МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
Национальный исследовательский ядерный университет «МИФИ»

Институт
интеллектуальных кибернетических систем

Кафедра кибернетики (№22)

Направление подготовки 09.03.04 Программная инженерия

Курсовая работа по предмету

«Основы автоматизированных информационных технологий»

Тема: «Grow Food — сервис доставки рационов готовой полезной еды»

Преподаватель: Тихомирова Дарья Валерьевна

Студент: Баранов Александр Тимурович

Группа: Б22-534

Москва, 2025

Содержание

1 Описание предметной области

1.1 Формулировка задания

Спроектировать базу данных для сервиса доставки здорового питания «Grow Food», поддерживающую полный жизненный цикл заказа: от его создания и обработки до доставки клиенту и взаимодействия с поставщиками продуктов и курьерами.

1.2 Конкретизация предметной области

Сервис доставки здорового питания «Grow Food» – это информационная система, предназначенная для автоматизации процессов работы сервиса. Система рассчитана на взаимодействие с сотрудниками сервиса, при этом пользователи (клиенты) не имеют прямого доступа к базе данных.

В рамках системы предполагается реализация следующего функционала:

- Работа с поставщиками: Учёт поставок продуктов и сроков доставки ингредиентов.
- Составление меню: Формирование меню на основе предпочтений пользователей и доступности ингредиентов. Меню составляется на следующий месяц и может быть адаптировано под индивидуальные запросы клиентов.
- Приготовление блюд: Учёт рецептов и формирование готовых рационов. Сотрудники кухни имеют доступ к информации об ингредиентах, необходимых для приготовления блюд, а также о самих блюдах: их количестве, размере порции, калорийности и других характеристиках.
- Доставка: Курьеры имеют доступ к информации о доставленных или доставляемых заказах: дата, адрес доставки, способ оплаты и сумма к оплате (при необходимости).
- Оплата: Поддержка различных способов оплаты заказов, включая онлайн-платежи, а также учёт финансовых операций.
- Учёт предпочтений пользователей: Хранение информации о диетических ограничениях, аллергиях и предпочтениях клиентов для персонализации меню. Каждому пользователю в его личном кабинете доступна информация о выбранных предпочтениях, расписании меню на следующий месяц, количестве накопленных баллов, а также предложении оформить заказ.

Система обеспечивает взаимодействие между всеми участниками процесса: сотрудни-

ками кухни, курьерами, поставщиками и администраторами сервиса, при этом пользователи взаимодействуют с системой через личный кабинет без прямого доступа к базе данных.

1.3 Пользователи системы

1. Клиенты:

- Это физические лица, которые заказывают готовые рационы питания через сервис «Grow Food».
- Пользователи не имеют прямого доступа к базе данных, но взаимодействуют с системой через личный кабинет.
- В личном кабинете клиентам доступна следующая информация:
  - Выбранные предпочтения (диетические ограничения, аллергии, предпочтения по ингредиентам).
  - Расписание меню на следующий месяц.
  - Количество накопленных баллов (если предусмотрена бонусная система).
  - Возможность оформления новых заказов и просмотра истории заказов.

2. Курьеры:

- Это сотрудники сервиса, отвечающие за доставку готовых рационов клиентам.
- Курьеры имеют доступ к информации о доставленных или доставляемых заказах, включая:
  - Дату и время доставки.
  - Адрес доставки.
  - Способ оплаты и сумму к оплате (если требуется).
- Их задача – обеспечить своевременную и корректную доставку заказов, а также взаимодействие с клиентами при необходимости.

3. Поставщики:

- Это юридические лица или индивидуальные предприниматели, поставляющие ингредиенты для приготовления блюд.
- Поставщики взаимодействуют с системой через сотрудников сервиса, предоставляя информацию о поставках:

- Наименование и количество поставляемых продуктов.
- Сроки и условия доставки.
- Качество и соответствие стандартам.
- Их роль – обеспечить своевременные и качественные поставки ингредиентов для приготовления блюд.

4. Администраторы:

- Это сотрудники сервиса, отвечающие за управление системой и поддержание её работоспособности.
- Администраторы имеют полный доступ к базе данных и выполняют следующие функции:
  - Управление пользователями (клиентами, курьерами, поставщиками).
  - Контроль за составлением меню и процессом приготовления блюд.
  - Мониторинг доставки заказов и взаимодействие с курьерами.
  - Настройка и поддержка системы, включая обработку ошибок и обновление данных.

1.4 Сроки хранения информации

В системе «Grow Food» предусмотрены различные сроки хранения информации в зависимости от её типа и назначения:

1. Информация о заказах:

- Данные о завершённых заказах (история заказов) хранятся в течение 3 лет для обеспечения возможности анализа и отчётности.
- По истечении этого срока заказы перемещаются в архив, где хранятся ещё 2 года перед окончательным удалением.

2. Информация о пользователях:

- Данные о клиентах (личные данные, предпочтения, история заказов) хранятся до момента удаления аккаунта пользователем.
- В случае неактивности аккаунта в течение 2 лет, данные перемещаются в архив и хранятся там ещё 1 год перед удалением.

3. Информация о поставщиках и ингредиентах:

- Данные о поставщиках и поставках хранятся в течение 5 лет для обеспечения контроля качества и анализа сотрудничества.
- Информация о списанных или более неиспользуемых ингредиентах перемещается в архив через 1 год после их последнего использования.

4. Информация о курьерах и доставках:

- Данные о курьерах и их доставках хранятся в течение 3 лет для анализа эффективности работы.
- По истечении этого срока информация перемещается в архив, где хранится ещё 1 год.

5. Архивные данные:

- Архивные данные хранятся в отдельной структуре базы данных для снижения нагрузки на основную систему.
- Доступ к архивным данным ограничен и предоставляется только администраторам для целей отчётности или анализа.

1.5 События, изменяющие состояние базы данных

Основные события, которые влияют на состояние базы данных системы «Grow Food», включают:

- Регистрация новых пользователей или удаление (деактивация) аккаунтов старых пользователей.
- Оформление нового заказа (изменение его статуса).
- Устройство нового курьера на работу.
- Добавление нового поставщика или изменение договора со старым поставщиком.
- Поступление ингредиентов от поставщиков.
- Изменение меню.
- Начисление баллов пользователям.
- Изменение в статусе оплаты заказа.

Эти события обеспечивают актуальность данных в системе и позволяют отслеживать все ключевые процессы, связанные с работой сервиса.

1.6 Основные запросы к базе данных (на естественном языке)

Основные запросы к БД в разрезе аналитики сервиса:

- Вычисление ежедневной выручки и её прироста.
- Вычисление НДС и цен без учёта налога.
- Анализ возвращённых или отменённых заказов.
- Анализ распределения пользователей по количеству их заказов, количеству бонусов.
- Нахождение блюд с популярными (по предпочтению пользователей) ингредиентами.
- Вычисление необходимого количества ингредиентов и прогнозирование поставок.
- Определение прибыльности и актуальности меню на основе заказов.

2 Концептуально-информационная модель предметной области

2.1 ER-диаграмма модели

Для создания диаграммы использовался сервис drawio. Получившийся рисунок приведен в приложении.

2.2 Оценка мощностных характеристик сущностей и связей

Сущность/связь	Мощность		
	минимальная	**средняя**	**максимальная**
Пользователь	0	10 000	1 000 000
Предпочтение	0	10	20
Способ оплаты	0	5	10
Заказ	0	50 000	5 000 000
Курьер	0	100	1 000
Меню	0	50	500
Блюдо	0	200	2 000
Ингредиент	0	500	5 000
Поставщик	0	20	100
Пользователь имеет предпочтения	0	20 000	2 000 000
Пользователь пригласил пользователя	0	2 000	200 000
Пользователь выбирает способ оплаты	0	10 000	1 000 000
Пользователь имеет заказы	0	50 000	5 000 000
Курьер доставляет заказы	0	50 000	5 000 000
Заказ содержит меню	0	50 000	5 000 000

Все представленные мощностные характеристики основываются на следующих соображениях:

- Минимальное количество соответствует состоянию, когда база данных только заведена и зарегестрированных пользователей еще нет.
- Средние значения указаны из расчёта на «нормальную» работу системы.

- Максимальные значения указаны для случая работы системы в режиме большой нагрузки.
- Сложно оценить мощностные характеристики числа приглашенных пользователей, а также числа выбранных пользователем предпочтений, так как это число напрямую зависит от характера пользователей.

3 Концептуальное проектирование

3.1 Принятые проектные соглашения

- Имена таблиц указываются в snake_case во множественном числе: users, orders.
- Составные атрибуты (например, “Имя, фамилия, отчество”) разделяются на отдельные поля: first_name, last_name.
- Для внешних ключей используется шаблон {table_name}_id (например, user_id для связи с пользователем).
- Первичные ключи (PK) являются автоинкрементными целыми числами (SERIAL в PostgreSQL) для всех сущностей.
- Внешние ключи (FK) реализуются с указанием действий при удалеии (CASCADE, SET NULL).
- Таблицы нормализованы по третьей нормальной форме (атрибуты зависят от PK и нет транзитивных связей).

3.2 Обоснование выбора модели базы данных

В качестве модели базы данных была выбрана реляционная модель. Это решение обосновано несколькими факторами:

1. Структурированность данных. Информация, обрабатываемая системой (пользователи, заказы, курьеры, предпочтения, способы оплаты), имеет чёткую табличную структуру с фиксированными атрибутами. Это естественно укладывается в формат таблиц с чётко заданными связями между ними.
2. С реляционными БД можно работать с помощью такой активно поддерживаемой СУБД, как PostgreSQL. Это позволит эффективно нормализовать данные и поддерживать целостность благодаря механизмам внешних ключей, ограничений и транзакций. Эта СУБД предоставляет мощные инструменты для выполнения аналитических запросов (в том числе оконные функции, CTE, агрегации), что соответствует потребностям проекта в построении аналитических отчётов и статистики.

3.3 Используемые в системе кодификаторы

В проектируемой информационной системе используются следующие кодификаторы:

1. preference_categories — категории предпочтений пользователей

Данный кодификатор содержит категории пищевых предпочтений и ограничений, позволяющие учитывать индивидуальные особенности клиентов при формировании рационов:

- аллергены
- десерты, выпечка, сахар
- мясо, рыба
- овощи, лук, чеснок
- гарниры, каши

2. payment_methods — методы оплаты

Кодификатор содержит список возможных способов оплаты, а также признак необходимости реквизитов:

Метод оплаты	Требует реквизиты
Наличные курьеру	Нет
Карта курьеру	Да
Картой онлайн	Да
Яндекс.Сплит	Да

3. order_status — статусы заказов

Кодификатор содержит возможные состояния заказа в процессе его обработки:

- new — заказ создан
- in_delivery — передан курьеру
- delivered — доставлен
- cancelled — отменён
- returned — возвращён

3.4 Концептуальная модель базы данных

Для создания диаграммы использовался сервис drawio. Получившийся рисунок приведен в приложении.

4 Логическое проектирование

4.1 ER-диаграмма базы данных

Для создания диаграммы использовался сервис drawio. Получившийся рисунок приведен в приложении.

4.2 Схемы отношений базы данных

Схема была построена встроеными средствами инструмента с графическим интерфейсом для управления базами данных PostgreSQL PgAdmin. Получившийся рисунок приведен в приложении.

4.3 Схема реляционной базы данных

Отношение	Атрибуты
R1	(#id, first_name, middle_name, last_name, sex, birth_date, phone_number, invited_by, bonuses, address)
R2	(#id, preference, user)
R3	(#id, title, preference_category)
R4	(#id, preference, preference_category)
R5	(#id, title)
R6	(#id, title, require_requisites)
R7	(#id, user, payment_method, requisites)
R8	(#id, user, menu, created_at, status, courier, payment_info)
R9	(#id, first_name, middle_name, last_name, birth_date)
R10	(#id, title, cost, count_dishes, colorfulness)
R11	(#id, title, type, weight, colorfulness)
R12	(#id, dish, ingredient)
R13	(#id, title, type, cost)
R14	(#id, ingredient, supplier)
R15	(#id, title, productivity)
R16	(#id, ingredient, preference)

4.4 Схемы основных запросов на реляционной алгебре

1. Получить всех пользователей, у которых задано хотя бы одно предпочтение

```
Users(id) = (user[user.id = users_preferences.user]users_preferences)[
    user.id]
```

2. Получить все блюда, в которых используются ингредиенты дороже 100

```
Dishes(id) = ((dish[dish.id = dishes_ingredients.dish]dishes_ingredients
    )
            [dishes_ingredients.ingredient = ingredient.id & ingredient.
               cost > 100]ingredient)[dish.id]
```

3. Получить заказы, оплаченные через метод, требующий реквизиты

```
Orders(id) = ((order[order.payment_info = payment_info.id]payment_info)
                [payment_info.payment_method = payment_method.id &
                payment_metho require_requisites = true]payment_method)[
                   order.id]
```

4. Получить всех поставщиков, которые поставляют хотя бы один ингредиент типа ‘овощ’

```
Suppliers(id) = ((supplier[supplier.id = ingredients_suppliers.supplier]
    ingredients_suppliers)
                [ingredients_suppliers.ingredient = ingredient.id &
                   ingredient.type = 'овощ'] ingredient)
                [supplier.id]
```

5. Найти заказы, содержащие блюда с калорийностью выше 300

```
Orders(id) = (order[order.menu = menu.id & menu.colorfulness > 300]menu)
    [order.id]
```

6. Получить все категории предпочтений, в которых есть хотя бы одно предпочтение

```
Categories(id) = (preference_category
                  [preference_category.id = preferences_categories.
                     preference_category]
                  preferences_categories)[preference_category.id]
```

5 Физическое проектирование

5.1 Обоснование выбора конкретной СУБД

В качестве системы управления базами данных (СУБД) для реализации проекта была выбрана PostgreSQL. Этот выбор обоснован рядом факторов:

- PostgreSQL — это мощная объектно-реляционная СУБД с открытым исходным кодом, которая поддерживает расширенные возможности SQL и соответствует стандартам.
- Система обладает высокой производительностью и эффективной планировкой запросов, что важно при работе с большим объемом данных и сложными связями между таблицами.
- PostgreSQL широко используется в индустрии и хорошо поддерживается сообществом, что обеспечивает доступность документации, инструментов и решений типовых задач.
- Также немаловажным преимуществом является наличие встроенной поддержки пользовательских типов, таких как перечисления (ENUM), которые активно применяются в данной базе данных.
- Наконец, PostgreSQL хорошо интегрируется с Python через библиотеки psycopg и SQLAlchemy, что важно на этапе генерации и заполнения данных.

Таким образом, PostgreSQL является оптимальным выбором как с технической, так и с практической точек зрения.

5.2 Создание базы данных

Создание и запуск базы данных производится с использованием среды контейнеризации Docker. Это позволяет упростить настройку окружения, добиться воспроизводимости конфигурации и обеспечить удобство развертывания.

Для запуска PostgreSQL используется официальный образ postgres, настраиваемый через docker-compose. Все параметры подключения (имя пользователя, пароль, имя базы данных и порт) передаются через файл .env, что обеспечивает гибкость и безопасность конфигурации.

Для работы с базой и отслеживания её состояния применяется PGAdmin, также запускаемый как отдельный контейнер. Он позволяет удобно просматривать содержимое таблиц, выполнять SQL-запросы и отслеживать структуру базы данных через графический интерфейс.

Создание таблиц производится автоматически при старте контейнера с помощью специ-

ального SQL-файла schema.sql, который монтируется в контейнер базы данных и исполняется через docker-entrypoint-initdb.d. Эта схема позволяет без участия пользователя разворачивать структуру БД при первом запуске.

Таким образом, вся инфраструктура для базы данных развернута в виде контейнеров, а структура таблиц автоматически формируется при старте, что делает процесс развертывания гибким и надежным.

5.3 Создание таблиц

Создание таблиц производилось с помощью инициирующего SQL-запроса schema.sql, исполняющегося через docker-entrypoint-initdb.d. Код для создания всех таблиц вынесен в приложение.

5.4 Заполнение таблиц. ETL-процессы загрузки базы данных

Заполнение базы данных проводилось в несколько этапов с использованием различных источников и инструментов, в зависимости от характера данных.

1. Источники данных

Для так называемых статических сущностей — то есть данных, которые либо редко изменяются, либо представляют собой заранее известные справочники, были использованы как реальные источники, так и логически обоснованные предположения:

- Данные о предпочтениях пользователей, категориях предпочтений и стоимости меню были заимствованы из существующего сервиса по доставке здорового питания GrowFood.
- Информация о типах ингредиентов, блюдах и их составе формировалась на основе анализа ассортимента типовых рационов здорового питания, а также общих гастрономических принципов. Эти данные предварительно подготавливались в виде CSV-файлов и импортировались в базу.

2. Генерация и вставка данных

Остальные сущности, в том числе:

- Пользователи,
- Курьеры,

- Заказы,
- Платёжные методы,
- Поставщики и поставки,

генерировались программно с использованием библиотеки Faker (ru_RU), обеспечивающей реалистичные русскоязычные данные (имена, адреса, даты рождения и т.д.).

Для этого были реализованы Python-скрипты, которые подключаются к базе данных с помощью библиотеки psycopg2. Скрипты построены по модульному принципу: каждая таблица имеет собственный скрипт-загрузчик (seeder), а главный модуль запускает заполнение в нужной последовательности, учитывая зависимости между таблицами (например, сначала пользователи — затем заказы).

Код для Python-скриптов вынесен в приложение

После выполнения всех функций заполнения базы данных были получены заполненные таблицы, проиллюстрированные в приложении

5.5 Запросы в терминах SQL

1. Найти выручку сервиса за последний месяц. Код запроса, результат выполнения.
2. Какая доля (в %) общей выручки приходится на меню с названием «Похудение»? Проценты округлить до двух знаков после запятой. Код запроса, результат выполнения.
3. Вычислить НДС каждого меню и рассчитать цену каждого меню не включая НДС. Вывести название меню, его текущую цену, НДС и цену без НДС. Значения округлить до двух знаков после запятой. Код запроса, результат выполнения.
4. Найти заказы, которые оказались возвращены и были оплачены с помощью «Яндекс.Сплит». Вывести id заказа, дату заказа и реквизиты платежа. Код запроса, результат выполнения.
5. Найти топ-5 пользователей по количеству бонусов, которые большинство своих заказов оплатили наличными курьеру. Вывести id пользователя, его имя и количество бонусов. Код запроса, результат выполнения.
6. Найти блюда, содержащие хотя бы один ингредиент, который относится к самой популярной категории предпочтений. Вывести только названия блюд, отсортированные в алфавитном порядке. Код запроса, результат выполнения.
7. Вывести 2 строки с названием меню, массивом входящих в него блюд и даты, которые

принесут больше всего прибыли (численно и в % относительно себестоимости). Код запроса, результат выполнения.

Пусть блюдо состоит из ингридиентов $x_1, x_2, \dots, x_n$. Закупочная стоимость каждого ингридиента равна $c_1, c_2, \dots, c_n$ соответственно. Тогда я предполагаю, что себестоимость блюда равна $p = \sum_{i=1}^{n} c_i$ (я не учитываю, что блюдо содержит a грамм данного ингридиента).

Пусть меню на конкретный день состоит из блюд $y_1, y_2, \dots, y_m$. Тогда себестоимость меню будет равна $P = \sum_{j=1}^{m} p_j$.

Розничная цена на данное меню равна S. Тогда прибыль от данного меню равна $S - P$. Будем находить 2 меню: с наибольшим значением $S - P$ и с наибольшим значением $\frac{S-P}{P}$.

Заметим, что цена ингредиента указана в долларах, а цена меню в рублях. Поэтому для вычисления прибыли нужно умножить цену ингредиента на курс доллара. Курс доллара принять равным 1 доллар = 7.55 рублей (поправка на нереалистичную заполненность базы данных и предположение о составе блюда).

8. Какие пары блюд находятся вместе в меню чаще всего? Вывести их названия и количество нахождений вместе. Код запроса, результат выполнения.
9. Найти медианное количество заказов у пользователей. Код запроса, результат выполнения.

Если количество строк чётное, то медиана - это среднее двух средних значений. Если количество строк нечётное, то медиана - это значение в середине.

Если всего строк чётное количество, то total_rows / 2 и total_rows / 2 + 1 - целые числа, на выходе будет две строки и среднее арифметическое возьмётся от двух значений.

Если всего строк нечётное количество, то total_rows / 2 и total_rows / 2 + 1 - нецелые числа, на выходе будет одна строка и это и будет медиана.

10. Рассчитать ежедневную выручку сервиса, рассчитать ежедневный прирост выручки (численно и в %) относительно предыдущего дня. Код запроса, результат выполнения.
11. Построить иерархию приглашенных и пригласивших пользователей. Код запроса, результат выполнения.
12. Пусть сегодня 1 число какого-то месяца. У нас есть список заказов за предыдущий месяц. В предположении, что количество заказов на каждую позицию в меню останется таким же, вывести список ингредиентов, для которых следует нанять ещё поставщиков (кото-

рых не хватит для изготовления нужных блюд на ближайший месяц). Рассчитывать со следующим допущением: если блюдо весит a грамм и для его изготовления нужно n ингредиентов, то каждого ингредиента нужно в количестве $\frac{a}{n}$ грамм. Код запроса, результат выполнения.

5.6 Оценка размеров базы данных и каждого из файлов

Отношение	Атрибут	Тип данных	Размер, байт	Среднее количество	Объем, байт	Объём, МБ
users	id	integer	4	10000	1687552	1.61
	sex	gender	4			
	birth_date	date	4			
	invited_by_id	integer	4			
	bonuses	integer	4			
	address	varchar(200)	400			
	first_name	varchar(50)	100			
	middle_name	varchar(50)	100			
	last_name	varchar(50)	100			
	phone_number	varchar(10)	20			
dishes	id	integer	4	106	57344	0.05
	weight	double	8			
	colorfulness	double	8			
	title	varchar(100)	200			
	type	varchar(100)	200			
dishes_ingredients	id	integer	4	534	81920	0.08
	dish_id	integer	4			
	ingredient_id	integer	4			
ingredients	id	integer	4	149	57344	0.05
	title	varchar(100)	200			
	type	varchar(100)	200			
	cost	double	8			
dishes_menus	id	integer	4	360	57344	0.05
	dish_id	integer	4			
	menu_id	integer	4			
	date	date	4			
menus	id	integer	4	3	24576	0.02
	title	varchar(100)	200			
	cost	double	8			
	count_dishes	integer	4			
	colorfulness	double	8			

Отношение	Атрибут	Тип данных	Размер, байт	Среднее количество	Объем, байт	Объём, МБ
ingredients_suppliers	id	integer	4	408	81920	0.08
	ingredient_id	integer	4			
	supplier_id	integer	4			
suppliers	id	integer	4	20	24576	0.02
	title	varchar(100)	200			
	productivity	double	8			
payment_infos	id	integer	4	19936	1638400	1.56
	user_id	integer	4			
	payment_method_id	integer	4			
	requisites	varchar(100)	200			
payment_methods	id	integer	4	4	24576	0.02
	title	varchar(100)	200			
	require_requisites	boolean	1			
preferences_categories	id	integer	4	5	24576	0.02
	title	varchar(100)	200			
couriers	id	integer	4	100	57344	0.05
	first_name	varchar(50)	100			
	middle_name	varchar(50)	100			
	last_name	varchar(50)	100			
	birth_date	date	4			
preferences	id	integer	4	26	24576	0.02
	title	varchar(100)	200			
	preference_category_id	integer	4			
ingredients_preferences	id	integer	4	86	24576	0.02
	ingredient_id	integer	4			
	preference_id	integer	4			
orders	id	integer	4	50000	4587520	4.38
	user_id	integer	4			
	courier_id	integer	4			
	payment_info_id	integer	4			
	menu_id	integer	4			
	created_at	timestamp	8			
	status	order_status	4			
preferences_users	id	integer	4	9051	655360	0.62
	user_id	integer	4			
	preference_id	integer	4			

6 Приложение. Отчеты

6.1 Приложения

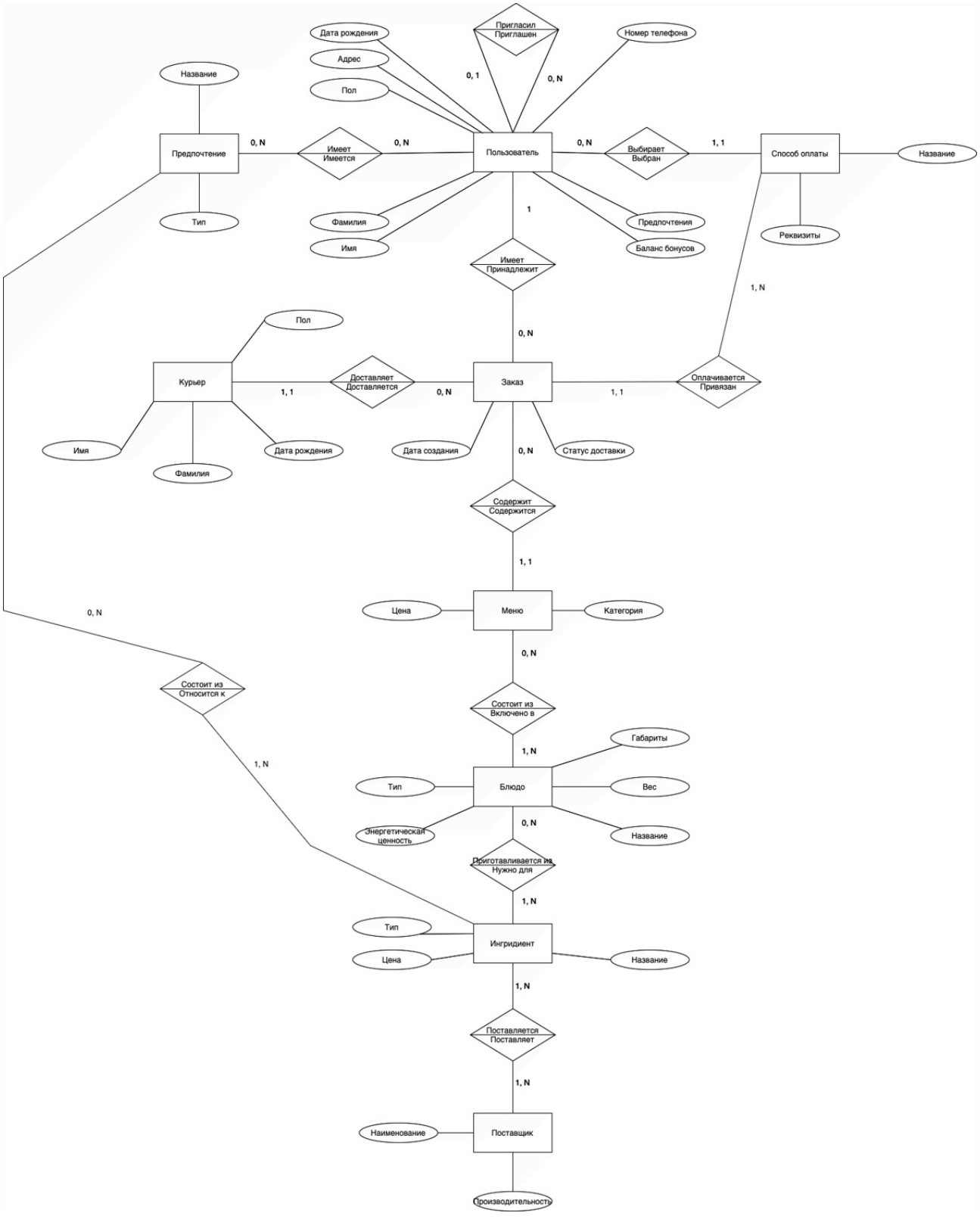


Рис. 1: ER-диаграмма концептуальной модели предметной области

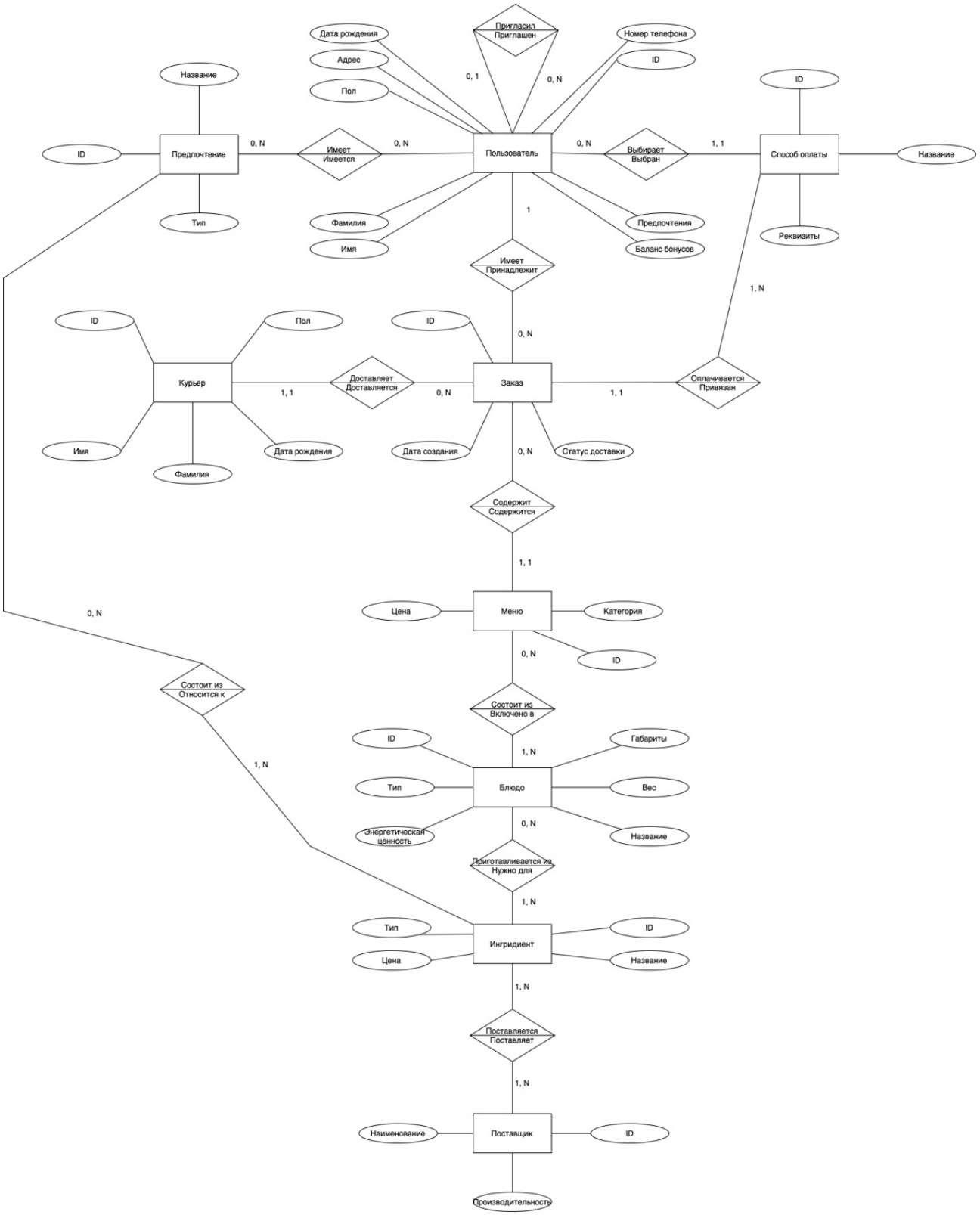


Рис. 2: ER-диаграмма концептуальной модели базы данных

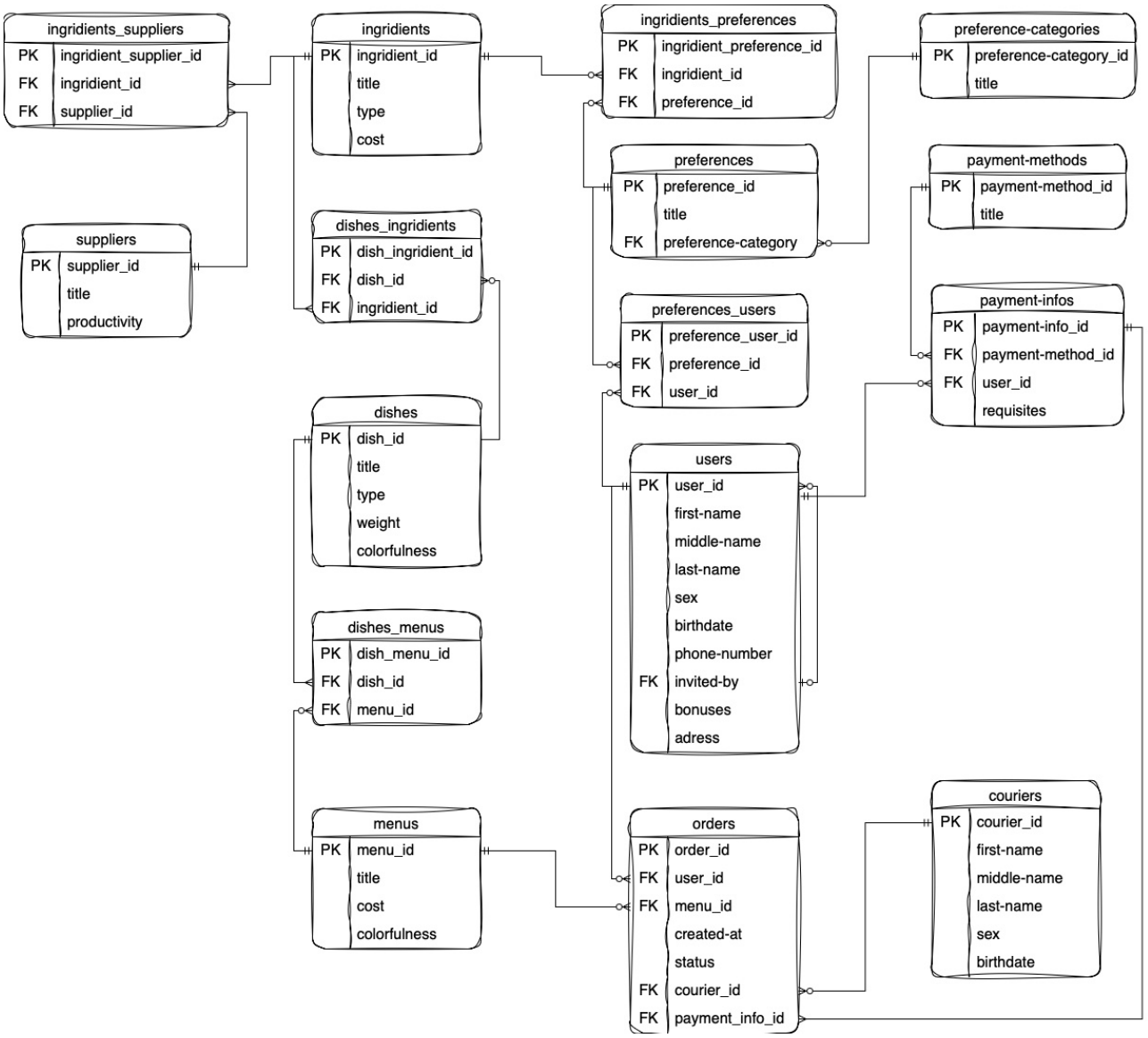


Рис. 3: ER-диаграмма логической модели базы данных

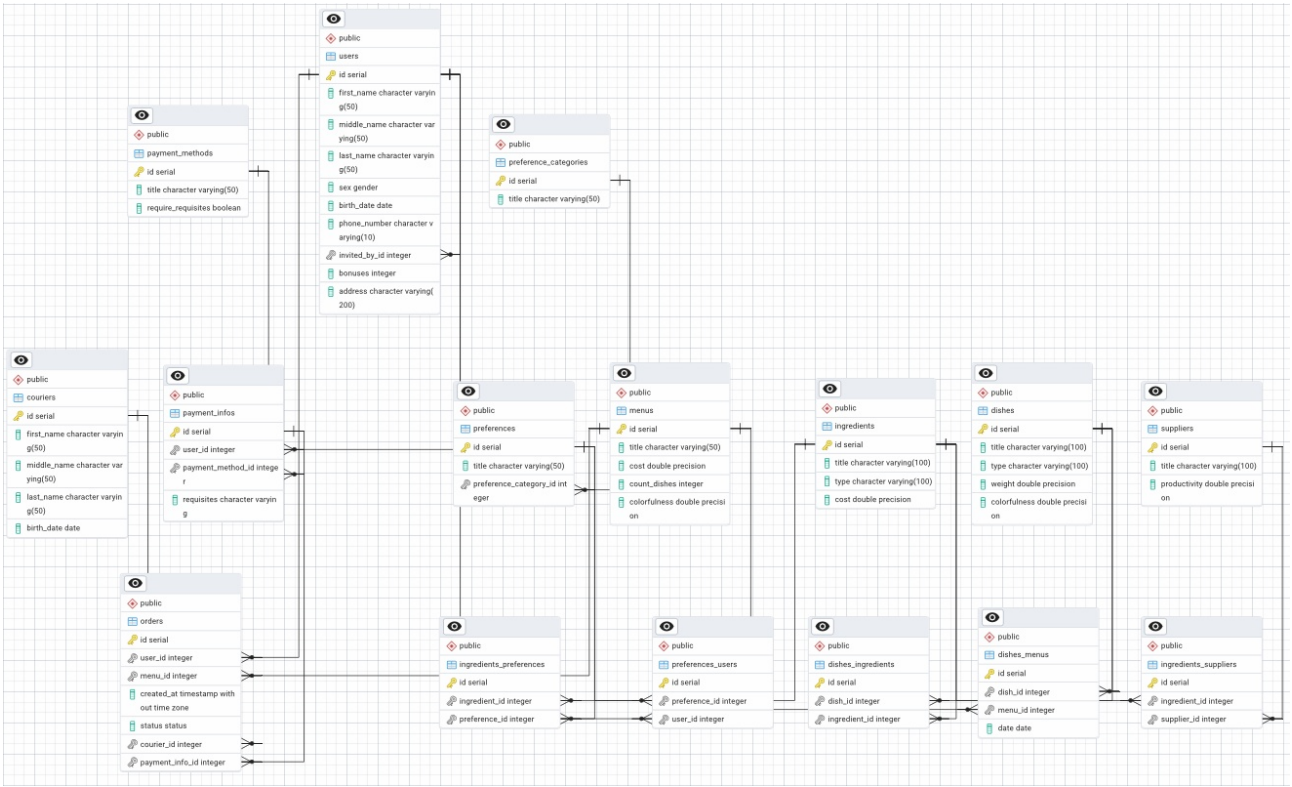


Рис. 4: Схемы отношений базы данных

Листинг 1: Создание ENUM-типов gender и status

```
CREATE TYPE gender AS ENUM ('male', 'female');

CREATE TYPE status AS ENUM ('new', 'in_delivery', 'delivered', 'canceled
    ', 'returned');
```

Листинг 2: Создание таблицы couriers

```
CREATE TABLE couriers (
    id SERIAL NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    middle_name VARCHAR(50),
    last_name VARCHAR(50) NOT NULL,
    birth_date DATE NOT NULL,
    PRIMARY KEY (id)
);
```

Листинг 3: Создание таблицы dishes

```
CREATE TABLE dishes (
    id SERIAL NOT NULL,
    title VARCHAR(100) NOT NULL,
    type VARCHAR(100) NOT NULL,
    weight FLOAT NOT NULL,
    colorfulness FLOAT NOT NULL,
```

```
    PRIMARY KEY (id)
);
```

Листинг 4: Создание таблицы ingredients

```
CREATE TABLE ingredients (
    id SERIAL NOT NULL,
    title VARCHAR(100) NOT NULL,
    type VARCHAR(100) NOT NULL,
    cost FLOAT NOT NULL,
    PRIMARY KEY (id)
);
```

Листинг 5: Создание таблицы menus

```
CREATE TABLE menus (
    id SERIAL NOT NULL,
    title VARCHAR(50) NOT NULL,
    cost FLOAT NOT NULL,
    count_dishes INTEGER NOT NULL,
    colorfulness FLOAT NOT NULL,
    PRIMARY KEY (id)
);
```

Листинг 6: Создание таблицы payment_methods

```
CREATE TABLE payment_methods (
    id SERIAL NOT NULL,
    title VARCHAR(50) NOT NULL,
    require_requisites BOOLEAN NOT NULL,
    PRIMARY KEY (id)
);
```

Листинг 7: Создание таблицы preference_categories

```
CREATE TABLE preference_categories (
    id SERIAL NOT NULL,
    title VARCHAR(50) NOT NULL,
    PRIMARY KEY (id)
);
```

Листинг 8: Создание таблицы suppliers

```
CREATE TABLE suppliers (
    id SERIAL NOT NULL,
    title VARCHAR(100) NOT NULL,
    productivity FLOAT NOT NULL,
    PRIMARY KEY (id)
);
```

Листинг 9: Создание таблицы users

```
CREATE TABLE users (
    id SERIAL NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    middle_name VARCHAR(50),
    last_name VARCHAR(50) NOT NULL,
    sex gender,
    birth_date DATE NOT NULL,
    phone_number VARCHAR(10) NOT NULL,
    invited_by_id INTEGER,
    bonuses INTEGER DEFAULT 0 NOT NULL,
    address VARCHAR(200),
    PRIMARY KEY (id),
    FOREIGN KEY(invited_by_id) REFERENCES users (id) ON DELETE SET NULL
);
```

Листинг 10: Создание таблицы dishes_ingredients

```
CREATE TABLE dishes_ingredients (
    id SERIAL NOT NULL,
    dish_id INTEGER NOT NULL,
    ingredient_id INTEGER NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY(dish_id) REFERENCES dishes (id) ON DELETE CASCADE,
    FOREIGN KEY(ingredient_id) REFERENCES ingredients (id) ON DELETE
        CASCADE
);
```

Листинг 11: Создание таблицы dishes_menus

```
CREATE TABLE dishes_menus (
    id SERIAL NOT NULL,
    dish_id INTEGER NOT NULL,
    menu_id INTEGER NOT NULL,
    date DATE DEFAULT 'now()' NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY(dish_id) REFERENCES dishes (id) ON DELETE CASCADE,
    FOREIGN KEY(menu_id) REFERENCES menus (id) ON DELETE CASCADE
);
```

Листинг 12: Создание таблицы ingredients_suppliers

```
CREATE TABLE ingredients_suppliers (
    id SERIAL NOT NULL,
    ingredient_id INTEGER NOT NULL,
    supplier_id INTEGER NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY(ingredient_id) REFERENCES ingredients (id) ON DELETE
        CASCADE,
    FOREIGN KEY(supplier_id) REFERENCES suppliers (id) ON DELETE CASCADE
```

```
);
```

Листинг 13: Создание таблицы payment_infos

```
CREATE TABLE payment_infos (
    id SERIAL NOT NULL,
    user_id INTEGER,
    payment_method_id INTEGER,
    requisites VARCHAR,
    PRIMARY KEY (id),
    FOREIGN KEY(user_id) REFERENCES users (id) ON DELETE CASCADE,
    FOREIGN KEY(payment_method_id) REFERENCES payment_methods (id) ON
        DELETE CASCADE
);
```

Листинг 14: Создание таблицы preferences

```
CREATE TABLE preferences (
    id SERIAL NOT NULL,
    title VARCHAR(50) NOT NULL,
    preference_category_id INTEGER NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY(preference_category_id) REFERENCES preference_categories
        (id) ON DELETE CASCADE
);
```

Листинг 15: Создание таблицы ingredients_preferences

```
CREATE TABLE ingredients_preferences (
    id SERIAL NOT NULL,
    ingredient_id INTEGER NOT NULL,
    preference_id INTEGER NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY(ingredient_id) REFERENCES ingredients (id) ON DELETE
        CASCADE,
    FOREIGN KEY(preference_id) REFERENCES preferences (id) ON DELETE
        CASCADE
);
```

Листинг 16: Создание таблицы orders

```
CREATE TABLE orders (
    id SERIAL NOT NULL,
    user_id INTEGER NOT NULL,
    menu_id INTEGER,
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT now() NOT NULL,
    status status DEFAULT 'new' NOT NULL,
    courier_id INTEGER,
    payment_info_id INTEGER,
    PRIMARY KEY (id),
    FOREIGN KEY(user_id) REFERENCES users (id) ON DELETE CASCADE,
```

```
    FOREIGN KEY(menu_id) REFERENCES menus (id) ON DELETE SET NULL,
    FOREIGN KEY(courier_id) REFERENCES couriers (id) ON DELETE SET NULL,
    FOREIGN KEY(payment_info_id) REFERENCES payment_infos (id) ON DELETE
        SET NULL
);
```

Листинг 17: Создание таблицы preferences_users

```
CREATE TABLE preferences_users (
    id SERIAL NOT NULL,
    preference_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY(preference_id) REFERENCES preferences (id) ON DELETE
       CASCADE,
    FOREIGN KEY(user_id) REFERENCES users (id) ON DELETE CASCADE
);
```

%python

Листинг 18: Подключение к базе данных с помощью psycopg2

```
from faker import Faker
import psycopg
import os

fake = Faker("ru_RU")

def get_conn():
    dsn = os.environ.get("DSN")
    if not dsn:
        raise RuntimeError("DSN not set in environment variables")

    return psycopg.connect(dsn, autocommit=True)
```

Листинг 19: Вызов функций заполнения всех таблиц

```
from seeders import *
from common import get_conn


def seed_all():
    with get_conn() as conn:
        seed_payment_methods(conn)
        seed_users(conn, 10000)
        seed_payment_infos(conn)

        seed_couriers(conn, 100)
        seed_menus(conn)
        seed_orders(conn, 50000, 100)
```

```
        seed_preference_categories(conn)
        seed_preferences(conn)
        seed_preferences_users(conn)

        seed_suppliers(conn, 20)
        seed_ingredients(conn)
        seed_ingredients_suppliers(conn)
        seed_dishes(conn)
        seed_dishes_ingredients(conn)
        seed_ingredients_preferences(conn)
        seed_dishes_menus(conn)
```

Листинг 20: Заполнение таблицы payment_methods

```
def seed_payment_methods(conn):
    payment_methods = [
        {
            "title": "Наличные курьеру",
            "require_requisites": False,
        },
        {
            "title": "Карта курьеру",
            "require_requisites": True,
        },
        {
            "title": "Карта",
            "require_requisites": True,
        },
        {
            "title": "ЯндексСплит.",
            "require_requisites": True,
        },
    ]

    with conn.cursor() as cur:
        for method in payment_methods:
            cur.execute(
                """
                INSERT INTO payment_methods (title, require_requisites)
                VALUES (%s, %s)
                """,
                (method["title"], method["require_requisites"]),
            )
    conn.commit()
    print("Inserted payment methods into the database.")
```

Листинг 21: Заполнение таблицы users

```
from common import fake
import datetime
import random
```

```
def generate_phone_number():
    """Форматирует номертелефонаввиде     10 цифрбезкодастраны     ."""
    return fake.numerify("9#########")


def generate_name_by_gender(gender):
    """Генерация имени, отчестваифамилииввзависимостиотпола          ."""
    if gender == "male":
        first_name = fake.first_name_male()
        middle_name = fake.middle_name_male()
        last_name = fake.last_name_male()
    else:
        first_name = fake.first_name_female()
        middle_name = fake.middle_name_female()
        last_name = fake.last_name_female()
    return first_name, middle_name, last_name

def generate_birth_date(min_age=18, max_age=80):
    """Генерация датырожденияввдиапазонеот      min_age до max_age летназад ."""
    today = datetime.date.today()
    start_date = today.replace(year=today.year - max_age)
    end_date = today.replace(year=today.year - min_age)
    return fake.date_between(start_date=start_date, end_date=end_date)

def seed_users(conn, n=50):
    with conn.cursor() as cur:
        for _ in range(n):
            sex = random.choice(["male", "female"])
            first_name, middle_name, last_name = generate_name_by_gender
                (sex)

            cur.execute(
                """
                INSERT INTO users (first_name, middle_name, last_name,
                    sex, birth_date, phone_number, address, bonuses)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                """,
                (
                    first_name,
                    middle_name,
                    last_name,
                    sex,
                    generate_birth_date(),
                    generate_phone_number(),
                    fake.street_address() if random.random() < 0.3 else
                        None,
                    0,
                ),
            )

        conn.commit()
        print(f"Inserted {n} users into the database.")
        cur.execute("SELECT id FROM users")
```

```
        all_users = cur.fetchall()
        for user_id in all_users:
            if random.random() < 0.3:
                cur.execute(
                    """
                    UPDATE users
                    SET invited_by_id = %s
                    WHERE id = %s
                    """,
                    (random.choice(all_users)[0], user_id[0]),
                )
        conn.commit()
        print("Updated invited_by_id for some users.")
```

Листинг 22: Заполнение таблицы payment_infos

```
from common import fake
import random

def seed_payment_infos(conn):
    with conn.cursor() as cur:
        cur.execute("SELECT id FROM users")
        users = cur.fetchall()

        cur.execute("SELECT id, require_requisites FROM payment_methods"
            )
        payment_methods = cur.fetchall()

        payment_infos = []
        for user in users:
            for payment_method in payment_methods:
                if random.random() < 0.5:
                    continue
                requisites = fake.credit_card_number() if payment_method
                    [1] else None
                payment_infos.append(
                    (user[0], payment_method[0], requisites)
                )

        cur.executemany(
            """
            INSERT INTO payment_infos (user_id, payment_method_id,
                requisites)
            VALUES (%s, %s, %s)
            """,
            payment_infos
        )
        conn.commit()
        print(f"Inserted {len(payment_infos)} payment infos into the
            database.")
```

Листинг 23: Заполнение таблицы couriers

```
from common import fake
import datetime

def generate_courier_name():
    first_name = fake.first_name_male()
    middle_name = fake.middle_name_male()
    last_name = fake.last_name_male()
    return first_name, middle_name, last_name


def generate_birth_date(min_age=18, max_age=60):
    today = datetime.date.today()
    start_date = today.replace(year=today.year - max_age)
    end_date = today.replace(year=today.year - min_age)
    return fake.date_between(start_date=start_date, end_date=end_date)

def seed_couriers(conn, n=50):
    with conn.cursor() as cur:
        for _ in range(n):
            first_name, middle_name, last_name = generate_courier_name()
            birth_date = generate_birth_date()

            cur.execute(
                """
                INSERT INTO couriers (first_name, middle_name, last_name
                    , birth_date)
                VALUES (%s, %s, %s, %s)
                """,
                (first_name, middle_name, last_name, birth_date),
            )
        conn.commit()
        print(f"Inserted {n} couriers into the database.")
```

Листинг 24: Заполнение таблицы menus

```
def seed_menus(conn):
    menus = [
        {
            "title": "Похудение",
            "cost": 700,
            "count_dishes": 3,
            "colorfulness": 1000,
        },
        {
            "title": "Баланс",
            "cost": 850,
            "count_dishes": 4,
            "colorfulness": 1500,
        },
        {
            "title": "Набор",
            "cost": 1000,
```

```
            "count_dishes": 5,
            "colorfulness": 2000,
        },
    ]

    with conn.cursor() as cur:
        for menu in menus:
            cur.execute(
                """
                INSERT INTO menus (title, cost, count_dishes,
                    colorfulness)
                VALUES (%s, %s, %s, %s)
                """,
                (menu["title"], menu["cost"], menu["count_dishes"], menu
                    ["colorfulness"]),
            )

        conn.commit()
        print(f"Inserted {len(menus)} menus into the database.")
```

Листинг 25: Заполнение таблицы orders

```
from common import fake
import random
import datetime

def generate_order_created_time(duration):
    now = datetime.datetime.now()
    start_time = now - datetime.timedelta(days=duration)
    return fake.date_time_between(start_date=start_time, end_date=now)

def generate_order_status(created_time):
    now = datetime.datetime.now()
    if now - created_time <= datetime.timedelta(days=1):
        status_weights = {
            'new': 0.5,
            'in_delivery': 0.5,
        }
        status = random.choices(
            list(status_weights.keys()), list(status_weights.values()),
                k=1
        )[0]
    elif datetime.timedelta(days=1) < now - created_time <= datetime.
        timedelta(days=3):
        status_weights = {'in_delivery': 0.1, 'delivered': 0.9}
        status = random.choices(
            list(status_weights.keys()), list(status_weights.values()),
                k=1
        )[0]
    else:
        status = 'delivered'

    if status == 'delivered' and random.random() < 0.05:
```

```
        status = 'returned'

    if random.random() < 0.05:
        status = 'canceled'

    return status

def seed_orders(conn, n=100, duration=30):
    with conn.cursor() as cur:
        cur.execute("SELECT id FROM users")
        users = cur.fetchall()

        cur.execute("SELECT id FROM menus")
        menus = cur.fetchall()

        cur.execute("SELECT id FROM couriers")
        couriers = cur.fetchall()

        cur.execute("SELECT id FROM payment_infos WHERE user_id IN (
           SELECT id FROM users)")
        payment_infos = cur.fetchall()

        orders = []
        for _ in range(n):
            user = random.choice(users)
            menu = random.choice(menus)
            courier = random.choice(couriers)
            payment_info = random.choice(payment_infos)
            created_at = generate_order_created_time(duration)

            order = (
                user[0],
                menu[0],
                created_at,
                generate_order_status(created_at),
                courier[0],
                payment_info[0],
            )
            orders.append(order)

        cur.executemany(
            """
            INSERT INTO orders (user_id, menu_id, created_at, status,
               courier_id, payment_info_id)
            VALUES (%s, %s, %s, %s, %s, %s)
            """,
            orders,
        )
        conn.commit()
        print(f"Inserted {len(orders)} orders into the database.")

        # Update user bonuses
        for order in orders:
```

```
            user_id = order[0]
            menu_id = order[1]
            cur.execute(
                """
                UPDATE users
                SET bonuses = bonuses + (SELECT cost FROM menus WHERE id
                    = %s) * 0.05
                WHERE id = %s
                """,
                (menu_id, user_id),
            )
        conn.commit()
        print("Updated user bonuses based on orders.")
```

Листинг 26: Заполнение таблицы preference_categories

```
def seed_preference_categories(conn):
    categories = [
        "аллергены",
        "десерты, выпечка, сахар",
        "мясо, рыба",
        "овощи, лук, чеснок",
        "гарниры, каши"
    ]
    with conn.cursor() as cur:
        for category in categories:
            cur.execute(
                """
                INSERT INTO preference_categories (title)
                VALUES (%s)
                """,
                (category,),
            )
        conn.commit()
        print(f"Inserted {len(categories)} preference categories into
           the database.")
```

Листинг 27: Заполнение таблицы preferences

```
def seed_preferences(conn):
    preferences = [
        {"title": "Без творога", "preference_category_id": 1},
        {"title": "Без орехов", "preference_category_id": 1},
        {"title": "Без меда", "preference_category_id": 1},
        {"title": "Без морепродуктов", "preference_category_id": 1},
        {"title": "Без горчицы", "preference_category_id": 1},
        {"title": "Без шоколада", "preference_category_id": 1},

        {"title": "Без десертов", "preference_category_id": 2},
        {"title": "Без выпечки", "preference_category_id": 2},
        {"title": "Без сэндвичейикруассанов  ", "preference_category_id": 2},
        {"title": "Без белогосахара ", "preference_category_id": 2},
```

```
        {"title": "Без свининыиветчины  ", "preference_category_id": 3},
        {"title": "Без красногомяса ", "preference_category_id": 3},
        {"title": "Без рыбы", "preference_category_id": 3},
        {"title": "Без мясаиптицы  ", "preference_category_id": 3},

        {"title": "Без сельдерея", "preference_category_id": 4},
        {"title": "Без грибов", "preference_category_id": 4},
        {"title": "Без стручковойфасоли ", "preference_category_id": 4},
        {"title": "Без брокколи", "preference_category_id": 4},
        {"title": "Без кабачков", "preference_category_id": 4},
        {"title": "Без лука", "preference_category_id": 4},
        {"title": "Без чеснока", "preference_category_id": 4},

        {"title": "Без нута", "preference_category_id": 5},
        {"title": "Без булгура", "preference_category_id": 5},
        {"title": "Без кускуса", "preference_category_id": 5},
        {"title": "Без гречки", "preference_category_id": 5},
        {"title": "Без молочныхкаш ", "preference_category_id": 5}
    ]

    with conn.cursor() as cur:
        for preference in preferences:
            cur.execute(
                """
                INSERT INTO preferences (title, preference_category_id)
                VALUES (%s, %s)
                """,
                (preference["title"], preference["preference_category_id
                    "]),
            )
        conn.commit()
        print(f"Inserted {len(preferences)} preferences into the
            database.")
```

Листинг 28: Заполнение таблицы preferences_users

```
import random

def seed_preferences_users(conn):
    with conn.cursor() as cur:
        cur.execute("SELECT id FROM users")
        users = cur.fetchall()

        cur.execute("SELECT id FROM preferences")
        preferences = cur.fetchall()

        users = [user[0] for user in users]
        preferences = [pref[0] for pref in preferences]

        for user in random.sample(users, int(len(users) * 0.3)):
            selected_prefs = random.sample(preferences, random.randint
                (1, 5))
```

```
            for pref_id in selected_prefs:
                cur.execute(
                    """
                    INSERT INTO preferences_users (user_id,
                        preference_id)
                    VALUES (%s, %s)
                    """,
                    (user, pref_id),
                )
        conn.commit()
        print("Inserted preferences_users into the database.")
```

Листинг 29: Заполнение таблицы suppliers

```
from common import fake
import random

def seed_suppliers(conn, n=10):
    with conn.cursor() as cur:
        for _ in range(n):
            title = fake.company()
            productivity = random.randint(1, 200)

            cur.execute(
                """
                INSERT INTO suppliers (title, productivity)
                VALUES (%s, %s)
                """,
                (title, productivity),
            )
        conn.commit()
        print(f"Inserted {n} suppliers into the database.")
```

Листинг 30: Заполнение таблицы ingredients

```
import csv
import os

def seed_ingredients(conn):
    file_path = os.path.join(os.path.dirname(__file__), "csv", "
        ingredients.csv")
    with open(file_path, "r") as file:
        reader = csv.DictReader(file)
        ingredients = []

        for row in reader:
            ingredient = {
                "title": row["title"],
                "type": row["type"],
                "cost": float(row["cost"]),
            }
            ingredients.append(ingredient)
```

```
    with conn.cursor() as cur:
        for ingredient in ingredients:
            cur.execute(
                """
                INSERT INTO ingredients (title, type, cost)
                VALUES (%s, %s, %s)
                """,
                (ingredient["title"], ingredient["type"], ingredient["
                    cost"]),
            )
    conn.commit()
    print(f"Inserted {len(ingredients)} ingredients into the database.")
```

Листинг 31: Заполнение таблицы ingredients_suppliers

```
import random

def seed_ingredients_suppliers(conn):
    with conn.cursor() as cur:
        cur.execute("SELECT id FROM ingredients")
        ingredients = cur.fetchall()

        cur.execute("SELECT id FROM suppliers")
        suppliers = cur.fetchall()

        ingredients_suppliers = []
        for ingredient in ingredients:
            random_suppliers = random.sample(suppliers, k=random.randint
                (1, 5))

            for supplier in random_suppliers:
                ingredients_suppliers.append(
                    (ingredient[0], supplier[0])
                )

        cur.executemany(
            """
            INSERT INTO ingredients_suppliers (ingredient_id,
                supplier_id)
            VALUES (%s, %s)
            """,
            ingredients_suppliers
        )
        conn.commit()
        print(f"Inserted {len(ingredients_suppliers)} ingredients-
            suppliers into the database.")
```

Листинг 32: Заполнение таблицы dishes

```
import csv
import os
```

```
def seed_dishes(conn):
    file_path = os.path.join(os.path.dirname(__file__), "csv", "dishes.
        csv")
    with open(file_path, "r") as file:
        reader = csv.DictReader(file)
        dishes = []

        for row in reader:
            dish = {
                "title": row["title"],
                "type": row["type"],
                "weight": float(row["weight"]),
                "colorfulness": float(row["colorfulness"]),
            }
            dishes.append(dish)
    with conn.cursor() as cur:
        for dish in dishes:
            cur.execute(
                """
                INSERT INTO dishes (title, type, weight, colorfulness)
                VALUES (%s, %s, %s, %s)
                """,
                (dish["title"], dish["type"], dish["weight"], dish["
                    colorfulness"]),
            )
    conn.commit()
    print(f"Inserted {len(dishes)} dishes into the database.")
```

Листинг 33: Заполнение таблицы dishes_ingredients

```
import os
import csv

def seed_dishes_ingredients(conn):
    file_path = os.path.join(os.path.dirname(__file__), "csv", "
        dishes_ingredients.csv")
    with open(file_path, "r") as file:
        reader = csv.DictReader(file)
        values = []

        for row in reader:
            value = {
                "dish_id": row["dish_id"],
                "ingredient_id": row["ingredient_id"],
            }
            values.append(value)

    with conn.cursor() as cur:
        cur.executemany(
            """
            INSERT INTO dishes_ingredients (dish_id, ingredient_id)
            VALUES (%s, %s)
```

```
            """,
            [(value["dish_id"], value["ingredient_id"]) for value in
                values]
        )
        conn.commit()
    print(f"Inserted {len(values)} dishes_ingredients into the database.
        ")
```

Листинг 34: Заполнение таблицы ingredients_preferences

```
import csv
import os

def seed_ingredients_preferences(conn):
    file_path = os.path.join(os.path.dirname(__file__), "csv", "
        ingredients_preferences.csv")
    with open(file_path, "r") as file:
        reader = csv.DictReader(file)
        values = []

        for row in reader:
            value = {
                "ingredient_id": row["ingredient_id"],
                "preference_id": row["preference_id"]
            }
            values.append(value)

    with conn.cursor() as cur:
        for value in values:
            cur.execute(
                """
                INSERT INTO ingredients_preferences (ingredient_id,
                    preference_id)
                VALUES (%s, %s)
                """,
                (value["ingredient_id"], value["preference_id"]),
            )
    conn.commit()
    print(f"Inserted {len(values)} ingredients preferences into the
        database.")
```

Листинг 35: Заполнение таблицы dishes_menus

```
import datetime
import random

def seed_dishes_menus(conn):
    today = datetime.date.today()
    date_range = [today + datetime.timedelta(days=i) for i in range(30)]

    with conn.cursor() as cur:
        # Получаемвсеменю
```

```
        cur.execute("SELECT id, count_dishes, colorfulness FROM menus")
        menus = cur.fetchall()  # списоккортежей : (id, count_dishes,
            colorfulness)

        for date in date_range:
            for menu_id, count_dishes, max_color in menus:
                selected_dishes = []
                colorfulness_accumulated = 0

                while len(selected_dishes) < count_dishes:
                    # Выбираемблюда , которыевписываютсявограничения
                    cur.execute(
                        """
                        SELECT id, colorfulness FROM dishes
                        WHERE colorfulness <= %s
                        """,
                        (max_color - colorfulness_accumulated,)
                    )
                    dishes = cur.fetchall()  # [(id, colorfulness), ...]

                    if not dishes:
                        break  # нетподходящихблюд—выходим

                    dish = random.choice(dishes)

                    if colorfulness_accumulated + dish[1] <= max_color:
                        selected_dishes.append(dish)
                        colorfulness_accumulated += dish[1]

                # Добавляемвыбранныеблюдавтаблицу
                for dish_id, _ in selected_dishes:
                    cur.execute(
                        """
                        INSERT INTO dishes_menus (dish_id, menu_id, date
                            )
                        VALUES (%s, %s, %s)
                        """,
                        (dish_id, menu_id, date)
                    )

        conn.commit()
```

	id	first_name	middle_name	last_name	sex	birth_date	phone_number	invited_by_id	bonuses	address
0	14	Демьян	Ильич	Романов	male	1991-05-11	9419405564	NaN	0	None
1	47	Панфил	Валерьянович	Назаров	male	1993-02-12	9381812017	NaN	0	None
2	1	Фёкла	Владиславовна	Мартынова	female	1982-02-01	9896072920	NaN	128	пр. Леонова, д. 537
3	2	Раиса	Кузьминична	Игнатова	female	1955-05-26	9190322073	NaN	78	пр. Щербакова, д. 114
4	4	Евпраксия	Мироновна	Гуляева	female	2003-05-26	9336371496	NaN	220	алл. Мусы Джалиля, д. 9/9
5	5	Пелагея	Оскаровна	Комарова	female	1960-12-18	9844123848	NaN	232	наб. Докучаева, д. 36 к. 424
6	6	Оксана	Кирилловна	Беспалова	female	1974-06-18	9524441984	NaN	212	ш. Чайковского, д. 8
7	9	Нонна	Олеговна	Лобанова	female	1986-10-17	9268817100	NaN	212	пр. Октября, д. 72 к. 696
8	10	Ираклий	Бенедиктович	Максимов	male	1998-05-26	9920674128	NaN	298	бул. Красина, д. 2 к. 38
9	11	Мариан	Александрович	Стрелков	male	1952-10-06	9966287019	NaN	128	ш. Культуры, д. 3/5 стр. 345
10	15	Тамара	Аркадьевна	Жукова	female	2006-03-19	9333688096	NaN	155	бул. Бригадный, д. 4/6 стр. 1
11	16	Виктория	Валентиновна	Калашникова	female	1963-03-16	9905105073	NaN	270	ул. Громова, д. 6
12	17	Герман	Бенедиктович	Ермаков	male	1952-03-09	9511625218	NaN	185	ул. Юбилейная, д. 1 стр. 34
13	20	Фотий	Валерьевич	Стрелков	male	1947-12-19	9491186390	NaN	178	наб. Аэродромная, д. 65 стр. 36
14	21	Майя	Львовна	Бурова	female	1997-02-02	9504390341	NaN	178	бул. Монтажников, д. 9 к. 856

Рис. 5: Заполненная таблица “users”

	id	title	require_requisites
0	1	Наличные курьеру	False
1	2	Карта курьеру	True
2	3	Карта	True
3	4	Яндекс.Сплит	True

Рис. 6: Заполненная таблица “payment_methods”

	id	user_id	payment_method_id	requisites
0	1	1	2	2201907139552375
1	2	1	3	349717068130383
2	3	1	4	8118732266029974
3	4	2	3	4643746123709936
4	5	2	4	5148949858501083
5	6	4	3	377704027535519
6	7	5	1	None
7	8	5	2	8197220808101745
8	9	5	4	2704328270012826
9	10	6	1	None
10	11	6	2	370127313355573
11	12	6	3	2201107745986451
12	13	9	1	None
13	14	9	3	6309636871527178
14	15	10	2	2683778571749002

Рис. 7: Заполненная таблица “payment_infos”

	id	title
0	1	аллергены
1	2	десерты, выпечка, сахар
2	3	мясо, рыба
3	4	овощи, лук, чеснок
4	5	гарниры, каши

Рис. 8: Заполненная таблица “preference_categories”

	id	title	preference_category_id
0	1	Без творога	1
1	2	Без орехов	1
2	3	Без меда	1
3	4	Без морепродуктов	1
4	5	Без горчицы	1
5	6	Без шоколада	1
6	7	Без десертов	2
7	8	Без выпечки	2
8	9	Без сэндвичей и круассанов	2
9	10	Без белого сахара	2
10	11	Без свинины и ветчины	3
11	12	Без красного мяса	3
12	13	Без рыбы	3
13	14	Без мяса и птицы	3
14	15	Без сельдерея	4

Рис. 9: Заполненная таблица “preferences”

	id	first_name	middle_name	last_name	birth_date
0	1	Демьян	Викторович	Рожков	1991-04-03
1	2	Измаил	Ефимьевич	Константинов	1988-05-06
2	3	Всеволод	Гертрудович	Бирюков	1965-11-29
3	4	Мокей	Александрович	Маслов	1973-06-16
4	5	Ефим	Илларионович	Селиверстов	1973-09-06
5	6	Станислав	Якубович	Самойлов	1979-04-21
6	7	Лаврентий	Августович	Виноградов	1990-11-01
7	8	Архип	Гурьевич	Егоров	2003-12-15
8	9	Феофан	Феофанович	Нестеров	1993-06-02
9	10	Лукьян	Валентинович	Хохлов	1972-08-20
10	11	Эраст	Аверьянович	Данилов	1979-12-08
11	12	Василий	Ефимьевич	Калашников	1991-11-25
12	13	Вацлав	Ермилович	Гришин	1990-07-20
13	14	Пимен	Эдгардович	Абрамов	1992-07-18
14	15	Никодим	Эдуардович	Макаров	1967-03-18

Рис. 10: Заполненная таблица “couriers”

	id	title	cost	count_dishes	colorfulness
0	1	Похудение	700.0	3	1000.0
1	2	Баланс	850.0	4	1500.0
2	3	Набор	1000.0	5	2000.0

Рис. 11: Заполненная таблица “menus”

	id	user_id	menu_id	created_at	status	courier_id	payment_info_id
0	1	1563	1	2024-08-23 02:38:24.791040	delivered	914	4770
1	2	3793	3	2024-10-24 13:24:12.374363	delivered	128	20906
2	3	6084	3	2024-10-29 18:55:40.730061	delivered	262	13826
3	4	14926	1	2024-10-18 06:45:32.744788	delivered	765	23951
4	5	1731	3	2024-11-03 02:24:43.124879	delivered	92	8290
5	6	1499	3	2024-11-03 09:22:10.711858	delivered	594	7856
6	7	10796	2	2024-08-25 11:24:10.927954	delivered	28	27027
7	8	6288	1	2024-09-06 02:24:32.935803	delivered	987	13893
8	9	12789	2	2024-10-14 04:27:13.514588	delivered	200	538
9	10	8749	2	2024-11-14 13:53:48.831527	delivered	938	11768
10	11	7636	2	2024-11-26 22:15:27.299888	new	503	11547
11	12	5434	1	2024-09-24 10:39:06.321188	returned	624	271
12	13	12614	3	2024-11-14 07:35:27.733649	delivered	570	16965
13	14	12391	2	2024-10-14 04:28:40.513829	delivered	658	24199
14	15	12069	2	2024-08-30 11:19:44.221203	delivered	52	17097

Рис. 12: Заполненная таблица “orders”

	id	title	type	weight	colorfulness
0	1	Окрошка	Суп	250.0	150.0
1	2	Борщ	Суп	300.0	250.0
2	3	Щи	Суп	300.0	230.0
3	4	Солянка	Суп	350.0	350.0
4	5	Рассольник	Суп	250.0	220.0
5	6	Пельмени	Второе	350.0	450.0
6	7	Вареники с картошкой	Второе	400.0	400.0
7	8	Жаркое	Второе	450.0	600.0
8	9	Котлеты по-киевски	Второе	300.0	550.0
9	10	Сельдь под шубой	Салат	250.0	300.0
10	11	Оливье	Салат	200.0	320.0
11	12	Винегрет	Салат	180.0	200.0
12	13	Мимоза	Салат	250.0	350.0
13	14	Шуба с рыбой	Салат	220.0	330.0
14	15	Курник	Выпечка	350.0	600.0

Рис. 13: Заполненная таблица “dishes”

	id	dish_id	menu_id	date
0	1	26	1	2024-11-27
1	2	83	1	2024-11-27
2	3	26	1	2024-11-27
3	4	11	2	2024-11-27
4	5	44	2	2024-11-27
5	6	96	2	2024-11-27
6	7	89	2	2024-11-27
7	8	13	3	2024-11-27
8	9	55	3	2024-11-27
9	10	49	3	2024-11-27
10	11	102	3	2024-11-27
11	12	85	3	2024-11-27
12	13	40	1	2024-11-28
13	14	40	1	2024-11-28
14	15	59	1	2024-11-28

Рис. 14: Заполненная таблица “dishes_menus”

	id	title	type	cost
0	1	Куриное филе	Мясо	4.99
1	2	Говядина	Мясо	6.49
2	3	Свинина	Мясо	5.99
3	4	Кролик	Мясо	7.99
4	5	Баранина	Мясо	8.99
5	6	Индейка	Мясо	5.49
6	7	Фарш мясной	Мясо	4.99
7	8	Лосось	Рыба	10.99
8	9	Треска	Рыба	7.49
9	10	Сельдь	Рыба	4.99
10	11	Скумбрия	Рыба	6.49
11	12	Камбала	Рыба	8.99
12	13	Морепродукты	Морепродукты	12.99
13	14	Окунь	Рыба	9.49
14	15	Крабовое мясо	Морепродукты	14.99

Рис. 15: Заполненная таблица “ingredients”

	id	dish_id	ingredient_id
0	1	1	70
1	2	1	92
2	3	1	130
3	4	1	42
4	5	1	44
5	6	1	100
6	7	1	14
7	8	2	48
8	9	2	134
9	10	2	22
10	11	2	107
11	12	2	130
12	13	2	101
13	14	3	69
14	15	3	12

Рис. 16: Заполненная таблица “dishes_ingredients”

	id	title	productivity
0	1	Котов Групп	134.0
1	2	ООО «Гуляев»	54.0
2	3	Евсеева Групп	122.0
3	4	ООО «Лазарев-Коновалов»	166.0
4	5	ОАО «Туров-Павлова»	35.0
5	6	ИП «Федосеева»	112.0
6	7	Уральский банк реконструкции и развития	105.0
7	8	АО «Селезнева Евсеева»	24.0
8	9	ЗАО «Воронцов, Федосеева и Александрова»	26.0
9	10	НПО «Попов-Некрасова»	191.0
10	11	АО «Назаров, Белозеров и Кузьмин»	161.0
11	12	РАО «Никифоров, Рябов и Субботина»	33.0
12	13	РАО «Кудряшова»	65.0
13	14	АО «Калашников-Калинин»	67.0
14	15	Родионов и партнеры	117.0

Рис. 17: Заполненная таблица “suppliers”

	id	ingredient_id	supplier_id
0	1	1	366
1	2	1	88
2	3	1	357
3	4	1	450
4	5	2	63
5	6	2	470
6	7	2	89
7	8	3	393
8	9	3	128
9	10	4	62
10	11	4	244
11	12	4	472
12	13	4	162
13	14	5	79
14	15	5	467

Рис. 18: Заполненная таблица “ingredients_suppliers”

	id	ingredient_id	preference_id
0	1	26	1
1	2	67	2
2	3	68	2
3	4	69	2
4	5	70	2
5	6	71	2
6	7	72	2
7	8	73	2
8	9	74	2
9	10	75	2
10	11	76	2
11	12	77	2
12	13	78	2
13	14	143	3
14	15	12	4

Рис. 19: Заполненная таблица “ingredients_preferences”

Листинг 36: Запрос 1 на SQL

```
WITH
    orders_last_month AS (
        SELECT
            *
        FROM
            orders
        WHERE
            created_at >= CURRENT_DATE - INTERVAL '1 month'
    )

SELECT
    SUM(m.cost) AS Выручка
FROM
    orders_last_month AS o
    INNER JOIN menus AS m on o.menu_id = m.id
```

	Выручка
0	13422150.0

Рис. 20: Результат выполнения запроса 1

Листинг 37: Запрос 2 на SQL

```
SELECT
    ROUND(
        SUM(m.cost) FILTER (WHERE m.title = 'Похудение')::DECIMAL /
        SUM(m.cost)::DECIMAL * 100, 2) AS losing_weight_revenue_ratio
FROM
    orders AS o
    INNER JOIN menus AS m on o.menu_id = m.id
```

	losing_weight_revenue_ratio
0	27.56

Рис. 21: Результат выполнения запроса 2

Листинг 38: Запрос 3 на SQL

```
SELECT
    title,
    cost,
    ROUND(cost::DECIMAL / 1.2 * 0.2, 2) AS tax,
    ROUND(cost::DECIMAL / 1.2, 2) AS cost_before_tax
FROM
    menus
```

	title	cost	tax	cost_before_tax
0	Похудение	700.0	116.67	583.33
1	Баланс	850.0	141.67	708.33
2	Набор	1000.0	166.67	833.33

Рис. 22: Результат выполнения запроса 3

Листинг 39: Запрос 4 на SQL

```
WITH
    yandex_split_payments AS (
        SELECT
            i.*
        FROM
```

```
            payment_infos i
            JOIN payment_methods m ON i.payment_method_id = m.id
        WHERE
            m.title = 'ЯндексСплит.'
    )

SELECT
    o.id AS order_id,
    DATE(o.created_at) AS order_created_at,
    p.requisites AS requisites
FROM
    orders o
    JOIN yandex_split_payments p ON o.payment_info_id = p.id
WHERE
    o.status = 'returned'
```

	order_id	order_created_at	requisites
0	10	2024-09-17	348003776203841
1	86	2024-11-02	2257648768555601
2	110	2024-09-21	8100939730499929
3	136	2024-11-16	6635525845179221
4	432	2024-11-09	4037182764191594
...	...	...	...
591	49466	2024-11-14	4214473832989134
592	49543	2024-10-04	6908438154829347
593	49789	2024-08-24	6546165912392635
594	49880	2024-10-09	6831288755820316
595	49953	2024-11-20	2712010099594008

596 rows × 3 columns

Рис. 23: Результат выполнения запроса 4

Листинг 40: Запрос 5 на SQL

```
WITH
    payment_infos_and_methods AS (
        SELECT
            i.id AS payment_info_id,
            m.title AS payment_method
        FROM
            payment_infos i
            JOIN payment_methods m ON i.payment_method_id = m.id
    ),
    user_method_counts AS (
        SELECT
            o.user_id,
            p.payment_method,
            COUNT(o.id) AS orders_count
        FROM
```

```
            orders o
            JOIN payment_infos_and_methods p USING (payment_info_id)
        GROUP BY
            o.user_id,
            p.payment_method
    ),
    user_method_counts_with_total AS (
        SELECT
            *,
            SUM(orders_count) OVER (
                PARTITION BY
                    user_id
            ) AS total_orders_count
        FROM
            user_method_counts
    ),
    users_with_almost_cash_orders AS (
        SELECT
            user_id AS id
        FROM
            user_method_counts_with_total
        WHERE
            payment_method = 'Наличные курьеру'
            AND orders_count > total_orders_count / 2
    )

SELECT
    u.id,
    CONCAT(u.first_name, ' ', u.middle_name, ' ', u.last_name) AS name,
    u.bonuses
FROM
    users u
    JOIN users_with_almost_cash_orders u_cash USING (id)
ORDER BY
    u.bonuses DESC
LIMIT
    5;
```

	id	name	bonuses
0	1227	Ольга Кузьминична Соколова	410
1	5673	Назар Анатольевич Орехов	398
2	3924	Ирина Геннадиевна Петухова	398
3	2461	Александра Романовна Кондратьева	398
4	7821	Ульяна Робертовна Юдина	390

Рис. 24: Результат выполнения запроса 5

Листинг 41: Запрос 6 на SQL

```
WITH
    most_popular_category AS (
        SELECT
            preference_category_id,
            COUNT(pu.id) AS count
        FROM
            preferences_users pu
            JOIN preferences p ON pu.preference_id = p.id
        GROUP BY
            preference_category_id
        ORDER BY
            count DESC
        LIMIT
            1
    ),
    preferences_from_most_popular_category AS (
        SELECT
            id,
            title
        FROM
            preferences
        WHERE
            preference_category_id = (
                SELECT
                    preference_category_id
                FROM
                    most_popular_category
            )
    ),
    ingredients_from_most_popular_category AS (
        SELECT
            ip.ingredient_id
        FROM
            ingredients_preferences ip
        JOIN preferences_from_most_popular_category p ON ip.
           preference_id = p.id
    )

SELECT
    DISTINCT d.title
FROM
    dishes_ingredients
    JOIN ingredients_from_most_popular_category i ON dishes_ingredients.
       ingredient_id = i.ingredient_id
    JOIN dishes d ON dishes_ingredients.dish_id = d.id
ORDER BY
    d.title;
```

	title
0	Борщ
1	Борщ с капустой
2	Вареники с картошкой
3	Винегрет
4	Гречка с грибами
5	Гречка с овощами
6	Гречневая каша
7	Жаркое
8	Запеканка с творогом
9	Каша овсяная
10	Каша пшенная
11	Каша рисовая с молоком
12	Котлеты по-киевски
13	Медовик
14	Морс
15	Пельмени с говядиной
16	Печенье "Маковое"
17	Печенье с орехами
18	Пирог с вишней
19	Пирог с картошкой и грибами
20	Пирог с мясом
21	Пирог с мясом и картошкой
22	Пирог с рыбой
23	Ризотто с грибами
24	Рыбный суп
25	Сельдь под шубой
26	Солянка с мясом
27	Суп из чечевицы
28	Суп с фрикадельками
29	Творожная запеканка
30	Творожный десерт
31	Торт "Птичье молоко"
32	Шарлотка
33	Шуба с рыбой
34	Щи

Рис. 25: Результат выполнения запроса 6

Листинг 42: Запрос 7 на SQL

```
WITH
    ingredients_costs_in_rubles AS (
        SELECT
            id,
            cost * 7.55 AS cost_rubles
        FROM
            ingredients
    ),
    dishes_costs AS (
        SELECT
            d.dish_id AS id,
            SUM(i.cost_rubles) AS cost
        FROM
            dishes_ingredients d
            JOIN ingredients_costs_in_rubles i ON d.ingredient_id = i.id
        GROUP BY
            d.dish_id
    ),
    menus_costs_and_dishes AS (
        SELECT
            menu_id,
            m.date,
            SUM(cost) AS our_cost,
            ARRAY_AGG(dish_id) AS dishes
        FROM
            dishes_menus m
            JOIN dishes_costs c ON m.dish_id = c.id
        GROUP BY
            menu_id,
            m.date
        ORDER BY
            m.date,
            menu_id
    ),
    menus_profits AS (
        SELECT
            mc.*,
            m.cost,
            ROUND((m.cost - mc.our_cost)::DECIMAL, 2) AS profit,
            ROUND((m.cost - mc.our_cost)::DECIMAL / mc.our_cost::DECIMAL
                * 100, 2) AS profit_percent
        FROM
            menus_costs_and_dishes mc
            JOIN menus m ON mc.menu_id = m.id
    ),
    max_profit AS (
        SELECT
            menu_id,
            date,
            dishes,
            profit,
            profit_percent
        FROM
```

```
            menus_profits
        ORDER BY
            profit_percent DESC
        LIMIT 1
    ),
    max_profit_percent AS (
        SELECT
            menu_id,
            date,
            dishes,
            profit,
            profit_percent
        FROM
            menus_profits
        ORDER BY
            profit DESC
        LIMIT 1
    )

SELECT * FROM max_profit UNION ALL SELECT * FROM max_profit_percent;
```

	menu_id	date	dishes	profit	profit_percent
0	1	2024-12-06	[12, 56, 103]	531.48	315.39
1	2	2024-12-08	[81, 56, 56, 65]	536.83	171.41

Рис. 26: Результат выполнения запроса 7

Листинг 43: Запрос 8 на SQL

```
WITH
    max_count_pair AS (
        SELECT DISTINCT
            ARRAY[dm1.dish_id, dm2.dish_id] AS pair,
            COUNT(*) AS count
        FROM
            dishes_menus dm1 JOIN dishes_menus dm2 USING(menu_id, date)
        WHERE
            dm1.dish_id < dm2.dish_id
        GROUP BY
            pair
        ORDER BY
            count DESC
        LIMIT 1
    )

SELECT
    d1.title AS dish1,
    d2.title AS dish2,
    count
FROM
```

```
    max_count_pair
    JOIN dishes d1 ON pair[1] = d1.id
    JOIN dishes d2 ON pair[2] = d2.id
```

	dish1	dish2	count
0	Сельдь под шубой	Кекс	3

Рис. 27: Результат выполнения запроса 8

Листинг 44: Запрос 9 на SQL

```
WITH
    main_table AS (
        SELECT
            u.id,
            COUNT(o.id) AS orders_count
        FROM
            users u
            JOIN orders o ON u.id = o.user_id
        GROUP BY
            u.id
        ORDER BY
            orders_count ASC
    ),
    series AS (
        SELECT
            orders_count,
            ROW_NUMBER() OVER (
                ORDER BY
                    orders_count
            ) AS row_number,
            COUNT(*) OVER () AS total_rows
        FROM
            main_table
    )

SELECT
    AVG(orders_count) AS median_count
FROM
    series
WHERE
    row_number BETWEEN total_rows / 2.0 AND total_rows / 2.0 + 1
```

	median_count
0	3.0

Рис. 28: Результат выполнения запроса 9

Листинг 45: Запрос 10 на SQL

```
WITH
    main_table AS (
        SELECT
            DATE (o.created_at) AS date,
            SUM(m.cost) AS revenue
        FROM
            orders o
            JOIN menus m ON o.menu_id = m.id
        GROUP BY
            DATE (o.created_at)
        ORDER BY
            date
    )

SELECT
    date,
    revenue,
    (revenue - LAG (revenue, 1) OVER ()) AS revenue_growth_abs,
    ROUND(100 * (revenue - LAG (revenue, 1) OVER ())::DECIMAL / LAG (
        revenue, 1) OVER ()::DECIMAL, 2) AS revenue_growth_rate
FROM
    main_table
```

	date	revenue	revenue_growth_abs	revenue_growth_rate
0	2024-08-19	108100.0	NaN	NaN
1	2024-08-20	423050.0	314950.0	291.35
2	2024-08-21	432450.0	9400.0	2.22
3	2024-08-22	405150.0	-27300.0	-6.31
4	2024-08-23	430900.0	25750.0	6.36
...	...	...	...	...
96	2024-11-23	474850.0	50600.0	11.93
97	2024-11-24	403450.0	-71400.0	-15.04
98	2024-11-25	417250.0	13800.0	3.42
99	2024-11-26	398900.0	-18350.0	-4.40
100	2024-11-27	299050.0	-99850.0	-25.03

101 rows × 4 columns

Рис. 29: Результат выполнения запроса 10

Листинг 46: Запрос 11 на SQL

```
WITH RECURSIVE
    users_tree AS (
        SELECT
            id,
            CONCAT (first_name, ' ', middle_name, ' ', last_name) AS
```

```
            full_name,
            invited_by_id AS parent_id,
            1 AS level
        FROM
            users
        WHERE
            invited_by_id IS NULL
        UNION ALL
        SELECT
            u.id,
            CONCAT(first_name, ' ', middle_name, ' ', last_name) AS
            full_name,
            invited_by_id AS parent_id,
            ut.level + 1
        FROM
            users u
            JOIN users_tree ut ON u.invited_by_id = ut.id
    )

SELECT
    *
FROM
    users_tree
```

	id	full_name	parent_id	level
0	14	Демьян Ильич Романов	NaN	1
1	47	Панфил Валерьянович Назаров	NaN	1
2	1	Фёкла Владиславовна Мартынова	NaN	1
3	2	Раиса Кузьминична Игнатова	NaN	1
4	4	Евпраксия Мироновна Гуляева	NaN	1
...	...	...	...	...
14995	228	Таисия Тарасовна Новикова	13314.0	6
14996	6681	Лариса Робертовна Афанасьева	6977.0	7
14997	2039	Мариан Дмитриевич Анисимов	10803.0	7
14998	5621	Леонтий Ярославович Никонов	524.0	7
14999	13393	Клавдия Геннадиевна Уварова	10741.0	7

15000 rows × 4 columns

Рис. 30: Результат выполнения запроса 11

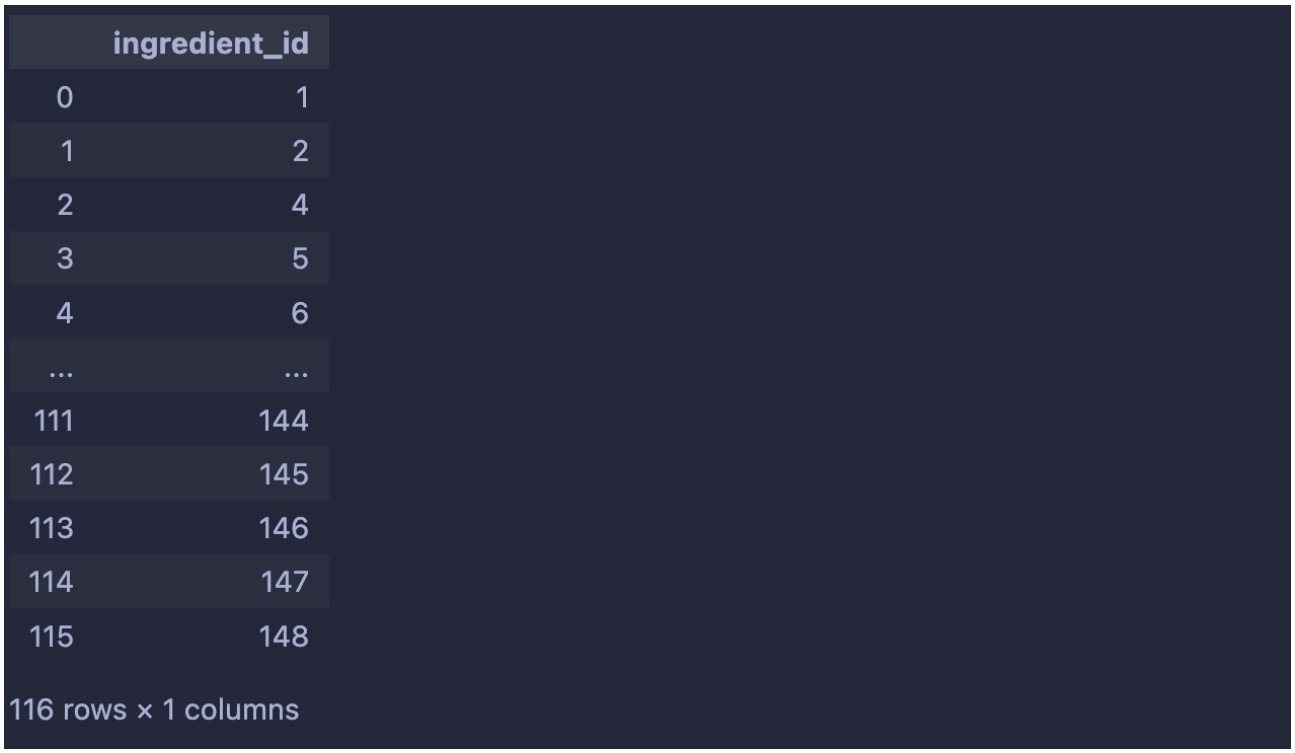
Листинг 47: Запрос 12 на SQL

```
WITH
    orders_last_month AS (
        SELECT
            menu_id,
            date(created_at) AS date,
```

```
            COUNT(*) AS orders_count
        FROM
            orders
        WHERE
            date (created_at) + INTERVAL '1 day' BETWEEN CURRENT_DATE -
               INTERVAL '1 month' AND CURRENT_DATE
        GROUP BY
            date (created_at),
            menu_id
    ),
    dishes_count AS (
        SELECT
            dish_id,
            SUM(orders_count) AS count
        FROM
            orders_last_month o
            JOIN dishes_menus dm ON o.menu_id = dm.menu_id
            AND o.date + INTERVAL '1 month 1 day' = dm.date
        GROUP BY
            dish_id
    ),
    dishes_ingredients_with_weights AS (
        SELECT
            dish_id,
            ingredient_id,
            weight::DECIMAL / COUNT(*) OVER (PARTITION BY dish_id)::
               DECIMAL AS ingredients_weight
        FROM
            dishes_ingredients AS di
            JOIN dishes AS d ON di.dish_id = d.id
    ),
    ingredients_total_weight AS (
        SELECT
            ingredient_id,
            SUM(ingredients_weight * count) AS total_weight
        FROM
            dishes_ingredients_with_weights AS weights
            JOIN dishes_count AS counts USING (dish_id)
        GROUP BY
            ingredient_id
    ),
    ingredients_productivity AS (
        SELECT
            ingredient_id,
            SUM(productivity) AS productivity
        FROM
            ingredients_suppliers AS ins
            JOIN suppliers s ON ins.supplier_id = s.id
        GROUP BY
            ingredient_id
    )

SELECT
```

```
    ingredient_id
FROM
    ingredients_total_weight AS itw
    JOIN ingredients_productivity AS ip USING (ingredient_id)
WHERE
    productivity * 1000 >= total_weight
ORDER BY
    ingredient_id
```



	ingredient_id
0	1
1	2
2	4
3	5
4	6
...	...
111	144
112	145
113	146
114	147
115	148

116 rows × 1 columns

Рис. 31: Результат выполнения запроса 12

6.2 Отчеты

6.2.1 Отчет №1

Название: Анализ выручки сервиса за предыдущие 30 дней

Дата создания отчета: 25.05.2025

Инструментальные средства: *Python, Matplotlib, Seaborn*

Диаграмма:

Рис. 32: Диаграмма к отчету №1

Код запроса:

Листинг 48: Код запроса к отчету №1

```
WITH
    main_table AS (
        SELECT
            DATE (o.created_at) AS date,
            SUM(m.cost) AS revenue
        FROM
            orders o
            JOIN menus m ON o.menu_id = m.id
        GROUP BY
            DATE (o.created_at)
        ORDER BY
            date
    )

SELECT
    date,
    revenue,
    (revenue - LAG (revenue, 1) OVER ()) AS revenue_growth_abs,
    ROUND(100 * (revenue - LAG (revenue, 1) OVER ())::DECIMAL / LAG (
        revenue, 1) OVER ()::DECIMAL, 2) AS revenue_growth_rate
FROM
    main_table
```

6.2.2 Отчет №2

Название: Анализ распределения количества бонусов среди пользователей

Дата создания отчета: 25.05.2025

Инструментальные средства: *Python, Pandas, Matplotlib, Seaborn*

Диаграмма:

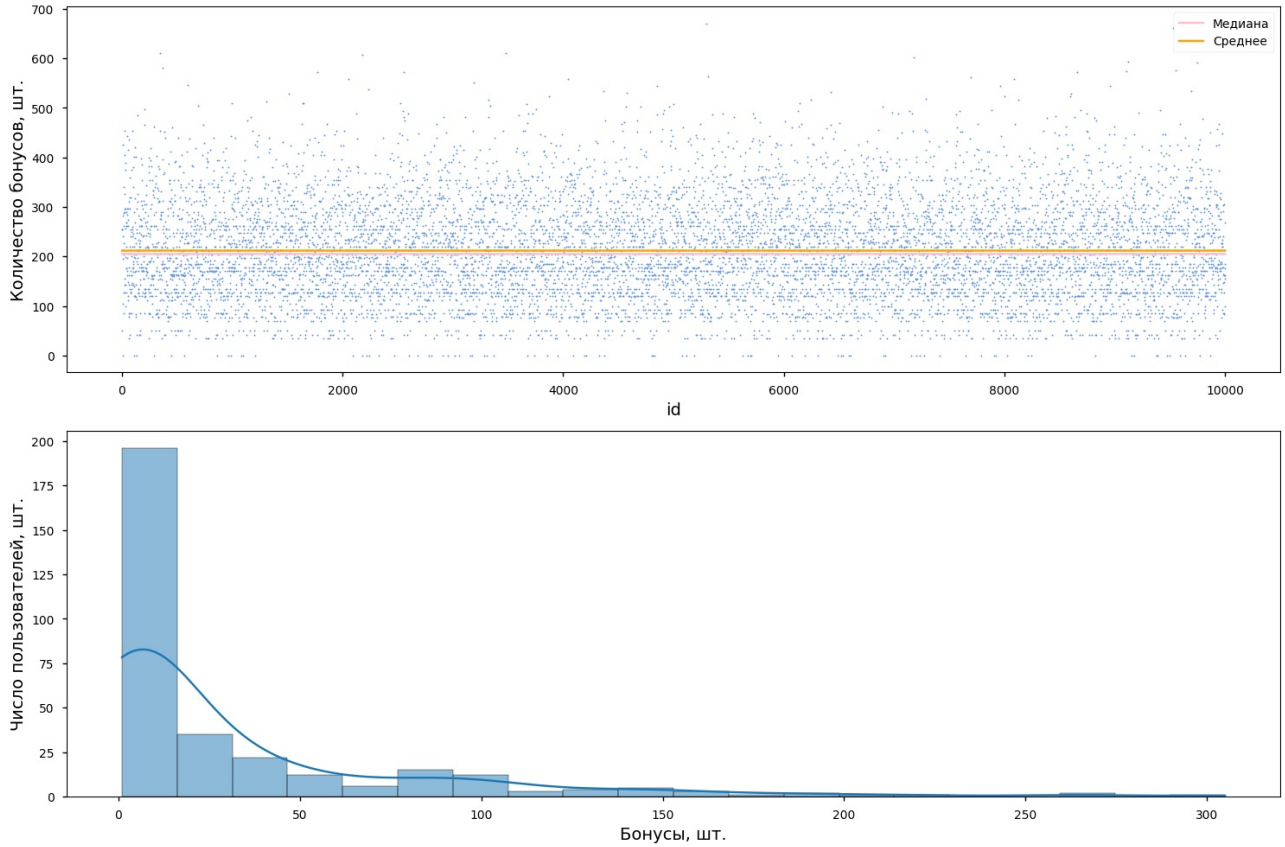


Рис. 33: Диаграмма к отчету №2

Коды запроса:

Листинг 49: Код запроса к отчету №2

```
SELECT
    id,
    bonuses
FROM
    users
ORDER BY
    id ASC
```

Листинг 50: Код запроса к отчету №2

```sql
SELECT
    bonuses,
    COUNT(*) AS count
FROM
    users
GROUP BY
    bonuses
ORDER BY
    bonuses ASC;
```

6.2.3 Отчет №3

Название: Анализ частоты встречаемости различных пар блюд в меню

Дата создания отчета: 25.05.2025

Инструментальные средства: *Python, Pandas, Matplotlib, Seaborn*

Диаграмма:

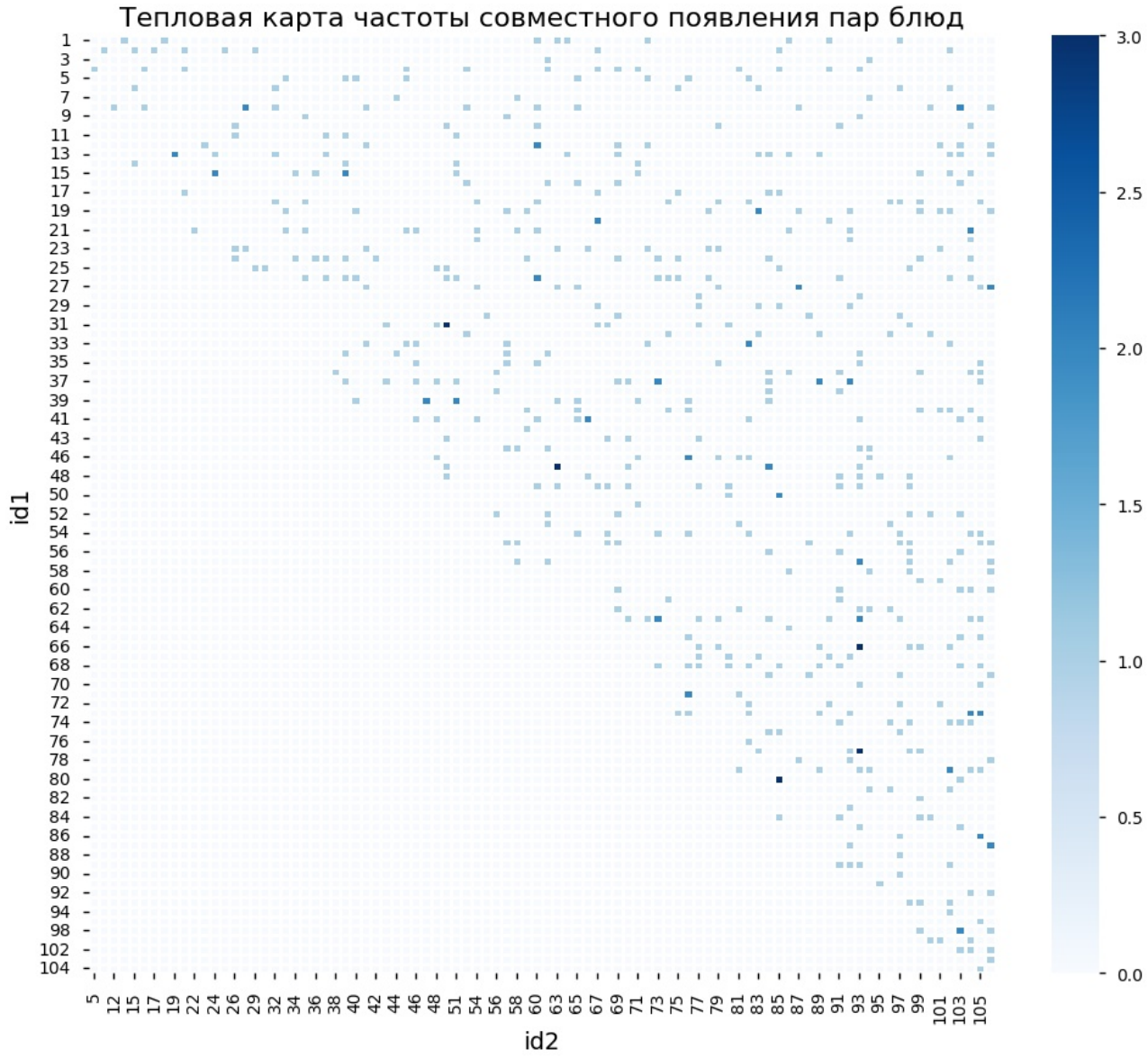


Рис. 34: Диаграмма к отчету №2

Код запроса:

Листинг 51: Код запроса к отчету №3

```
WITH
    max_count_pair AS (
        SELECT DISTINCT
            ARRAY[dm1.dish_id, dm2.dish_id] AS pair,
            COUNT(*) AS count
        FROM
            dishes_menus dm1 JOIN dishes_menus dm2 USING(menu_id, date)
        WHERE
            dm1.dish_id < dm2.dish_id
        GROUP BY
            pair
    )

SELECT
    d1.id AS id1,
    d2.id AS id2,
    count
FROM
    max_count_pair
    JOIN dishes d1 ON pair[1] = d1.id
    JOIN dishes d2 ON pair[2] = d2.id
```