Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ
Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Базы данных

Тема «Столовая Лидо»
Лабораторная работа №1
Разработка серверной части прикладной программы

Студент: М.С. Патюпин
Преподаватель: С.С. Силич

МИНСК 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данная лабораторная работа выполняется на основе лабораторной работы №6 первого семестра по дисциплине «Базы данных», то есть весь реализованный ранее функционал необходимо сохранить. Данная работа включает:

– разработку спецификаций (требований к ПО) серверной части;

– программирование серверной части.

Разработка спецификаций (технических требований) серверной части (backend) программы. Включает минимально:

– составление уточненной реляционной схемы, определение имен и типов данных для всех полей БД;

– приведение имен в БД и имен в реляционной схеме в полное соответствие и создание подробного описания для каждого имени (имена таблиц и полей в реляционной схеме должны быть идентичны именам в SQL операторах);

– выделение двух ролей — пользователь и суперпользователь и определение их прав;

– выделение основной таблицы, с отображения которой по умолчанию стартует приложение;

– выделение справочников (LookUp Table) в отдельную категорию таблиц, право на запись в которые имеет только суперпользователь;

– оформление спецификации в соответствии с ГОСТ.

Программирование серверной части программы. Включает:

– создание скриптов для создания пустой базы данных;

– первичное наполнение таблиц-справочников;

– создание генераторов и скриптов для наполнения основной и прочих больших таблиц (объем в несколько сот записей);

– создание и отладка необходимых триггеров, функций и операторов;

– создание скриптов для сохранения/восстановления БД;

– тестирование сервера с помощью psql;

– оформление отчета о лабораторной работе;

– использование библиотеки libpq.

1 ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Уточненная реляционная схема изображается в соответствии с UML и должна иметь вид, изображенный на рисунке 1.1.

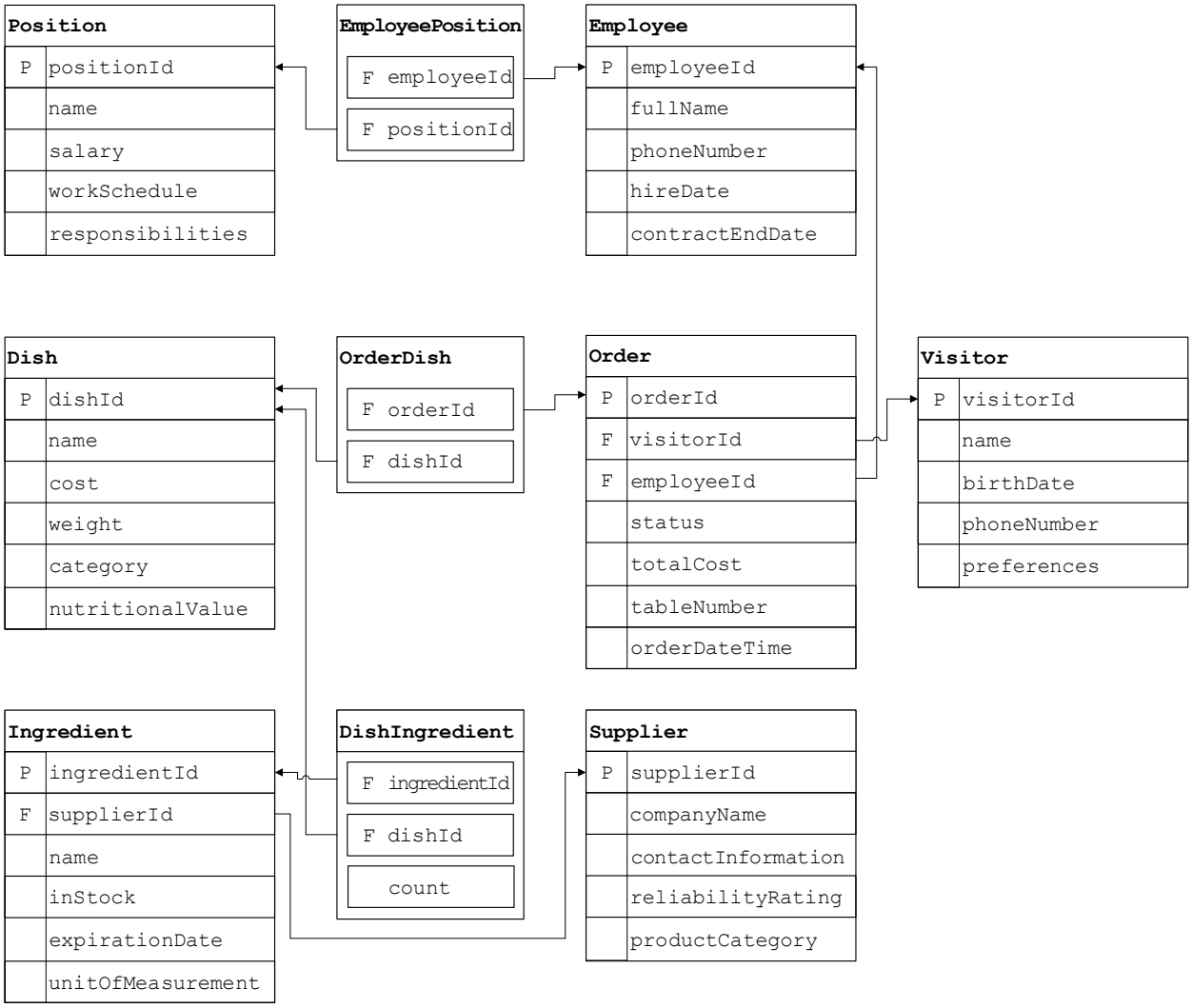

Рисунок 1.1 – Уточненная реляционная схема

Таблица `Order` представляет заказы в ресторане. Описание полей таблицы `Order`:

– `orderid`: идентификатор заказа. Первичный ключ;

– `status`: статус заказа (например, «в обработке», «выполнен»). Обязательное поле;

– `totalcost`: общая стоимость заказа. Обязательное поле;

– `numberofguests`: количество гостей;

– `orderdatetime`: дата и время заказа. Обязательное поле;

– `visitorid`: идентификатор посетителя, сделавшего заказ. Внешний ключ;

– `employeeid`: идентификатор сотрудника, принявшего заказ.

Внешний ключ, обязательное поле.

Таблица dish представляет блюда в меню ресторана. Описание полей таблицы dish:

– dishid: идентификатор блюда. Первичный ключ;

– name: название блюда. Обязательное поле;

– weight: вес блюда. Обязательное поле;

– nutritionalvalue: пищевая ценность блюда;

– cost: стоимость блюда;

– category: категория блюда (например, "закуски", "основные блюда").

Таблица dishingredient представляет связь между блюдами и ингредиентами. Описание полей таблицы dishingredient:

– dishid: идентификатор блюда. Первичный ключ, внешний ключ;

– ingredientid: идентификатор ингредиента. Первичный ключ, внешний ключ;

– quantity: количество ингредиента в блюде.

Таблица employee представляет сотрудников ресторана. Описание полей таблицы employee:

– employeeid: идентификатор сотрудника. Первичный ключ;

– fullname: полное имя сотрудника. Обязательное поле;

– phonenumber: номер телефона сотрудника;

– email: электронная почта сотрудника;

– hiredate: дата найма сотрудника;

– contractenddate: дата окончания контракта сотрудника.

Таблица employeeposition представляет связь между сотрудниками и их должностями. Описание полей таблицы employeeposition:

– employeeid: идентификатор сотрудника. Первичный ключ, внешний ключ;

– positionid: идентификатор должности. Первичный ключ, внешний ключ.

Таблица ingredient представляет ингредиенты, используемые в блюдах. Описание полей таблицы ingredient:

– ingredientid: идентификатор ингредиента. Первичный ключ;

– name: название ингредиента. Обязательное поле;

– unitofmeasurement: единица измерения ингредиента;

– instock: количество ингредиента на складе. Обязательное поле;

– expirationdate: срок годности ингредиента;

– supplierid: идентификатор поставщика ингредиента. Внешний ключ.

Таблица orderdish представляет связь между заказами и блюдами. Описание полей таблицы orderdish:

– orderid: идентификатор заказа. Первичный ключ, внешний ключ;

– dishid: идентификатор блюда. Первичный ключ, внешний ключ;

– count: количество порций блюда в заказе. Обязательное поле, по умолчанию 1.

Таблица position представляет должности сотрудников ресторана. Описание полей таблицы position:

– positionid: идентификатор должности. Первичный ключ;

– name: название должности. Обязательное поле;

– salary: зарплата по должности. Обязательное поле;

– workschedule: график работы;

– responsibilities: обязанности по должности.

Таблица supplier представляет поставщиков ингредиентов. Описание полей таблицы supplier:

– supplierid: идентификатор поставщика. Первичный ключ;

– companyname: название компании-поставщика. Обязательное поле;

– contactinformation: контактная информация поставщика;

– reliabilityrating: рейтинг надежности поставщика;

– productcategory: категория продукции поставщика.

Таблица visitor представляет посетителей ресторана. Описание полей таблицы visitor:

– visitorid: идентификатор посетителя. Первичный ключ;

– name: имя посетителя. Обязательное поле;

– birthdate: дата рождения посетителя (в формате строки);

– phonenumber: номер телефона посетителя;

– preferences: предпочтения посетителя.

Справочные таблицы: position, supplier, dish. Эти таблицы содержат предварительно определенные неизменяемые данные, используемые для поиска и сопоставления значений в других таблицах.

Основная таблица: Order. Эта таблица содержит основные бизнес-события (заказы), данные в ней динамичны и постоянно изменяются.

Обычный пользователь должен обладать правами просмотра, сохранения результатов запросов и редактирования всех таблиц, кроме справочных (position, supplier, dish).

Суперпользователь должен обладать всеми правами обычного пользователя, а также правом редактирования справочных таблиц и создания резервной копии базы данных. Для выполнения действий от имени суперпользователя приложение должно запрашивать пароль суперпользователя.

Серверная часть прикладной программы должна быть реализована в виде HTTP-сервера. Тела запросов и ответов сервера должны быть представлены в формате JSON.

Для взаимодействия с ресурсами (таблицами) должны использоваться

стандартные HTTP-методы:

1) GET – получение данных о ресурсе.

2) POST – создание нового ресурса.

3) PUT – обновление существующего ресурса.

4) DELETE – удаление ресурса.

Каждый ресурс должен быть доступен по уникальному URL:

– /api/orders: таблица `Order`;

– /api/dishes: таблица `dish`;

– /api/dishingredients: таблица `dishingredient`;

– /api/employees: таблица `employee`;

– /api/employeepositions: таблица `employeeposition`;

– /api/ingredients: таблица `ingredient`;

– /api/orderdishes: таблица `orderdish`;

– /api/positions: таблица `position`;

– /api/suppliers: таблица `supplier`;

– /api/visitors: таблица `visitor`.

Серверная часть прикладной программы должна предоставлять следующие операции для работы с базой данных:

– просмотр таблиц;

– фильтрация содержимого таблиц;

– добавление записей в таблицы;

– обновление записей в таблицах;

– удаление записей из таблиц;

– выполнение специальных запросов;

– создание бэкапов базы данных;

– сохранение результатов запросов в файл.

Исходный код базы данных приведен в приложении А.

Исходный код создания ролей и выделение таблиц справочников в базе данных приведен в приложении Б.

2 ПРОГРАММИРОВАНИЕ СЕРВЕРНОЙ ЧАСТИ

Работа с базой данных в серверной части осуществляется с использованием библиотеки libpq(psycopg2). Как было сказано ранее, данная лабораторная работа выполняется на основе уже имеющегося приложения для работы с базой данных. В данном случае, приложение было написано на языке программирования, который предоставляет инструменты для встраивания и вызова кода на C. Таким образом, уже существующее приложение можно адаптировать под отдельную серверную часть, в которой будет осуществляться взаимодействие с базой данных с помощью библиотеки psycopg2 через механизм другого языка.

Реализация серверной части в виде HTTP-сервера обеспечит возможность взаимодействия с ней, например из браузера, расширяя возможности реализации пользовательского интерфейса прикладной программы.

Так как функционал серверной части сохраняет функционал приложения из лабораторной работы №6, за исключением того, что для получения данных теперь используются HTTP-запросы, в качестве примера будет показан запрос на получение всех данных из таблицы `Order`. Для этого пользователю необходимо выполнить GET-запрос следующего формата: http://{адрес хоста}:{порт}/api/orders и войти под одним из зарегистрированных пользователей. Результат запроса в браузере изображен на рисунке 2.1, 2.2.

Скрипт для создания пустой базы данных приведен приложении А.

Скрипты для заполнения базы данных приведены в приложении В.

Скрипт создания функции вычисления стоимости заказа приведен в приложении Г.

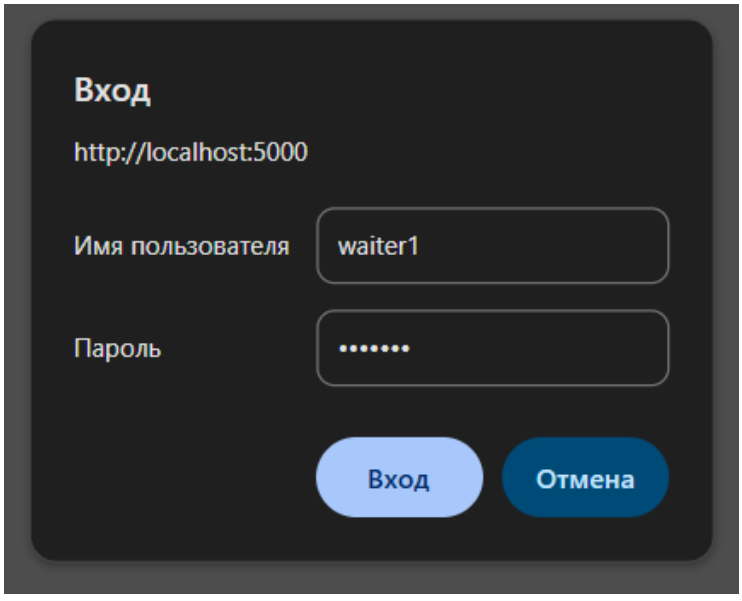

Рисунок 2.1 – Вход пользовател

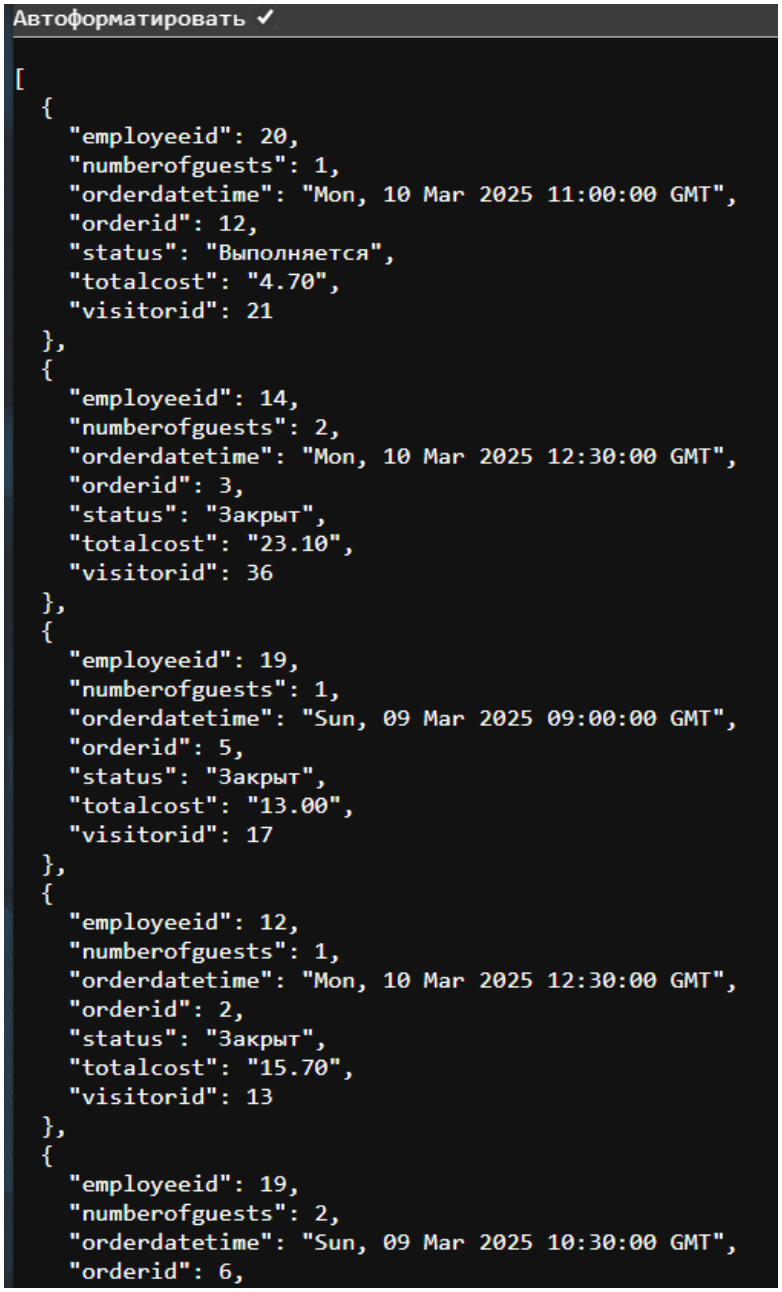

Рисунок 2.2 – Некоторые данные запрашиваемой таблицы

Исходный код программы сервера приведен в приложении Д.

ЗАКЛЮЧЕНИЕ

Разработанное программа представляет собой современное и эффективное решение для работы с реляционными базами данных, сочетающее в себе простоту использования и мощный функционал

Благодаря интуитивно понятному графическому интерфейсу, приложение позволяет значительно упростить и ускорить выполнение рутинных операций, таких как просмотр, редактирование, добавление и удаление данных, без необходимости написания сложных SQL-запросов вручную.

Важным преимуществом программа является его способность экономить время и снижать количество ошибок, связанных с ручным вводом данных. Автоматизация многих процессов и продуманный интерфейс минимизируют риски возникновения неточностей, что особенно критично при работе с большими объемами информации. Приложение не только упрощает взаимодействие с базами данных, но и повышает продуктивность пользователей, позволяя им сосредоточиться на анализе данных, а не на технических деталях их обработки.

Таким образом, данное приложение не только отвечает актуальным требованиям к работе с базами данных, но и задает новый стандарт удобства и эффективности. Оно является примером того, как современные технологии могут сделать сложные процессы доступными и понятными для пользователей с разным уровнем подготовки, способствуя повышению качества и скорости работы с информацией.

ПРИЛОЖЕНИЕ А
(обязательное)

Исходный код базы данных

```
CREATE TABLE IF NOT EXISTS public."Order"
(
    orderid serial NOT NULL,
    status character varying(50) COLLATE pg_catalog."default" NOT NULL,
    totalcost numeric(10, 2) NOT NULL,
    numberofguests integer,
    orderdatetime timestamp without time zone NOT NULL,
    visitorid integer,
    employeeid integer NOT NULL,
    CONSTRAINT "Order_pkey" PRIMARY KEY (orderid)
);

CREATE TABLE IF NOT EXISTS public.dish
(
    dishid serial NOT NULL,
    name character varying(100) COLLATE pg_catalog."default" NOT NULL,
    weight numeric(10, 2) NOT NULL,
    nutritionalvalue character varying(100) COLLATE pg_catalog."default",
    cost numeric(10, 2),
    category character varying(100) COLLATE pg_catalog."default",
    CONSTRAINT dish_pkey PRIMARY KEY (dishid)
);

CREATE TABLE IF NOT EXISTS public.dishingredient
(
    dishid integer NOT NULL,
    ingredientid integer NOT NULL,
    quantity numeric(10, 2),
    CONSTRAINT dishingredient_pkey PRIMARY KEY (dishid, ingredientid)
);

CREATE TABLE IF NOT EXISTS public.employee
(
    employeeid serial NOT NULL,
    fullname character varying(200) COLLATE pg_catalog."default" NOT NULL,
    phonenumber character varying(15) COLLATE pg_catalog."default",
    email character varying(100) COLLATE pg_catalog."default",
    hiredate date,
    contractenddate date,
    CONSTRAINT employee_pkey PRIMARY KEY (employeeid)
);

CREATE TABLE IF NOT EXISTS public.employeeposition
(
    employeeid integer NOT NULL,
    positionid integer NOT NULL,
    CONSTRAINT employeeposition_pkey PRIMARY KEY (employeeid, positionid)
);

CREATE TABLE IF NOT EXISTS public.ingredient
(
    ingredientid serial NOT NULL,
    name character varying(100) COLLATE pg_catalog."default" NOT NULL,
    unitofmeasurement character varying(50) COLLATE pg_catalog."default",
    instock numeric(10, 2) NOT NULL,
    expirationdate date,
```

```
    supplierid integer,
    CONSTRAINT ingredient_pkey PRIMARY KEY (ingredientid)
);

CREATE TABLE IF NOT EXISTS public.orderdish
(
    orderid integer NOT NULL,
    dishid integer NOT NULL,
    count integer NOT NULL DEFAULT 1,
    CONSTRAINT orderdish_pkey PRIMARY KEY (orderid, dishid)
);

CREATE TABLE IF NOT EXISTS public."position"
(
    positionid serial NOT NULL,
    name character varying(100) COLLATE pg_catalog."default" NOT NULL,
    salary numeric(10, 2) NOT NULL,
    workschedule character varying(100) COLLATE pg_catalog."default",
    responsibilities text COLLATE pg_catalog."default",
    CONSTRAINT position_pkey PRIMARY KEY (positionid)
);

CREATE TABLE IF NOT EXISTS public.supplier
(
    supplierid serial NOT NULL,
    companyname character varying(100) COLLATE pg_catalog."default" NOT NULL,
    contactinformation character varying(255) COLLATE pg_catalog."default",
    reliabilityrating integer,
    productcategory character varying(100) COLLATE pg_catalog."default",
    CONSTRAINT supplier_pkey PRIMARY KEY (supplierid)
);

CREATE TABLE IF NOT EXISTS public.visitor
(
    visitorid serial NOT NULL,
    name character varying(200) COLLATE pg_catalog."default" NOT NULL,
    birthdate character varying(5) COLLATE pg_catalog."default",
    phonenumber character varying(15) COLLATE pg_catalog."default",
    preferences text COLLATE pg_catalog."default",
    CONSTRAINT visitor_pkey PRIMARY KEY (visitorid)
);

ALTER TABLE IF EXISTS public."Order"
    ADD CONSTRAINT "Order_employeeid_fkey" FOREIGN KEY (employeeid)
    REFERENCES public.employee (employeeid) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;


ALTER TABLE IF EXISTS public."Order"
    ADD CONSTRAINT "Order_visitorid_fkey" FOREIGN KEY (visitorid)
    REFERENCES public.visitor (visitorid) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;


ALTER TABLE IF EXISTS public.dishingredient
    ADD CONSTRAINT dishingredient_dishid_fkey FOREIGN KEY (dishid)
    REFERENCES public.dish (dishid) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;
```

```
ALTER TABLE IF EXISTS public.dishingredient
    ADD CONSTRAINT dishingredient_ingredientid_fkey FOREIGN KEY
(ingredientid)
    REFERENCES public.ingredient (ingredientid) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;


ALTER TABLE IF EXISTS public.employeeposition
    ADD CONSTRAINT employeeposition_employeeid_fkey FOREIGN KEY (employeeid)
    REFERENCES public.employee (employeeid) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;


ALTER TABLE IF EXISTS public.employeeposition
    ADD CONSTRAINT employeeposition_positionid_fkey FOREIGN KEY (positionid)
    REFERENCES public."position" (positionid) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;


ALTER TABLE IF EXISTS public.ingredient
    ADD CONSTRAINT ingredient_supplierid_fkey FOREIGN KEY (supplierid)
    REFERENCES public.supplier (supplierid) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;


ALTER TABLE IF EXISTS public.orderdish
    ADD CONSTRAINT orderdish_dishid_fkey FOREIGN KEY (dishid)
    REFERENCES public.dish (dishid) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;


ALTER TABLE IF EXISTS public.orderdish
    ADD CONSTRAINT orderdish_orderid_fkey FOREIGN KEY (orderid)
    REFERENCES public."Order" (orderid) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;

END;
```

ПРИЛОЖЕНИЕ Б
(обязательное)

Исходный код определения пользователей

```
-- Создание роли для обычного пользователя (user)
CREATE ROLE "user" WITH
    LOGIN
    PASSWORD 'user'
    NOSUPERUSER
    NOCREATEDB
    NOCREATEROLE
    INHERIT
    NOREPLICATION;
-- Создание роли для администратора (admin)
CREATE ROLE admin WITH
    LOGIN
    PASSWORD 'admin'
    NOSUPERUSER
    NOCREATEDB
    NOCREATEROLE
    INHERIT
    NOREPLICATION;
-- Предоставление прав на просмотр (SELECT) для всех таблиц для роли user
GRANT SELECT ON TABLE public."Order", public.dish, public.dishingredient,
public.employee,
    public.employeeposition,       public.ingredient,       public.orderdish,
public.position,
    public.supplier, public.visitor TO "user";
-- Предоставление прав на создание, обновление и удаление (INSERT, UPDATE,
DELETE) для несправочных таблиц для user
GRANT INSERT, UPDATE, DELETE ON TABLE public."Order", public.dishingredient,
public.employee,
    public.employeeposition,       public.ingredient,       public.orderdish,
public.visitor TO "user";
-- Предоставление прав на использование последовательностей для несправочных
таблиц для user
GRANT    USAGE,    SELECT    ON    SEQUENCE    public."Order_orderid_seq",
public.employee_employeeid_seq,
    public.ingredient_ingredientid_seq,    public.visitor_visitorid_seq    TO
"user";
-- Предоставление всех прав для admin на все таблицы
GRANT    ALL    PRIVILEGES    ON    TABLE    public."Order",    public.dish,
public.dishingredient, public.employee,
    public.employeeposition,       public.ingredient,       public.orderdish,
public.position,
    public.supplier, public.visitor TO admin;
-- Предоставление прав на использование всех последовательностей для admin
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO admin;

-- Создание пользователей
CREATE USER waiter1 WITH PASSWORD 'waiter1';
CREATE USER manager1 WITH PASSWORD 'manager1';

-- Назначение ролей пользователям
GRANT "user" TO waiter1;
GRANT admin TO manager1;

-- Установка схемы по умолчанию для ролей
ALTER ROLE "user" SET search_path TO public;
ALTER ROLE admin SET search_path TO public;
```

ПРИЛОЖЕНИЕ В
(обязательное)

Исходный код для заполнения таблиц базы данных

Исходный код программы таблицы supplier

```
INSERT INTO public.supplier (
    companyname, contactinformation, reliabilityrating, productcategory
) VALUES
('ЛОГАЛ-БИО ООО', 'logal-bio.by, info@logal-bio.by', 8, 'Грибы'),
('АМИФРУТ ООО', 'www.amifruit.by', 7, 'Мясо'),
('ЕВРОФОЛИЯ ЧТПУП', '220018, город Минск, Шаранговича 7', 9, 'Мясо'),
('ИМПЕРИЯ ЗЛАКОВ ТМ СМОРГОНСКИЙ КОМБИНАТ ХЛЕБОПРОДУКТОВ УПП', 'Беларусь,
Гродненская область, Сморгонский район, Сморгонь, Комсомольский переулок, 20',
7, 'Пищевые добавки'),
('ВИГОЛ ООО', 'tastee.by', 8, 'Tastee'),
('ПИЩЕВОЙ КОМБИНАТ БЕЛКООПСОЮЗА ЧУП', '220075, город Минск, пр Партизанский
168', 6, 'Продукты питаня'),
('ДОБРЫЙ ДЕНЬ ООО', '220140, город Минск, Лещинского 8/4-1', 9, 'Продукты
питаня'),
('АВС ПЛЮС ФИРМА ООО ПРЕДСТАВИТЕЛЬСТВО', '212011, город Могилев, пер Березовский
1', 7, 'Продукты питаня'),
('КИТАЙСКИЙ ЧАЙ РЕКОЕ.BY МАГАЗИН ЧТУП АДС-ВОСТОК', '210015, город Витебск,
Генерала Белобородова 3-65а (ТДЦ Беларусь)', 8, 'Продукты питаня'),
('ОРЕХОВАЯ КОМПАНИЯ ООО', '212040, город Могилев, пер 1-й Южный 21 комн 1', 6,
'Продукты питаня'),
('БАКАЛЕЯ МОГИЛЕВ ОАО', '212040, город Могилев, Залуцкого 25', 9, 'Бакалея'),
('ИВАСИ-ТОРГ ЧАСТНОЕ ТОРГОВОЕ УНИТАРНОЕ ПРЕДПРИЯТИЕ', '211388, город Орша,
Ленина 230Ц', 7, 'Морепродукты'),
('МИРАНА ООО', '224005, город Брест, Комсомольская 23/1', 8, 'Консервы'),
('ВЕРХНИЙ ЛУГ СООО', '211622, Верхнедвинский район, Партизанская 1, д Янино',
6, 'Молочные продукты'),
('КРАСНЫЙ ПИЩЕВИК ОАО', 'priemnaya@zefir.by', 9, 'Продукты питаня'),
('ЛЕБУР-ПРОДУКТ-ПЛЮС ООО', '223050, Минский район, Базовская 1Б-2, пос
Сухорукие', 7, 'Мясо');
```

Исходный код программы таблицы position

```
INSERT INTO public."position" (
    name, salary, workschedule, responsibilities
) VALUES
('Менеджер', 2000.00, '2/2 по 12 часов', 'Управление проектами'),
('Разработчик', 2000.00, '2/5 по 6 часов', 'Разработка ПО'),
('Тестировщик', 1200.00, '2/5 по 6 часов', 'Тестирование ПО'),
('Аналитик', 2500.00, '2/5 по 6 часов', 'Анализ данных'),
('Шеф-повар', 2500.00, '2/2 по 12 часов', 'Руководство кухней'),
('Су-шеф', 2000.00, '2/2 по 12 часов', 'Помощь шеф-повару'),
('Повар', 1500.00, '2/2 по 12 часов', 'Приготовление блюд'),
('Кондитер', 1400.00, '4/3 по 4 часа', 'Приготовление десертов'),
('Пекарь', 1300.00, '4/3 по 4 часа', 'Выпечка хлебобулочных изделий'),
('Бариста', 1200.00, '2/2 по 12 часов', 'Приготовление кофе'),
('Официант', 1000.00, '2/2 по 12 часов', 'Обслуживание клиентов'),
('Хостес', 1100.00, '2/2 по 12 часов', 'Встреча гостей'),
('Администратор', 1500.00, '2/2 по 12 часов', 'Управление персоналом'),
('Кассир', 900.00, '2/2 по 12 часов', 'Работа на кассе'),
('Уборщик', 800.00, '2/2 по 12 часов', 'Уборка помещений'),
('Мойщик посуды', 700.00, '2/2 по 12 часов', 'Мытье посуды');
```

Исходный код программы таблицы employee

```
INSERT INTO public.employee (
    fullname, phonenumber, email, hiredate, contractenddate
) VALUES
('Бакунович Никита Андреевич', '+375291234501', 'bakunovich@lido.by', '2025-03-01', '2026-03-01'),
('Велич Никита Олегович', '+375291234502', 'velich@lido.by', '2025-03-02', '2026-03-02'),
('Георгиев Никита Димитров', '+375291234503', 'georgiev@lido.by', '2025-03-03', '2027-03-03'),
('Говор Павел Сергеевич', '+375291234504', 'govor@lido.by', '2025-03-04', '2025-09-04'),
('Горчаков Никита Сергеевич', '+375291234505', 'gorchakov@lido.by', '2025-03-05', '2026-03-05'),
('Гусаков Святослав', '+375291234506', 'gusakov@lido.by', '2025-03-01', '2027-03-01'),
('Демидович Руслан Сергеевич', '+375291234507', 'demidovich@lido.by', '2025-03-02', '2025-09-02'),
('Дылевский Егор Олегович', '+375291234508', 'dylevski@lido.by', '2025-03-03', '2026-03-03'),
('Жуковская Вероника Кирилловна', '+375291234509', 'zhukovskaya@lido.by', '2025-03-04', '2027-03-04'),
('Золотницкий Алексей Андреевич', '+375291234510', 'zolotnitski@lido.by', '2025-03-05', '2025-09-05'),
('Каражан Ксения Александровна', '+375291234511', 'karajan@lido.by', '2025-03-01', '2026-03-01'),
('Коледа Анна Валерьевна', '+375291234512', 'koleda@lido.by', '2025-03-02', '2027-03-02'),
('Корзун Ксения Игоревна', '+375291234513', 'korzun@lido.by', '2025-03-03', '2025-09-03'),
('Кузьмин Дмитрий Сергеевич', '+375291234514', 'kuzmin@lido.by', '2025-03-04', '2026-03-04'),
('Курдеко Никита Андреевич', '+375291234515', 'kurdeko@lido.by', '2025-03-05', '2027-03-05'),
('Курчицкий Глеб Анатольевич', '+375291234516', 'kurchitski@lido.by', '2025-03-01', '2025-09-01');
```

Исходный код программы таблицы dish

```
INSERT INTO public.dish (
    name, weight, nutritionalvalue, cost, category
) VALUES
('Горчица', 60.00, 'Белки –7,1 г; Жиры – 8,7 г; Углеводы – 12,0 г; Энергетическая ценность – 157,6 Ккал/659 кДж', 2.00, 'Соусы'),
('Блины с картофелем и грибами', 135.00, 'Белки –6,9 г; Жиры – 9,7 г; Углеводы – 26,3 г; Энергетическая ценность – 221 Ккал/925 кДж', 3.00, 'Блины'),
('Суп сырный с грибами', 280.00, 'Белки – 1,27г; Жиры – 2,3; Углеводы – 5,0 г; Энергетическая ценность – 49 Ккал/205 кДж', 2.40, 'Супы'),
('Суп куриный', 280.00, 'Белки – 3,6 г; Жиры – 2,2 г; Углеводы – 3,8 г; Энергетическая ценность – 50 Ккал/209 кДж', 2.80, 'Супы'),
('Солянка мясная', 280.00, 'Белки – 1,9 г; Жиры – 6,9 г; Углеводы – 2,7 г; Энергетическая ценность – 84 Ккал/351 кДж', 4.50, 'Супы'),
('Колбаса "Куриная с майораном" жареная', 100.00, 'Белки – 20,5 г; Жиры – 33,3 г; Углеводы – 0,9 г; Энергетическая ценность – 387 Ккал/1618 кДж', 5.30, 'Горячие блюда'),
('Скумбрия гриль', 100.00, 'Белки – 21,3 г; Жиры – 16,1 г; Углеводы – 0,9 г; Энергетическая ценность – 235 Ккал/983 кДж', 6.00, 'Гриль'),
('Шашлык из свинины "Фирменный"', 150.00, 'Белки – 19,4 г; Жиры – 45,5 г; Углеводы – 0,9 г; Энергетическая ценность – 496 Ккал/2075 кДж', 13.50, 'Гриль'),
```

```
('Шашлык из птицы "Оригинальный"', 160.00, 'Белки – 29,1 г; Жиры – 6,6 г; Углеводы – 2,1 г; Энергетическая ценность – 183 Ккал/766 кДж', 9.50, 'Гриль'),
('Окорочка "Ароматные"', 100.00, 'Белки – 26,0 г; Жиры – 21,0 г; Углеводы – 4,1 г; Энергетическая ценность – 305 Ккал/1275 кДж', 3.80, 'Горячие блюда'),
('Хлеб "Мамма Миа"', 60.00, 'Белки –11,0 г; Жиры – 4,0 г; Углеводы – 46 г; Энергетическая ценность – 260 Ккал/1120 кДж', 1.00, 'Выпечка'),
('Чиабатта', 100.00, 'Белки – 8 г; Жиры – 2 г; Углеводы – 52 г; Энергетическая ценность – 260 Ккал/1100 кДж', 1.00, 'Выпечка'),
('Хлеб', 35.00, 'Белки – 6,0 г; Жиры – 0,5 г; Углеводы – 48,0 г; Энергетическая ценность – 220 Ккал/950 кДж', 0.13, 'Выпечка'),
('Напиток из шиповника и боярышника', 250.00, 'Белки – 0,3 г; Жиры – 0 г; Углеводы – 8,6 г; Энергетическая ценность – 35 Ккал/146 кДж', 1.90, 'Напитки'),
('Топпинг клубничный', 60.00, 'Белки – 0,0 г; Жиры – 0,0 г; Углеводы – 69,0 г; Энергетическая ценность – 264 Ккал/1105 кДж', 2.50, 'Соусы'),
('Аджика "Домашняя"', 60.00, 'Белки – 1 г; Жиры – 4,6 г; Углеводы – 9,1 г', 0.90, 'Соусы');
```

Исходный код программы таблицы `visitor`

```
INSERT INTO public.visitor (
    name, birthdate, phonenumber, preferences
) VALUES
('Никульшин Борис Викторович', '01-01', '+375172932379', 'Предпочитает вегетарианские блюда, любит столик у окна'),
('Куприянова Диана Вячеславовна', '15-02', '+375172938617', 'Предпочитает блюда с морепродуктами, любит столик на террасе'),
('Перцев Дмитрий Юрьевич', '10-03', '+375172938039', 'Предпочитает блюда с низким содержанием жира, любит столик в углу'),
('Татур Михаил Михайлович', '25-04', '+375172938564', 'Предпочитает мясные блюда, любит столик у камина'),
('Старовойтов Валерий Васильевич', '30-05', NULL, 'Предпочитает морепродукты, любит столик на террасе'),
('Воронов Александр', '05-06', NULL, 'Предпочитает блюда с высоким содержанием белка, любит столик у окна'),
('Кобяк Игорь', '20-07', '+375172938569', 'Предпочитает блюда с высоким содержанием белка, любит столик в углу'),
('Луцик Юрий', '15-08', '+375172938697', 'Предпочитает блюда без сахара, любит столик у камина'),
('Одинец Дмитрий', '10-09', '+375172932389', 'Предпочитает блюда без сахара, любит столик у окна'),
('Селезнёв Игорь', '25-10', '+375172938569', 'Предпочитает блюда с низким содержанием углеводов, любит столик на террасе'),
('Фролов Игорь', '30-11', NULL, 'Предпочитает блюда с низким содержанием углеводов, любит столик в углу'),
('Байрак Сергей', '05-12', '+375172932389', 'Предпочитает веганские блюда, любит столик у камина'),
('Глецевич Иван', '10-01', '+375172938587', 'Предпочитает веганские блюда, любит столик у окна'),
('Ковальчук Анна', '20-02', '+375172938697', 'Предпочитает блюда с высоким содержанием клетчатки, любит столик на террасе'),
('Лукьянова Ирина', '15-03', '+375172938697', 'Предпочитает блюда с высоким содержанием клетчатки, любит столик в углу'),
('Поденок Леонид', '10-04', '+375172938039', 'Предпочитает блюда с низким содержанием жира, любит столик у камина');
```

Исходный код программы таблицы `employeeposition`

```
INSERT INTO public.employeeposition (
    employeeid, positionid
) VALUES
(1, 1),
(2, 1),
```

```
(3, 2),
(3, 3),
(3, 4),
(4, 5),
(5, 5),
(5, 17),
(6, 6),
(7, 6),
(8, 7),
(9, 7),
(10, 8),
(10, 9),
(11, 25),
(11, 26);
```

Исходный код программы таблицы ingredient

```
NSERT INTO public.ingredient (
    name, unitofmeasurement, instock, expirationdate, supplierid
) VALUES
('Мука', 'кг', 100.00, '2025-12-31', 8),
('Сахар', 'кг', 50.00, '2025-12-31', 9),
('Соль', 'кг', 20.00, '2025-12-31', 10),
('Молоко', 'л', 50.00, '2025-04-30', 41),
('Яйца', 'шт', 300.00, '2025-04-30', 14),
('Масло', 'кг', 50.00, '2025-12-31', 14),
('Курица', 'кг', 150.00, '2025-12-31', 6),
('Говядина', 'кг', 100.00, '2025-12-31', 2),
('Свинина', 'кг', 100.00, '2025-12-31', 7),
('Рыба', 'кг', 100.00, '2025-12-31', 12),
('Картофель', 'кг', 200.00, '2025-12-31', 34),
('Грибы', 'кг', 50.00, '2025-12-31', 1),
('Морковь', 'кг', 50.00, '2025-12-31', 32),
('Лук', 'кг', 50.00, '2025-03-31', 44),
('Оливки', 'кг', 30.00, '2025-12-31', 13),
('Лимон', 'кг', 20.00, '2025-06-30', 43);
```

Исходный код программы таблицы dishingredient

```
INSERT INTO public.dishingredient (
    dishid, ingredientid, quantity
) VALUES
(164, 19, 0.03),
(165, 1, 0.03),
(165, 2, 0.01),
(165, 3, 0.01),
(165, 4, 0.05),
(165, 5, 1.00),
(165, 11, 0.03),
(165, 12, 0.02),
(166, 12, 0.05),
(166, 13, 0.03),
(166, 18, 0.15),
(166, 21, 0.10),
(167, 7, 0.20),
(167, 13, 0.05),
(167, 14, 0.03),
(167, 21, 0.10);
```

Исходный код программы таблицы Order

```
INSERT INTO public."Order" (
    orderid, status, totalcost, numberofguests, orderdatetime, visitorid,
employeeid
) VALUES
(2, 'Закрыт', 15.70, 1, '2025-03-10 12:30:00', 13, 12),
(3, 'Закрыт', 23.10, 2, '2025-03-10 12:30:00', 36, 14),
(5, 'Закрыт', 13.00, 1, '2025-03-09 09:00:00', 17, 19),
(6, 'Закрыт', 34.40, 2, '2025-03-09 10:30:00', 22, 19),
(7, 'Закрыт', 25.05, 3, '2025-03-09 11:45:00', 29, 19),
(8, 'Закрыт', 77.60, 4, '2025-03-09 12:15:00', 25, 19),
(9, 'Закрыт', 55.91, 5, '2025-03-09 14:00:00', 31, 19),
(10, 'Закрыт', 29.00, 5, '2025-03-10 09:30:00', 42, 18),
(11, 'Закрыт', 13.00, 1, '2025-03-10 10:15:00', 15, 20),
(12, 'Закрыт', 4.70, 1, '2025-03-10 11:00:00', 21, 20),
(13, 'Закрыт', 19.40, 2, '2025-03-10 13:30:00', 19, 20),
(14, 'Закрыт', 33.75, 2, '2025-03-10 15:00:00', 37, 20),
(15, 'Закрыт', 25.05, 3, '2025-03-09 16:15:00', 32, 19),
(16, 'Закрыт', 39.95, 3, '2025-03-09 17:30:00', 33, 19),
(17, 'Закрыт', 27.70, 4, '2025-03-09 18:00:00', 38, 19),
(18, 'Закрыт', 49.50, 4, '2025-03-09 19:30:00', 12, 19);
```

Исходный код программы таблицы `orderdish`

```
INSERT INTO public.orderdish(
      orderid, dishid, count)
      VALUES

-- Заказ 2 (1 гость)
(2, 168, 1), -- Солянка
(2, 174, 1), -- Хлеб
(2, 183, 1),  -- Кефир (по  порции)

-- Заказ 3 (2 гостя)
(3, 177, 2), -- Напиток
(3, 169, 2), -- Колбаса (по 2 порции)
(3, 194, 1), -- Картофель отварной обжаренный (по 1 порции)
(3, 193, 1), -- Микс овощной (по 1 порции)

-- Заказ 11 (1 гость)
(5 , 180, 1), -- Десерт "Вишневый пломбир" (по  порции)
(5 , 181, 1), -- Фруктовый коктейль (по  порции)

-- Заказ 6 (2 гостя)
(6, 172, 2), -- Шашлык из птицы (по 3 порции)
(6, 177, 1), -- Напиток
(6, 204, 2), -- Пиво, в ассортименте (по  порции)
(6, 194, 2), -- Картофель отварной обжаренный (по 3 порции)

-- Заказ 3 (4 гостя)
(7, 177, 3), -- Напиток
(7, 202, 3), -- Котлета (по 4 порции)
(7, 194, 3), -- Картофель отварной обжаренный (по 4 порции)

-- Заказ 8 (4 гостя)
(8, 171, 4), -- Шашлык из свинины (по 4 порции)
(8, 177, 4), -- Напиток
(8, 200, 4), -- Картофель жареный соломкой (по 4 порции)
(8 , 174, 2), -- Хлеб "Мамма Миа" (по 2 порции)
(8 , 175, 2), -- Чиабатта (по 2 порции)

-- Заказ 9 (5 гостей)
(9, 177, 5), -- Напиток
```

```
(9, 169, 5), -- Колбаса (по 5 порций)
(9, 164, 2), -- Горчица (по 2 порции)
(9, 176, 2), -- Хлеб (по  порции)
(9, 175, 2), -- Чиабатта (по  порции)+
(9, 194, 5), -- Картофель отварной обжаренный (по 5 порций)
(9, 183, 2), -- Кефир (по  порции)

-- Заказ 10 (5 гостей)
(10, 185, 1), -- Блины с начинкой из вишни (по  порции)
(10, 186, 1), -- Блины с творогом (по  порции)
(10, 187, 1), -- Блины с мясом цыпленка (по  порции)
(10, 188, 1), -- Блины с сыром и ветчиной (по  порции)
(10, 189, 1), -- Блины с яблоками и клюквой (по  порции)
(10, 177, 3), -- Напиток
(10, 182, 5), -- Сметана (по  порции)
(10, 183, 1), -- Кефир (по  порции)+
(10, 184, 1), -- Йогурт питьевой (по  порции)


-- Заказ 11 (1 гость)
(11 , 180, 1), -- Десерт "Вишневый пломбир" (по  порции)
(11 , 181, 1), -- Фруктовый коктейль (по  порции)

-- Заказ 12 (1 гость)
(12, 167, 1), -- Суп куриный
(12, 177, 1), -- Напиток

-- Заказ 13 (2 гостя)
(13, 170, 2), -- Скумбрия (по 2 порции)
(13, 177, 2), -- Напиток
(13, 196, 2), -- Рис с куркумой (по 2 порции)

-- Заказ 14 (2 гостя)
(14, 172, 2), -- Шашлык из птицы (по 2 порции)
(14, 177, 2), -- Напиток
(14, 194, 2), -- Картофель отварной обжаренный (по 2 порции)
(14, 193, 1), -- Микс овощной (по  порции)

-- Заказ 15 (3 гостя)
(15, 202, 3), -- Котлета (по 3 порции)
(15, 177, 3), -- Напиток
(15, 194, 3), -- Картофель отварной обжаренный (по 3 порции)

-- Заказ 16 (3 гостя)
(16, 171, 1), -- Шашлык из свинины (по 1 порции)
(16, 177, 3), -- Напиток
(16, 200, 2), -- Картофель жареный соломкой (по 3 порции)
(16, 201, 1), -- Картофель отварной с укропом (по  порции)
(16, 202, 1), -- Котлета из свинины (по  порции)
(16, 203, 1), -- Говядина с грибами под сыром (по  порции)

-- Заказ 17 (4 гостя)
(17, 191, 4), -- Блины, 2 шт (по  порции)
(17, 178, 2), -- Топпинг клубничный (по  порции)
(17, 179, 2), -- Аджика "Домашняя" (по  порции)
(17, 177, 4), -- Напиток
(17, 194, 2), -- Картофель отварной обжаренный (по 4 порции)

-- Заказ 18 (4 гостя)
(18, 169, 1), -- Колбаса "Куриная с майораном" жареная (по  порции)
(18, 170, 1), -- Скумбрия гриль (по  порции)
(18, 171, 1), -- Шашлык из свинины "Фирменный" (по  порции)
(18, 172, 1), -- Шашлык из птицы "Оригинальный" (по  порции)
```

```
(18, 177, 4), -- Напиток
(18, 196, 2), -- Рис с куркумой (по 4 порции)
(18, 199, 2), -- Картофельное пюре (по  порции)

-- Заказ 19 (5 гостей)
(19, 198, 1), -- Драники по-домашнему (по  порции)
(19, 199, 1), -- Картофельное пюре (по  порции)+
(19, 200, 1), -- Картофель жареный соломкой (по  порции)+
(19, 201, 1), -- Картофель отварной с укропом (по  порции)+
(19, 202, 1), -- Котлета из свинины (по  порции)+
(19, 203, 1), -- Говядина с грибами под сыром (по  порции)+
(19, 168, 5), -- Солянка мясная (по  порции)
(19, 204, 5), -- Напиток


-- Заказ 20 (5 гостей)
(20, 180, 5), -- Десерт "Вишневый пломбир" (по  порции)
(20, 181, 5), -- Фруктовый коктейль (по  порции)

-- Заказ 21 (1 гость)
(21 , 181, 1), -- Фруктовый коктейль (по  порции)
(21, 178, 1), -- Топпинг клубничный

-- Заказ 22 (1 гость)
(22, 180, 1), -- Десерт "Вишневый пломбир"
(22, 177, 1), -- Напиток

-- Заказ 23 (2 гостя)
(23, 181, 2), -- Фруктовый коктейль (по 2 порции)

-- Заказ 24 (2 гостя)
(24, 186, 2), -- Блины с творогом (по 2 порции)
(24, 183, 2), -- Напиток

-- Заказ 25 (3 гостя)
(25, 185, 3), -- Блины с начинкой из вишни (по 3 порции)
(25, 182, 3), -- Сметана (по  порции)+
(25, 177, 3), -- Напиток
(25, 184, 1), -- Йогурт питьевой (по  порции)+

-- Заказ 26 (3 гостя)
(26, 190, 3), -- Шампиньоны "Ароматные" (по 3 порции)
(26, 192, 2), -- Стручковая фасоль с морковью и кукурузой (по  порции)\
(26, 195, 1), -- Картофель в кожуре обжаренный (по  порции)
(26, 177, 3), -- Напиток

-- Заказ 27 (4 гостя)
(27, 193, 4), -- Микс овощной (по 4 порции)
(27, 177, 4), -- Напиток

-- Заказ 28 (4 гостя)
(28, 192, 4), -- Стручковая фасоль с морковью и кукурузой (по 4 порции)
(28, 203, 4), -- Говядина с грибами под сыром (по  порции)+
(28, 177, 4); -- Напиток
```

ПРИЛОЖЕНИЕ Г
(обязательное)

Исходный код функции вычисления стоимости заказа

```
CREATE FUNCTION public.update_order_total() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
BEGIN
    UPDATE public."Order"
    SET totalcost = (
        SELECT COALESCE(SUM(d.cost * od.count), 0)  -- Используем COALESCE для
обработки NULL
        FROM public."orderdish" od
        JOIN public."dish" d ON od.dishid = d.dishid
        WHERE od.orderid = NEW.orderid
    )
    WHERE orderid = NEW.orderid;

    RETURN NEW;
END;
$$;
```

ПРИЛОЖЕНИЕ Д
(обязательное)

Исходный код программы сервера

```
from flask import Flask, jsonify, g
from flask_cors import CORS
from db import get_db_connection
import psycopg2
from auth import auth
import logging
from logging.handlers import RotatingFileHandler

from routes import (
    orders, dishes, dishingredients, employees, employeepositions,
    ingredients, orderdishes, positions, suppliers, visitors, queries, admin
)

# Настройка логирования
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
handler = RotatingFileHandler('app.log', maxBytes=10_000_000, backupCount=5)
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)
console_handler = logging.StreamHandler()
console_handler.setFormatter(formatter)
logger.addHandler(console_handler)

app = Flask(__name__)

app.url_map.strict_slashes = False

CORS(app, resources={
    r"/*": {
        "origins": "http://localhost:5173",
        "methods": ["GET", "POST", "PUT", "DELETE", "OPTIONS"],
        "allow_headers": ["Content-Type", "Authorization"],
        "expose_headers": ["Content-Disposition"],
        "supports_credentials": False
    }
})

@app.route('/', methods=['GET'])
@auth.login_required
def home():
    user = auth.username()
    role = auth.current_user()
    logger.info(f"User {user} (role: {role}) accessed endpoint: GET /")
    try:
        conn = get_db_connection(user, g.password)
        cur = conn.cursor()
        cur.execute('SELECT * FROM public."Order";')
        data = cur.fetchall()
        cur.close()
        conn.close()
        logger.info(f"User {user} (role: {role}) successfully retrieved
{len(data)} orders")
        return jsonify(data)
    except psycopg2.Error as e:
```

```
        logger.error(f"User {user} (role: {role}) failed to retrieve orders:
{str(e)}")
        return jsonify({'error': str(e)}), 400

# Регистрация Blueprint'ов
app.register_blueprint(orders, url_prefix='/api/orders')
app.register_blueprint(dishes, url_prefix='/api/dishes')
app.register_blueprint(dishingredients, url_prefix='/api/dishingredients')
app.register_blueprint(employees, url_prefix='/api/employees')
app.register_blueprint(employeepositions, url_prefix='/api/employeepositions')
app.register_blueprint(ingredients, url_prefix='/api/ingredients')
app.register_blueprint(orderdishes, url_prefix='/api/orderdishes')
app.register_blueprint(positions, url_prefix='/api/positions')
app.register_blueprint(suppliers, url_prefix='/api/suppliers')
app.register_blueprint(visitors, url_prefix='/api/visitors')
app.register_blueprint(queries, url_prefix='/api/queries')
app.register_blueprint(admin, url_prefix='/api/admin')

if __name__ == '__main__':
    logger.info("Starting Flask application")
    app.run(debug=True, host='0.0.0.0', port=5000)

from flask_httpauth import HTTPBasicAuth
from db import get_db_connection
import psycopg2
import logging
from flask import g

auth = HTTPBasicAuth()
logger = logging.getLogger(__name__)

@auth.verify_password
def verify_password(username, password):
    logger.info(f"Authentication attempt for user: {username}")
    try:
        # Проверяем, можно ли подключиться к базе данных с указанными учетными
данными
        conn = get_db_connection(username, password)
        cur = conn.cursor()
        # Получаем текущую роль пользователя
        cur.execute('SELECT current_user;')
        role = cur.fetchone()['current_user']
        cur.close()
        conn.close()
        # Проверяем, является ли роль допустимой (user, admin или пользователи,
унаследовавшие эти роли)
        if role in ['user', 'admin', 'waiter1', 'manager1']:
            # Сохраняем пароль в контексте запроса
            g.password = password
            logger.info(f"User {username} authenticated successfully with role:
{role}")
            return role
        logger.warning(f"User {username} provided invalid role: {role}")
        return None
    except psycopg2.Error as e:
        logger.error(f"Authentication failed for user {username}: {str(e)}")
        return None

@auth.get_user_roles
def get_user_roles(user):
    # Если пользователь waiter1 или другой с ролью user, возвращаем 'user'
    # Если пользователь manager1 или другой с ролью admin, возвращаем 'admin'
    if user in ['admin', 'manager1']:
```

```
        return 'admin'
    return 'user'

@auth.get_password
def get_password(username):
    # Пароль будет проверяться через psycopg2.connect, поэтому возвращаем None
    return None


import psycopg2
from psycopg2.extras import RealDictCursor

def get_db_connection(user, password):
    return psycopg2.connect(
        dbname="Lido",
        user=user,
        password=password,
        host="localhost",
        port="5432",
        cursor_factory=RealDictCursor
    )



from flask import Blueprint, jsonify, request, send_file, g
from db import get_db_connection
from auth import auth
import psycopg2
import json
import os
import subprocess
from datetime import datetime
import logging

# Настройка логирования
logger = logging.getLogger(__name__)

# Создание Blueprint'ов для каждой таблицы
orders = Blueprint('orders', __name__)
dishes = Blueprint('dishes', __name__)
dishingredients = Blueprint('dishingredients', __name__)
employees = Blueprint('employees', __name__)
employeepositions = Blueprint('employeepositions', __name__)
ingredients = Blueprint('ingredients', __name__)
orderdishes = Blueprint('orderdishes', __name__)
positions = Blueprint('positions', __name__)
suppliers = Blueprint('suppliers', __name__)
visitors = Blueprint('visitors', __name__)
queries = Blueprint('queries', __name__)
admin = Blueprint('admin', __name__)

# Список справочных таблиц, доступных только для admin
REFERENCE_TABLES = ['position', 'supplier', 'dish']

# Вспомогательная функция для проверки роли
def is_admin():
    return auth.current_user() in ['admin', 'manager1']

# Универсальная функция для фильтрации
def apply_filters(table_name, params):
    query = f'SELECT * FROM public."{table_name}" WHERE 1=1'
    values = []
    for key, value in params.items():
```

```
        if key in ['status', 'name', 'category', 'companyname',
'productcategory', 'phonenumber', 'email', 'workschedule', 'responsibilities',
'preferences', 'unitofmeasurement']:
            query += f' AND {key} ILIKE %s'
            values.append(f'%{value}%')
        elif key in ['totalcost', 'weight', 'cost', 'quantity', 'instock',
'salary', 'count', 'numberofguests', 'reliabilityrating']:
            query += f' AND {key} = %s'
            values.append(value)
        elif key in ['orderdatetime', 'hiredate', 'contractenddate',
'expirationdate', 'birthdate']:
            query += f' AND {key}::text ILIKE %s'
            values.append(f'%{value}%')
    return query, values

# Универсальная функция для обработки CRUD
def create_crud_routes(bp, table_name, primary_key, fields,
is_reference=False):
    @bp.route('/', methods=['GET'])
    @auth.login_required
    def get_all():
        user = auth.username()
        role = auth.current_user()
        logger.info(f"User {user} (role: {role}) accessed endpoint: GET
/{table_name}")
        try:
            conn = get_db_connection(auth.username(), g.password)
            cur = conn.cursor()
            query, values = apply_filters(table_name, request.args)
            cur.execute(query + ';', values)
            data = cur.fetchall()
            cur.close()
            conn.close()
            logger.info(f"User {user} (role: {role}) retrieved {len(data)}
records from {table_name}")
            return jsonify(data)
        except psycopg2.Error as e:
            logger.error(f"User {user} (role: {role}) failed to retrieve
records from {table_name}: {str(e)}")
            return jsonify({'error': str(e)}), 400

    @bp.route('/<id>', methods=['GET'])
    @auth.login_required
    def get_by_id(id):
        user = auth.username()
        role = auth.current_user()
        logger.info(f"User {user} (role: {role}) accessed endpoint: GET
/{table_name}/{id}")
        try:
            conn = get_db_connection(auth.username(), g.password)
            cur = conn.cursor()
            cur.execute(f'SELECT * FROM public."{table_name}" WHERE
{primary_key} = %s;', (id,))
            data = cur.fetchone()
            cur.close()
            conn.close()
            if data:
                logger.info(f"User {user} (role: {role}) retrieved record {id}
from {table_name}")
                return jsonify(data)
            logger.warning(f"User {user} (role: {role}) failed to find record
{id} in {table_name}")
            return jsonify({'error': 'Not found'}), 404
```

```
        except psycopg2.Error as e:
            logger.error(f"User {user} (role: {role}) failed to retrieve record
{id} from {table_name}: {str(e)}")
            return jsonify({'error': str(e)}), 400

    @bp.route('/', methods=['POST'])
    @auth.login_required
    def create():
        user = auth.username()
        role = auth.current_user()
        logger.info(f"User {user} (role: {role}) accessed endpoint: POST
/{table_name}")
        if is_reference and not is_admin():
            logger.warning(f"User {user} (role: {role}) attempted to modify
reference table {table_name}")
            return jsonify({'error': 'Only admin can modify reference
tables'}), 403
        try:
            data = request.get_json()
            conn = get_db_connection(auth.username(), g.password)
            cur = conn.cursor()
            columns = ', '.join(data.keys())
            placeholders = ', '.join(['%s'] * len(data))
            query = f'INSERT INTO public."{table_name}" ({columns}) VALUES
({placeholders}) RETURNING *;'
            cur.execute(query, list(data.values()))
            conn.commit()
            result = cur.fetchone()
            cur.close()
            conn.close()
            logger.info(f"User {user} (role: {role}) created record in
{table_name}: {result}")
            return jsonify(result), 201
        except psycopg2.Error as e:
            conn.rollback()
            logger.error(f"User {user} (role: {role}) failed to create record
in {table_name}: {str(e)}")
            return jsonify({'error': str(e)}), 400

    @bp.route('/<id>', methods=['PUT'])
    @auth.login_required
    def update(id):
        user = auth.username()
        role = auth.current_user()
        logger.info(f"User {user} (role: {role}) accessed endpoint: PUT
/{table_name}/{id}")
        if is_reference and not is_admin():
            logger.warning(f"User {user} (role: {role}) attempted to modify
reference table {table_name}")
            return jsonify({'error': 'Only admin can modify reference
tables'}), 403
        try:
            data = request.get_json()
            conn = get_db_connection(auth.username(), g.password)
            cur = conn.cursor()
            updates = ', '.join([f'{k} = %s' for k in data.keys()])
            query = f'UPDATE public."{table_name}" SET {updates} WHERE
{primary_key} = %s RETURNING *;'
            cur.execute(query, list(data.values()) + [id])
            conn.commit()
            result = cur.fetchone()
            cur.close()
            conn.close()
```

```
            if result:
                logger.info(f"User {user} (role: {role}) updated record {id} in
{table_name}: {result}")
                return jsonify(result)
            logger.warning(f"User {user} (role: {role}) failed to find record
{id} in {table_name}")
            return jsonify({'error': 'Not found'}), 404
        except psycopg2.Error as e:
            conn.rollback()
            logger.error(f"User {user} (role: {role}) failed to update record
{id} in {table_name}: {str(e)}")
            return jsonify({'error': str(e)}), 400

    @bp.route('/<id>', methods=['DELETE'])
    @auth.login_required
    def delete(id):
        user = auth.username()
        role = auth.current_user()
        logger.info(f"User {user} (role: {role}) accessed endpoint: DELETE
/{table_name}/{id}")
        if is_reference and not is_admin():
            logger.warning(f"User {user} (role: {role}) attempted to modify
reference table {table_name}")
            return jsonify({'error': 'Only admin can modify reference
tables'}), 403
        try:
            conn = get_db_connection(auth.username(), g.password)
            cur = conn.cursor()
            cur.execute(f'DELETE FROM public."{table_name}" WHERE
{primary_key} = %s RETURNING *;', (id,))
            conn.commit()
            result = cur.fetchone()
            cur.close()
            conn.close()
            if result:
                logger.info(f"User {user} (role: {role}) deleted record {id}
from {table_name}")
                return jsonify(result)
            logger.warning(f"User {user} (role: {role}) failed to find record
{id} in {table_name}")
            return jsonify({'error': 'Not found'}), 404
        except psycopg2.Error as e:
            conn.rollback()
            logger.error(f"User {user} (role: {role}) failed to delete record
{id} from {table_name}: {str(e)}")
            return jsonify({'error': str(e)}), 400

# Определение маршрутов для каждой таблицы
create_crud_routes(orders, 'Order', 'orderid', [
    'status', 'totalcost', 'numberofguests', 'orderdatetime', 'visitorid',
'employeeid'])
create_crud_routes(dishes, 'dish', 'dishid', [
    'name', 'weight', 'nutritionalvalue', 'cost', 'category'],
is_reference=True)
create_crud_routes(dishingredients, 'dishingredient', 'dishid,ingredientid', [
    'dishid', 'ingredientid', 'quantity'])
create_crud_routes(employees, 'employee', 'employeeid', [
    'fullname', 'phonenumber', 'email', 'hiredate', 'contractenddate'])
create_crud_routes(employeepositions, 'employeeposition',
'employeeid,positionid', [
    'employeeid', 'positionid'])
create_crud_routes(ingredients, 'ingredient', 'ingredientid', [
    'name', 'unitofmeasurement', 'instock', 'expirationdate', 'supplierid'])
```

```
create_crud_routes(orderdishes, 'orderdish', 'orderid,dishid', [
    'orderid', 'dishid', 'count'])
create_crud_routes(positions, 'position', 'positionid', [
    'name', 'salary', 'workschedule', 'responsibilities'], is_reference=True)
create_crud_routes(suppliers, 'supplier', 'supplierid', [
    'companyname',          'contactinformation',          'reliabilityrating',
'productcategory'], is_reference=True)
create_crud_routes(visitors, 'visitor', 'visitorid', [
    'name', 'birthdate', 'phonenumber', 'preferences'])

# Маршрут для специальных запросов и сохранения результатов
@queries.route('/execute', methods=['POST'])
@auth.login_required
def execute_query():
    user = auth.username()
    role = auth.current_user()
    logger.info(f"User   {user}   (role:   {role})   accessed   endpoint:   POST
/queries/execute")
    try:
        data = request.get_json()
        query = data.get('query')
        save_to_file = data.get('save_to_file', False)
        if not query:
            logger.warning(f"User {user} (role: {role}) sent empty query")
            return jsonify({'error': 'Query is required'}), 400
        conn = get_db_connection(auth.username(), g.password)
        cur = conn.cursor()
        cur.execute(query)
        if query.strip().upper().startswith('SELECT'):
            results = cur.fetchall()
            logger.info(f"User  {user}  (role:  {role})  executed  SELECT  query,
retrieved {len(results)} records")
        else:
            conn.commit()
            results = {'message': 'Query executed successfully'}
            logger.info(f"User   {user}   (role:   {role})   executed   non-SELECT
query")
        cur.close()
        conn.close()
        if save_to_file:
            filename                                                              =
f'query_result_{datetime.now().strftime("%Y%m%d_%H%M%S")}.json'
            with open(filename, 'w') as f:
                json.dump(results, f, indent=2, default=str)
            logger.info(f"User  {user}  (role:  {role})  saved  query  results  to
{filename}")
            return send_file(filename, as_attachment=True)
        return jsonify(results)
    except psycopg2.Error as e:
        logger.error(f"User  {user}  (role:  {role})  failed  to  execute  query:
{str(e)}")
        return jsonify({'error': str(e)}), 400

# Маршрут для создания бэкапа (только для admin)
@admin.route('/backup', methods=['POST'])
@auth.login_required
def create_backup():
    user = auth.username()
    role = auth.current_user()
    logger.info(f"User   {user}   (role:   {role})   accessed   endpoint:   POST
/admin/backup")
    if not is_admin():
```

```
        logger.warning(f"User {user} (role: {role}) attempted to create backup
without admin privileges")
        return jsonify({'error': 'Only admin can create backups'}), 403
    try:
        data = request.get_json()
        admin_password = data.get('admin_password')
        if not admin_password:
            logger.warning(f"User {user} (role: {role}) did not provide admin
password")
            return jsonify({'error': 'Admin password is required'}), 400
        # Проверяем пароль администратора
        try:
            conn = get_db_connection('manager1', admin_password)
            conn.close()
        except psycopg2.Error:
            logger.error(f"User {user} (role: {role}) provided invalid admin
password")
            return jsonify({'error': 'Invalid admin password'}), 401
        backup_file = f'backup_{datetime.now().strftime("%Y%m%d_%H%M%S")}.sql'
        subprocess.run([
            'pg_dump',
            '-U', 'manager1',
            '-h', 'localhost',
            '-p', '5432',
            '-F', 'p',
            '-f', backup_file,
            'restaurant_db'
        ], env={'PGPASSWORD': admin_password}, check=True)
        logger.info(f"User    {user}    (role:    {role})    created    backup:
{backup_file}")
        return send_file(backup_file, as_attachment=True)
    except subprocess.CalledProcessError:
        logger.error(f"User {user} (role: {role}) failed to create backup")
        return jsonify({'error': 'Failed to create backup'}), 500
    except Exception as e:
        logger.error(f"User  {user}  (role:  {role})  encountered  error  during
backup: {str(e)}")
        return jsonify({'error': str(e)}), 400
```