МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий
Кафедра Информационных систем и технологий
Специальность 1-40 01 01 «Программное обеспечение информационных технологий»
Специализация Программирование интернет-приложений

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

Веб-приложение «Парикмахерская» с технологией полнотекстового поиска.

Выполнил студент Теханов Андрей Викторович
(Ф.И.О.)

Руководитель проекта асс. Нистюк О.А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Нистюк О.А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Нистюк О.А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Минск 2021

Оглавление

Введение

Данный курсовой работы посвящен созданию Веб-приложения и моделирования базы данных для парикмахерской. Актуальность темы заключается в том, что с развитием информационных технологий и телекоммуникаций жизнь становится все более мобильной и информативной, новые технологии прочно входят в различные сферы жизни, отрасли хозяйствования и несут новые нормы в них. Следуя вышеперечисленному, на рынке сферы услуг бизнес успешно осуществляет цифровизацию своих сервисов, чтобы сделать их более доступными, удобными, а также уменьшить издержки работы.

Целью данного курсового проекта является разработка базы данных в PostgreSQL соответствующей требованиям и пользовательского Веб-интерфейса.

Для осуществления обозначенной цели необходимы реализации следующих задач:

– авторизация и регистрация пользователей;
– разделение пользователей на клиента, администратора и мастера;
– добавление заявок на обслуживание;
– удаление редактирование и просмотр записи;
– просмотр заявок;
– просмотр информации о предоставляемых сервисах.

При написание курсового проекта использовались общетеоретические и практические методы.

Постановка задачи

Согласно требованиям к курсовому проекту, были сформулированы и поставлены такие задачи как выделение основных сущностей базы данных, описание их соответствующими атрибутами, проектирование необходимых связей. После этапа проектирования, реализация описанных сущностей в виде объектов базы данных.

Необходимо написать соответствующие процедуры для доступа к данным. При необходимости, реализовать триггеры, представления и функции. Реализовать импорт и экспорт данных в CSV формате. Протестировать производительность базы данных. Применить технологию полнотекстового поиска.

Функционально должны быть выполнены следующие задачи:

- регистрация и авторизация пользователей;
- редактирование личного профиля;
- добавление записей на обслуживание;
- удаление, редактирование и просмотр записей;
- просмотр доступных мастеров и соответствующих услуг;
- просмотр информации о парикмахерской.

1. Аналитический обзор литературы

Для примера я выбрал сайт gorod-krasoty.by, так как он имеет близкую тему к моему проекту и база данных будет проектироваться на основе данного сайта.

Основная идея заключается в том, чтобы было удобно использовать приложение как клиентам, так и персоналу парикмахерской. Задача – спроектировать базу данных, которая будет соответствовать проекту, который я описал выше, представленному на рисунке 1.1

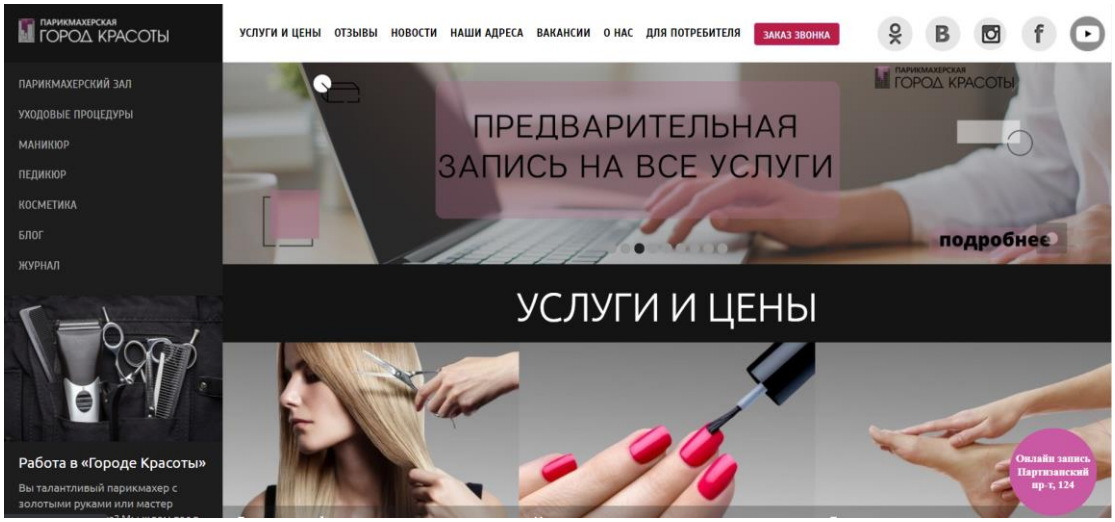


Рисунок 1.1 – Главная страница сайта gorod-krasoty.by

Сайт содержит в себе разграничение по категориям услуг, поиск, форму оформления заказа, а также возможность зарегистрироваться или авторизоваться на сайте.

Просмотрев ещё несколько сайтов, можно сделать вывод, что все сайты по данной тематике имеют одинаковый функционал. Они предоставляют возможность выбрать услугу, оформить заявку, а также поиск и регистрация/авторизация.

Моя задача в том, чтобы сделать такую базу данных, чтобы она в последующем была готова к эксплуатации любым салоном красоты, который захочет ей воспользоваться.

2. Разработка модели базы данных

Разработка модели базы данных – это набор процессов, которые облегчают проектирование, разработку, внедрение и обслуживание систем управления данными.

Правильно спроектированная база данных проста в обслуживании, улучшает согласованность данных и экономичная с точки зрения дискового пространства для хранения. Разработчик решает, как элементы данных соотносятся и какие данные должны быть сохранены. Для проектирования баз данных используются системы управления базами данных. Система управления базами данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Одной из самых популярных СУБД является PostgreSQL. PostgreSQL — свободная объектно-реляционная система управления базами данных, основанная на языке SQL. Рассмотрим основные характеристики данной СУБД, которую я выбрал для выполнения данного курсового проекта:

– возможность создания пользовательских типов данных;
– разные типы индексов;
– поддерживает транзакции;
– высокая производительность и надёжность.

Для базы данных были разработаны 5 таблиц таких как users, visits, services, visit_services, master_services. Диаграмма связей таблиц для необходимой базы данных представлена в приложении А.

Таблица users содержит пользователей приложения, состоит из следующих столбцов:

– id — уникальный идентификатор, первичный ключ;
– first_name — имя пользователя;
– last_name — фамилия пользователя;
– email — адрес электронной почты пользователя;
– phone_number — телефонный номер пользователя;
– encrypted_password — пароль пользователя;
– type — тип пользователя: admin, client, master;
– created_at — дата добавления записи в таблицу;
– updated_at — дата обновления записи в таблице;
– confirmed_at — дата подтверждения регистрации по почте;
– confirmed_ sent_at — дата отправки письмя для регистрации на почту;
– confirmation_token — токен подтверждения;
– reset_password_token — токен сброса пароля;
– resent_password_sent_at — дата отпраки нового пароля.

Поле type таблицы users необходимо для реализации паттерна проектирования STI. STI (Single Table Inheritance) — паттерн проектирования, который позволяет перенести объектно-ориентированное наследование на таблицу реляционной базы данных. В таблице БД должно присутствовать поле идентифицирующее название класса в иерархии.

Таблица visits содержит заявки пользователей на посещение парикмахерской:

– id — уникальный идентификатор, первичный ключ;

– client_id — уникальный идентификатор клиента, внешний ключ на поле id таблицы users;

– master_id — уникальный идентификатор мастера, оказывающего услугу;

– state — текущие состояние заявки;

– date — дата заявки;

– addition — описание к заявке от клиента;

– created_at — дата добавления записи в таблицу;

– updated_at — дата обновления записи в таблице.

Таблица services содержит услуги, предоставляемые мастерами парикмахерской:

– id — уникальный идентификатор;

– service_name — название услуги;

– price_cents — стоимость услуги;

– created_at — дата добавления записи в таблицу;

– updated_at — дата обновления записи в таблице.

Связующая таблица master_services предназначена для реализации связи многие ко многим между таблицами masters и services, состоит из следующих столбцов:

– id — уникальный идентификатор, первичный ключ;

– service_id — уникальный идентификатор сервиса, внешний ключ на поле id таблицы services;

– master_id — уникальный идентификатор мастера, внешний ключ на поле id таблицы masters;

– created_at — дата добавления записи в таблицу;

– updated_at — дата обновления записи в таблице.

Связующая таблица service_visits предназначена для реализации связи многие ко многим между таблицами services и visits, состоит из следующих столбцов:

– id — уникальный идентификатор, первичный ключ;

– service_id — уникальный идентификатор сервиса, внешний ключ на поле id таблицы services;

– visit_id — уникальный идентификатор заявки, внешний ключ на поле id таблицы visits;

– created_at — дата добавления записи в таблицу;

– updated_at — дата обновления записи в таблице.

Модель базы данных, разрабатываемой в рамках курсового проекта, представлена на рисунке 3.1.

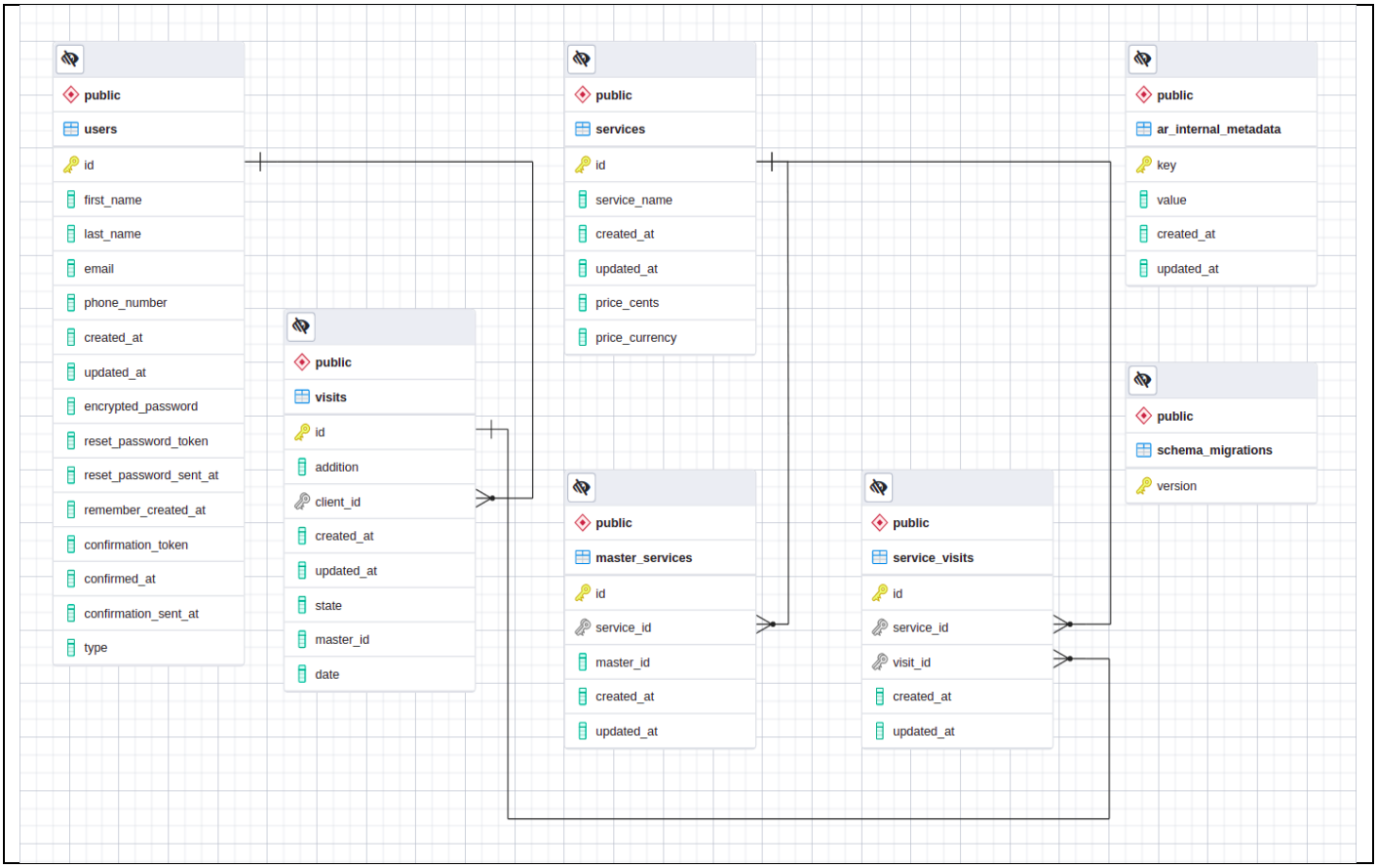


Рисунок 3.1 – Структура модели базы данных

В итоге разработки базы данных была получена модель, приведенная на рисунке 1.1 выше. Данная модель полностью покрывает область курсового проекта.

Скрипты создания всех таблиц базы данных представлены в приложении А.

В следующих разделах будут описаны этапы разработки реализации объектов вместе с внутренней логикой работы.

3. Разработка необходимых объектов

2.1. Таблицы

В ходе выполнения был разработан набор таблиц, описанных во второй главе. Для правильного хранения созданных таблиц, необходимо, чтобы хранящиеся данные подчинялись некоторым правилам.

Данного эффекта можно достичь за счет определения конкретного типа данных хранящихся в строках таблиц и установки необходимых ограничений целостности.

Таким образом, разработка и применение ограничений приведет к общему дизайну базы данных и поможет обеспечить гибкость при необходимости обновлений.

Скрипты создания всех таблиц базы данных представлены в приложении Б.

2.2. Пользователи

Пользователь базы данных – это какое-либо лицо, имеющее доступ к БД и пользующееся услугами информационной системы для получения информации.

При проектировании базы данных были использованы 3 пользователя. Первый пользователь – Client – потребитель услуг. Через пользовательский интерфейс имеет доступ для чтения таблиц, связанных с визитами, а также к изменению данных о своих визитах и о себе.

Второй пользователь – Admin – обладает более обширным перечнем прав. Имеет права для чтения, изменения и записи в таблицах.

Третий пользователь – Master – обладает правами аналогичными первому пользователю.

2.3. Процедуры

В требованиях к проекту было указано, что одним из способов доступа к данных должны быть процедуры.

Такой подход означает, что требуется вынести необходимый функционал взаимодействия с базой данных в отдельные процедуры.

Таким образом, за счет того, что взаимодействие с базой данных происходит по средствам вызова процедур, мы, как разработчики, можем более детально настроить доступ к хранящимся данным. Ограничивать и контролировать доступ к данным.

В ходе курсового проекта был разработан набор процедур. Скрипт создания процедур приведен в приложении В.

2.4. Функции

Функция — это подпрограмма, которая исполняет заранее заготовленный блок кода и возвращает результат.

В PostgreSQL только функции могут возвращать результирующий набор, предоставляя этим более гибкий функционал, чем процедуры. Поэтому выборка данных из таблиц реализована через соответствующие функции.

Скрипты создания функций приведены в приложении Г.

2.5. Индексы

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени.

В моём случае индексы были созданы автоматически при создании таблиц и указания constraint primary key.

4. Описание процедур импорта и экспорта данных

Для импорта и экспорта данных был выбран CSV формат, т.к. он поддерживается СУБД PostgreSQL.

CSV — текстовый формат, предназначенный для представления данных в виде таблицы. В таких файлах для разделения текста на колонки используется специальный символ — разделитель. Файлы импорта и экспорта будут представлять из себя такой файлы, чьи значения будут разделены запятыми, а каждая строка будет соответствовать новой записи в таблице. Формат CSV имеет такое важное преимущество как маленький объём занимаемой памяти. Место на диске не тратится на определение тегов, как например в XML.

5. Технология полнотекстового поиска

Многие СУБД поддерживают методы полнотекстового поиска (Fulltext search), которые позволяют очень быстро находить нужную информацию в больших объемах текста.

В отличие от оператора LIKE, такой тип поиска предусматривает создание соответствующего полнотекстового индекса, который представляет собой своеобразный словарь упоминаний слов в полях. Под словом обычно понимается совокупность из не менее 3-х не пробельных символов (но это может быть изменено). В зависимости от данных словаря может быть вычислена релевантность – сравнительная мера соответствия запроса найденной информации.

В данном проекте я использовал такую технологию полнотекстового поиска как Elasticssearch.

Elasticsearch - это масштабируемый распределенный поисковый сервер на основе Lucene. Данная технология скрывает сложность Lucene и предоставляет интерфейсы Restful для управления индексированием и поиском. А также поддерживает многопользовательский доступ, совместное использование одних и тех же системных или программных компонентов в многопользовательской среде и обеспечивает изоляцию данных между пользователями.

Некоторые преимущества данной технологии:

– Обладая хорошей масштабируемостью, он может развертывать сотни кластеров серверов для обработки петабайт данных.

– Индексировать данные и искать данные практически в реальном времени.

5. Тестирование производительности базы данных

Для тестирования производительности была взята за основу таблица services, а результаты работы демонстрируются графическим клиентом для работы с сервером pgAdmin.

Изначально таблица services была заполнена 100000 запсями.

Выполним запрос, выводящий все записи данной таблицы.

```
select * from Services;
```

Листинг 5.1 – Запрос

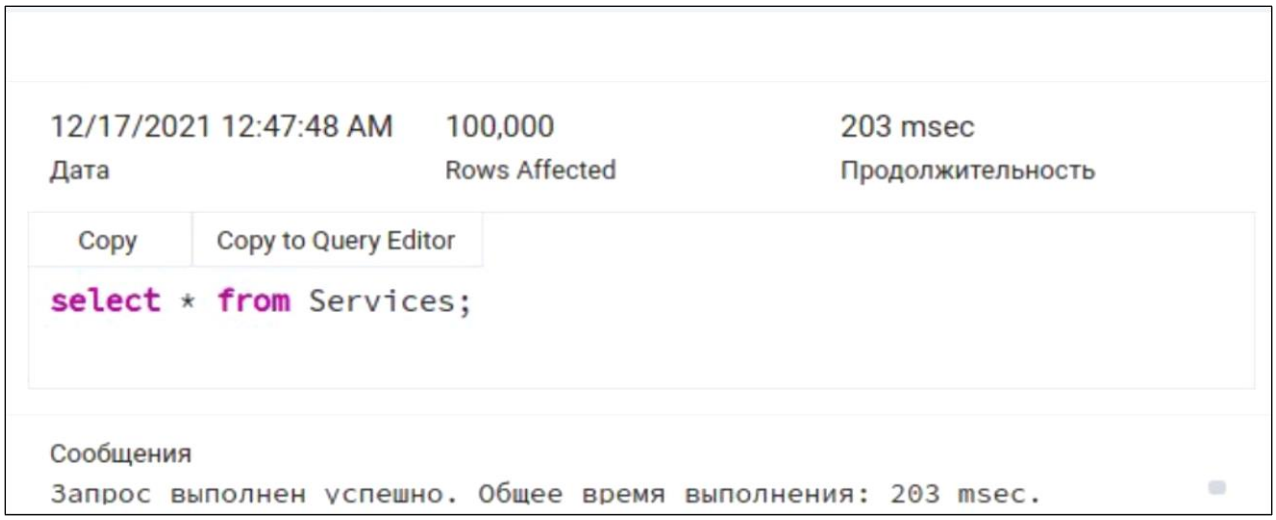


Рисунок 5.1 – Вывод скорости выполнения запроса

Теперь проведем тестирование производительности функции, выводщей все записи таблицы Services. Функция AllServices() представленна в листинге 5.2.

```
CREATE or REPLACE FUNCTION AllServices ()
	returns table (
		name_service varchar,
		service_price integer
	)
	LANGUAGE plpgsql
as $$
begin
	return query
		select
			service_name,
```

```
                price_cents
            from
                Services;
end;$$
```

Листинг 5.2 – скрипт функции AllServices()

Вызовем функцию AllServices() для получения данных о ходе ее выполнения

```
select * from AllServices();
```

Листинг 5.3 – вызов функции AllServices()

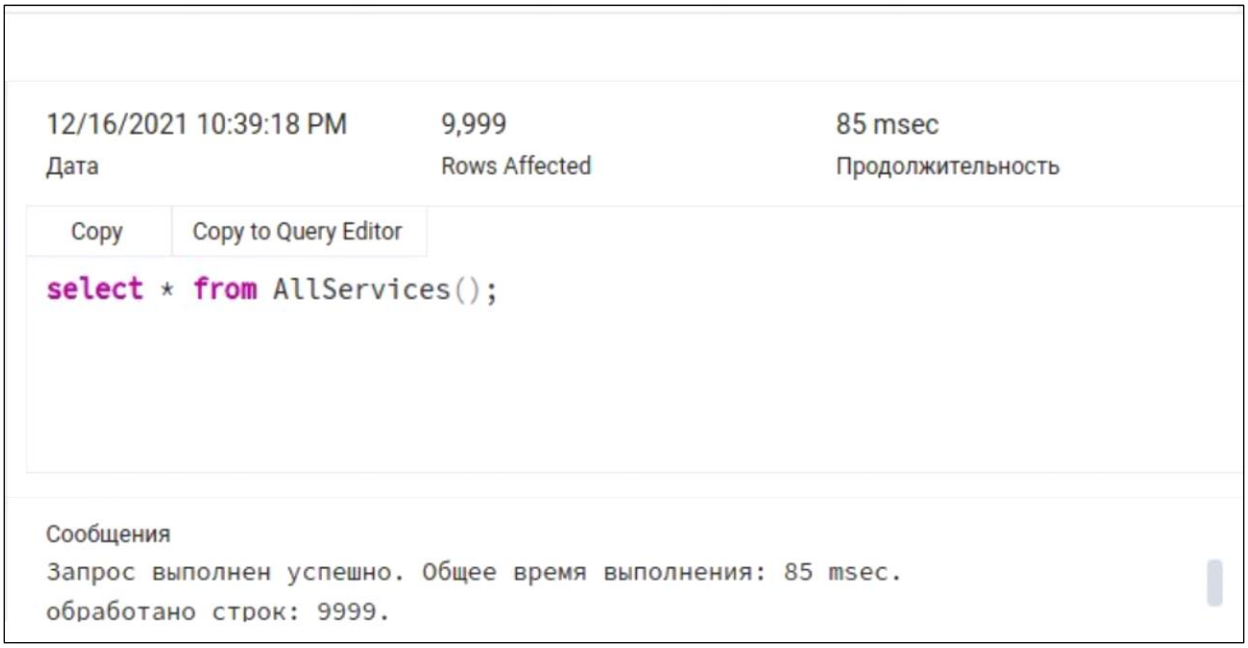


Рисунок 5.2 – Вывод скорости выполнения вызова функции AllServices()

Данная скорость запроса достигнута благодаря оптимизации архитектуры базы данных путем внедрения индексов. Например, таблице services соответсвует индекс «index_services_on_service_name».

Таким образом работа базы данных была оптимизированна с применением индексов, позволяющие выполнять выборку записей таблице числом в 100000 строк за 85 msec секунды.

6. Руководство пользователя

Для взаимодействия пользователя с базой данных было разработано программное средство, использующее соответствующие функции и процедуры. Список процедур и функций доступных пользователю представлен ниже, скрипты данных объектов базы данных расположены в приложении Г.

Функции:

– AllMasters ()

– AllServices ()

– AllUsers ()

– AllVisits ()

– VisitsById (integer)

– VisitMasterFullName (integer)

Процедуры:

– DeleteUser(integer)

– UpdateUser("a" character varying, "b" character varying, "c" character varying, "d" character varying, e integer)

– UpdateVisit("a" timestamp without time zone, "b" character varying, c integer)

– DeleteVisit (integer)

Заключение

В ходе выполнения курсового проекта мной были реализованы поставленные задачи курсового проекта по созданию Веб-приложения «Парикмахерская» с технологией полнотекстового поиска и соответствующей базой данных.

Также я получил как опыт проектирования, так и опыт разработки баз данных, создания Веб-интерфейса для работы с пользователем.

В процессе работы с базой данных мной были изучены способы полнотекстового поиска. Разработаны процедуры и функции, осуществляющие доступ к данным.

При разработке проекта выполнены следующие пункты:

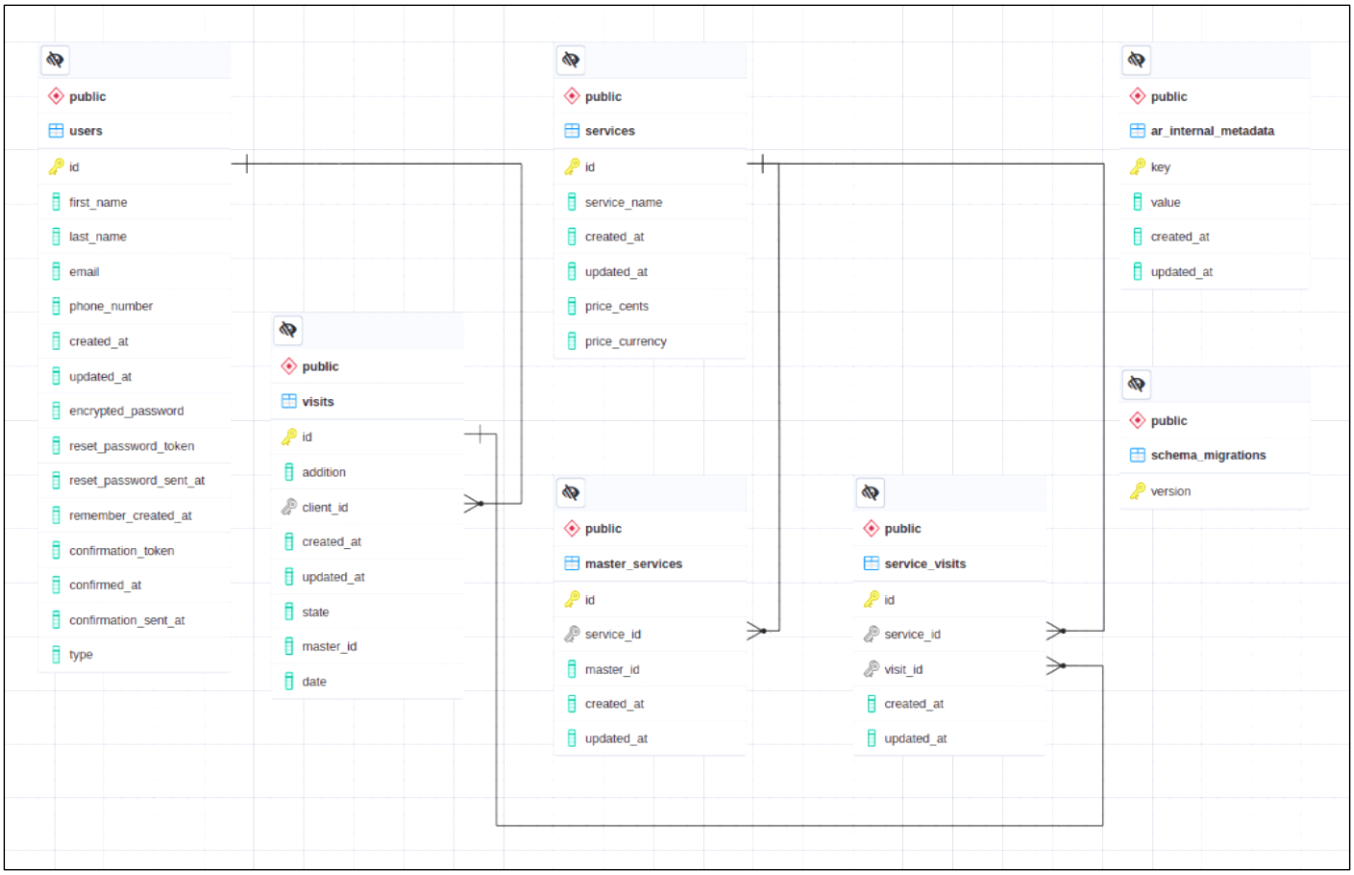
- регистрация и авторизация пользователей;
- редактирование личного профиля;
- добавление записей на обслуживание;
- удаление, редактирование и просмотр записей;
- просмотр доступных мастеров и соответствующих услуг;
- просмотр информации о парикмахерской.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объёме.

Список использованной литературы

1. Сайт gorod-krasoty.by – https://gorod-krasoty.by/. – Дата доступа: 22.1.2021.
2. Хранимые процедуры – https://postgrespro.ru/docs/postgresql/11/sql-createprocedure/. – Дата доступа: 22.11.2021.
3. Функции – https://postgrespro.ru/docs/postgresql/9.6/sql-createfunction/. – Дата доступа: 22.11.2021.
4. Экспорт и импорт в CSV – https://skyvia.com/blog/complete-guide-on-how-to-import-and-export-csv-files-to-postgresql/. – Дата доступа: 22.11.2021.
5. Сайт rubyonrails.org – https://guides.rubyonrails.org/. – Дата доступа: 22.11.2021.
6. Сайт postgresql.org – https://www.postgresql.org/. – Дата доступа: 22.11.2021.

Приложение А



Приложение Б

```
CREATE TABLE IF NOT EXISTS public.master_services
(
    id bigint NOT NULL DEFAULT nextval('master_services_id_seq'::regclass),
    service_id integer,
    master_id integer,
    created_at timestamp(6) without time zone NOT NULL,
    updated_at timestamp(6) without time zone NOT NULL,
    CONSTRAINT master_services_pkey PRIMARY KEY (id),
    CONSTRAINT fk_rails_9426e2aa9b FOREIGN KEY (service_id)
        REFERENCES public.services (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public.master_services
    OWNER to andrew;
```

```
CREATE TABLE IF NOT EXISTS public.service_visits
(
    id bigint NOT NULL DEFAULT nextval('service_visits_id_seq'::regclass),
    service_id integer,
    visit_id integer,
    created_at timestamp(6) without time zone NOT NULL,
    updated_at timestamp(6) without time zone NOT NULL,
    CONSTRAINT service_visits_pkey PRIMARY KEY (id),
    CONSTRAINT fk_rails_7380cdb842 FOREIGN KEY (service_id)
        REFERENCES public.services (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_rails_acfbfd2570 FOREIGN KEY (visit_id)
        REFERENCES public.visits (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public.service_visits
    OWNER to andrew;
```

```
CREATE TABLE IF NOT EXISTS public.services
(
    id bigint NOT NULL DEFAULT nextval('services_id_seq'::regclass),
    service_name character varying COLLATE pg_catalog."default" NOT NULL,
    created_at timestamp(6) without time zone NOT NULL,
    updated_at timestamp(6) without time zone NOT NULL,
    price_cents integer NOT NULL DEFAULT 0,
    price_currency character varying COLLATE pg_catalog."default" NOT NULL DEFAULT 'USD'::character varying,
    CONSTRAINT services_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE public.services
    OWNER to andrew;
-- Index: index_services_on_service_name

-- DROP INDEX public.index_services_on_service_name;

CREATE UNIQUE INDEX index_services_on_service_name
    ON public.services USING btree
    (service_name COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```
CREATE TABLE IF NOT EXISTS public.users
(
    id bigint NOT NULL DEFAULT nextval('users_id_seq'::regclass),
    first_name character varying COLLATE pg_catalog."default" NOT NULL,
    last_name character varying COLLATE pg_catalog."default" NOT NULL,
    email character varying COLLATE pg_catalog."default" NOT NULL,
    phone_number character varying COLLATE pg_catalog."default" NOT NULL,
    created_at timestamp(6) without time zone NOT NULL,
    updated_at timestamp(6) without time zone NOT NULL,
    encrypted_password character varying COLLATE pg_catalog."default" NOT NULL DEFAULT ''::character varying,
    reset_password_token character varying COLLATE pg_catalog."default",
    reset_password_sent_at timestamp without time zone,
    remember_created_at timestamp without time zone,
    type character varying COLLATE pg_catalog."default" NOT NULL DEFAULT 'Client'::character varying,
    confirmation_token character varying COLLATE pg_catalog."default",
    confirmed_at timestamp without time zone,
    confirmation_sent_at timestamp without time zone,
    CONSTRAINT users_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE public.users
    OWNER to andrew;
-- Index: index_users_on_confirmation_token

-- DROP INDEX public.index_users_on_confirmation_token;
```

```
CREATE UNIQUE INDEX index_users_on_confirmation_token
    ON public.users USING btree
    (confirmation_token COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: index_users_on_email

-- DROP INDEX public.index_users_on_email;

CREATE UNIQUE INDEX index_users_on_email
    ON public.users USING btree
    (email COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: index_users_on_reset_password_token

-- DROP INDEX public.index_users_on_reset_password_token;

CREATE UNIQUE INDEX index_users_on_reset_password_token
    ON public.users USING btree
    (reset_password_token COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```sql
CREATE TABLE IF NOT EXISTS public.visits
(
    id bigint NOT NULL DEFAULT nextval('visits_id_seq'::regclass),
    addition character varying COLLATE pg_catalog."default",
    client_id bigint NOT NULL,
    created_at timestamp(6) without time zone NOT NULL,
    updated_at timestamp(6) without time zone NOT NULL,
    state character varying COLLATE pg_catalog."default" NOT NULL DEFAULT 'sent'::character varying,
    master_id bigint NOT NULL,
    date timestamp without time zone,
    CONSTRAINT visits_pkey PRIMARY KEY (id),
    CONSTRAINT fk_rails_09e5e7c20b FOREIGN KEY (client_id)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public.visits
    OWNER to andrew;
-- Index: index_visits_on_client_id

-- DROP INDEX public.index_visits_on_client_id;

CREATE INDEX index_visits_on_client_id
    ON public.visits USING btree
    (client_id ASC NULLS LAST)
    TABLESPACE pg_default;
```

Приложение В

```
CREATE or REPLACE PROCEDURE DeleteUser(integer)
LANGUAGE SQL
AS $$
    DELETE FROM
        Users
    WHERE
        id = $1;
$$;
select * from Users;
call DeleteUser(3);
```

```
CREATE or REPLACE PROCEDURE UpdateUser("a" character varying, "b" character varying,
"c" character varying, "d" character varying, e integer)
LANGUAGE SQL
AS $$
    UPDATE
        Users
    SET
        first_name = "a",
        last_name = "b",
        email = "c",
        phone_number = "d"
    WHERE
        id = $5;
$$;
select * from Users;
call UpdateUser('Andrew', 'Tehanov', 'andrewtehanov@gmail.com', '+375447756860', 2);
```

```
CREATE or REPLACE PROCEDURE UpdateVisit("a" timestamp without time zone,
 "b" character varying, c integer)
LANGUAGE SQL
AS $$
    UPDATE
        Visits
    SET
        date = "a",
        addition = "b"
    WHERE
        id = $3;
$$;
select * from Visits;
call UpdateVisit('2021-12-31T19:29', 'something!', 20);

CREATE or REPLACE PROCEDURE DeleteVisit ( integer )
LANGUAGE SQL
AS $$
    DELETE FROM
        service_visits
    WHERE
        id = $1;
    DELETE FROM
        Visits
    WHERE
        id = $1;
$$;
select * from Visits;
select * from service_visits;
call DeleteVisit(20);
```

Приложение Г

```
CREATE or REPLACE FUNCTION AllClients ()
    returns table (
        client_first_name varchar,
        client_last_name varchar,
        client_phone_number varchar,
        client_email varchar
    )
    LANGUAGE plpgsql
as $$
begin
    return query
        select
            first_name,
            last_name,
            phone_number,
            email
        from
            Users
        where
            type = 'Client';
end;$$
```

```
CREATE or REPLACE FUNCTION AllMasters ()
    returns table (
        master_first_name varchar,
        master_last_name varchar,
        master_phone_number varchar
    )
    LANGUAGE plpgsql
as $$
begin
    return query
        select
            first_name,
            last_name,
            phone_number
        from
            Users
        where
            type = 'Master';
end;$$
select * from AllMasters();
CREATE or REPLACE FUNCTION AllUsers ()
    returns table (
        user_first_name varchar,
        user_last_name varchar,
        user_phone_number varchar,
        user_email varchar
    )
    LANGUAGE plpgsql
as $$
begin
    return query
        select
            first_name,
            last_name,
            phone_number,
            email
        from
            Users;
end;$$
```

```
CREATE or REPLACE FUNCTION AllServices ()
    returns table (
        name_service varchar,
        service_price integer
    )
    LANGUAGE plpgsql
as $$
begin
    return query
        select
            service_name,
            price_cents
        from
            Services;
end;$$
CREATE or REPLACE FUNCTION AllVisits ()
    returns table (
        visit_id bigint,
        visit_client_id bigint,
        visit_master_id bigint,
        visit_state varchar,
        visit_date timestamp,
        visit_addition varchar
    )
    LANGUAGE plpgsql
as $$
begin
    return query
        select
            id,
            client_id,
            master_id,
            state,
            date,
            addition
        from
            Visits;
end;$$
```

```
CREATE or REPLACE FUNCTION VisitsById ( integer)
    returns table (
        visit_id bigint,
        visit_client_id bigint,
        visit_master_id bigint,
        visit_state varchar,
        visit_date timestamp,
        visit_addition varchar
    )
    LANGUAGE plpgsql
as $$
begin
    return query
        select
            id,
            client_id,
            master_id,
            state,
            date,
            addition
        from
            Visits
        where
            id=$1;
end;$$
```

```
CREATE or REPLACE FUNCTION VisitMasterFullName (integer)
    returns table (
        master_first_name varchar,
        master_last_name varchar
    )
    LANGUAGE plpgsql
as $$
begin
    return query
        select
            first_name,
            last_name
        from
            Users
        where
            type='Master'
        and
            id=$1;
end;$$
```