
userPhotoPath	user_photo_path
UserDialog	Пользователь-диалог
Dialog	Диалог
dialogID	dialog_ID
date_connect_to_dialog	date_connect_to_dialog
dialogName	dialog_name
Message	Собобщение
messageID	message_ID

Продолжение таблицы А.1 Таблица соответствия логических и физических имен

Физическое имя	Логическое имя
messageText	message_text
messageDateAndTime	message_date_and_time
PhotoInMessage	Фотографии в сообщениях
messagePhotoID	message_photo_ID
messagePhotoPath	message_photo_path

SQL команды создания таблиц БД

```
CREATE TABLE RegistrationData (
    accountID      NUMERIC(10) NOT NULL,
    login          CHAR (30) NOT NULL,
    password       CHAR (30) NOT NULL,
    Email          CHAR (30) NOT NULL,
    PRIMARY KEY (accountID)
);
```

```
CREATE TABLE User (
    userID         NUMERIC(10) NOT NULL,
    nickname       CHAR (30) NOT NULL,
    name           CHAR (30) NOT NULL,
    surname        CHAR (30) NOT NULL,
    patronymic     CHAR (30),
    about          CHAR (1000),
    birthday       DATE (8),

    PRIMARY KEY (accountID)
    FOREIGN KEY (accountID)
```

REFERENCES RegistrationData  
ON DELETE SET NULL

);

CREATE TABLE Friends (

Status BOOLEAN NOT NULL,

PRIMARY KEY (accountID)

PRIMARY KEY (fiendID)

FOREIGN KEY (accountID)

REFERENCES User

ON DELETE SET NULL

FOREIGN KEY (fiendID)

REFERENCES User

ON DELETE SET NULL

);

CREATE TABLE UserPhoto (

userPhotoID NUMERIC(10) NOT NULL,

userPhotoPath CHAR (100) NOT NULL,

userID NUMERIC(10) NOT NULL,

PRIMARY KEY (userPhotoID),

FOREIGN KEY (userID)

REFERENCES User

ON DELETE SET NULL

);

CREATE TABLE UserDialog (

userID NUMERIC(10) NOT NULL,

dialogID NUMERIC(10) NOT NULL,

```

        dateConnectToDialog DATETIME (14) NOT NULL
        PRIMARY KEY (userID),
        PRIMARY KEY (dialogID),
        FOREIGN KEY (userID)
            REFERENCES User
            ON DELETE SET NULL
        FOREIGN KEY (dialogID)
            REFERENCES Dialog
            ON DELETE SET NULL
    );

    CREATE TABLE Dialog (
        dialogID          NUMERIC(10) NOT NULL,
        dialogName        CHAR (30) NOT NULL,
        PRIMARY KEY (dialogID)
    );

    CREATE TABLE Message (
        messageID          NUMERIC(10) NOT NULL,
        messageText        CHAR (1000) NOT NULL,
        messageDate-AndTime DATETIME (14) NOT NULL
        dialogID          NUMERIC(10) NOT NULL,
        PRIMARY KEY (messageID),
        FOREIGN KEY (dialogID)
            REFERENCES Dialog
            ON DELETE SET NULL
        FOREIGN KEY (userID)
            REFERENCES UserDialog
            ON DELETE SET NULL
    );

    CREATE TABLE PhotoInMessage (

```

```
messagePhotoID          NUMERIC(10) NOT NULL,  
messagePhotoPath  CHAR (100) NOT NULL,  
messageID          NUMERIC(10) NOT NULL,  
PRIMARY KEY (messagePhotoID),  
FOREIGN KEY (messageID)  
REFERENCES Message  
ON DELETE SET NULL  
);
```



## Приложение Б

ТИПОВЫЕ ЗАПРОСЫ К БД "Многопользовательское приложение для коммуникации"

Запрос для подсчета сообщений в диалоге

```
SELECT COUNT(*)  
FROM Dialog, Message  
WHERE (Dialog. dialogID =Message. dialogID) and  
(dialogID ='0000000001')
```

Запрос для вывода Информации о пользователе, сгруппированной по годам.

```
SELECT accountID, nickname, birthday  
FROM User  
group by  
cube (birthday)
```

Информации о фотографиях пользователя, сгруппированная по нику пользователя.

```
SELECT nickname, userPhotoID, userPhotoPath  
FROM User, UserPhoto  
WHERE (UserPhoto. accountID =User. accountID)  
group by  
rollup (nickname)
```

## Приложение В

(обязательное)

## Исходный код логики приложения

Исходный код части с логикой:

```

def RegistrationPage(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            print(form.cleaned_data)

            form.save()
            return HttpResponseRedirect(reverse('loginPage',))
    else:
        form = RegistrationForm()
    return render(request, 'main/registration.html',
                  {'form': form})

def LoginPages(request):
    if request.method == 'POST':
        request = request.POST
        print(list(request.values())[1])
        print(list(request.values())[2])

        verificate      =      RegistrationData.objects.filter(Логин      =
list(request.values())[1], Пароль=list(request.values())[2])
        if verificate:
            return redirect('/main')

    return redirect('/main/login')

```

```
else:
```

```
    form = LoginForm()
```

```
    return render(request, 'main/login.html',
```

```
        {'form': form})
```

```
def LoginPage(request):
```

```
    loginForm = LoginForm()
```

```
    login = ''
```

```
    sessionKey = request.session._get_or_create_session_key()
```

```
    print(sessionKey)
```

```
    if request.method == 'GET':
```

```
        if 'action' in request.GET:
```

```
            action = request.GET.get('action')
```

```
            if action == 'logout':
```

```
                registrationObject
```

```
RegistrationData.objects.get(sessionKey=sessionKey)
```

```
                print(registrationObject)
```

```
                registrationObject.sessionKey = ''
```

```
                registrationObject.save()
```

```
                return render(request, 'main/login.html', {
```

```
                    'form': loginForm,
```

```
                    'login': login
```

```
                })
```

```
    if RegistrationData.objects.filter(sessionKey=sessionKey):
```

```
        login = sessionKey
```

```
    elif request.method == 'POST':
```

```
        loginForm = LoginForm(request.POST)
```

```
        if loginForm.is_valid():
```

```
            login = loginForm.cleaned_data['login']
```

```

        password = loginForm.cleaned_data['password']
        if RegistrationData.objects.filter(Логин = login.strip(),
Пароль=password.strip()):
            print(RegistrationData.objects.get(Логин = login.strip(),
Пароль=password.strip()))
            registrationObject = RegistrationData.objects.get(Логин =
login.strip(), Пароль=password.strip())
            registrationObject.sessionKey = sessionKey
            registrationObject.save()
            return HttpResponseRedirect(reverse('userPage'))
        else:
            login = ''

    return render(request, 'main/login.html', {
        'form': loginForm,
        'login': login
    })

def UserPage(request):
    sessionKey = request.session._get_or_create_session_key()
    form = UserForm()
    friends = []
    photos = []
    dialogs = []
    if RegistrationData.objects.filter(sessionKey=sessionKey):
        sessionKey = sessionKey
        objectRegistrationData =
RegistrationData.objects.get(sessionKey=sessionKey)
        print(type(objectRegistrationData))

```

#если создан

```
if User.objects.filter(userID=objectRegistrationData):
```

```
    formTMP = User.objects.get(userID=objectRegistrationData)
```

```
    if UserDialog.objects.filter(userID=formTMP):
```

```
        print("ssas")
```

```
        dialogs = UserDialog.objects.filter(userID=formTMP)
```

```
        print(dialogs)
```

```
    if UserPhoto.objects.filter(userID=formTMP):
```

```
        photos = UserPhoto.objects.filter(userID=formTMP)
```

```
    print(formTMP)
```

```
    form = UserForm(instance=formTMP)
```

```
if request.method == 'POST':
```

```
    form = UserForm(request.POST, instance=formTMP )
```

```
    if form.is_valid():
```

```
        form.save()
```

```
if Friends.objects.filter(userID=formTMP):
```

```
    friends = Friends.objects.filter(userID=formTMP)
```

```
elif request.method == 'POST':
```

```
    req = request.POST.dict()
```

```
    req['userID'] = objectRegistrationData
```

```
    print(req)
```

```
    form = UserForm(req)
```

```
    if form.is_valid():
```

```

        form.save()
        #return HttpResponseRedirect(reverse('index',))

    else:

        form = UserForm()

    return render(request, 'main/user.html',
                  {'form': form, 'friends':friends, 'photos':photos,'dialogs':dialogs})

def DialogPage(request, dialogID):
    sessionKey = request.session._get_or_create_session_key()
    listUserDialog = UserDialog.objects.filter(dialogID=dialogID)
    form = MessageForm()
    messages = []
    if RegistrationData.objects.filter(sessionKey=sessionKey):
        currentUser =
User.objects.get(userID=RegistrationData.objects.get(sessionKey=sessionKey))
        otherUser = UserDialog.objects.get(~Q(userID = currentUser)).userID
        messages = Message.objects.filter(dialogID = dialogID)
        photos = PhotoInMessage.objects.all()

    if request.method == 'POST':
        req = request.POST.dict()
        req['userID'] = currentUser
        req['dialogID'] = dialogID
        form = MessageForm(req)
        if form.is_valid():

```

```
form.save()
```

```
form = MessageForm()
```

```
return render(request, 'main/dialog.html',  
              {'form': form, 'messages': messages, 'photos': photos})
```