Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Курсовая работа по дисциплине

«Базы данных»

Тема: «Разработка информационной системы сети ресторанов Sushi Shop»

Выполнил: студент группы РИС-23-1б

Гордеев Василий Андреевич

Проверил: доцент кафедры ИТАС

Петренко Александр Анатольевич

г. Пермь – 2025

РЕФЕРАТ

Отчет 65 страниц, 3 рисунка, 1 таблица, 7 источников, 9 приложений.

БАЗА ДАННЫХ, SUSHI SHOP, ВЕБ-ПРИЛОЖЕНИЕ, POSTGRESQL, СУБД, КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ, PYTHON, HTML

Объект исследования: Процесс управления данными в сети кафе-ресторанов.

Предмет исследования: Методы и средства проектирования, реализации и эксплуатации реляционной базы данных для задач учета и анализа в ресторанном бизнесе.

Цель работы: Разработка информационной системы для автоматизации учета продуктов, расчета себестоимости и маржинальности блюд на примере сети ресторанов "Sushi-shop".

Задачи:

* Проанализировать предметную область и сформулировать требования к базе данных.
* Разработать концептуальную, логическую и физическую модели данных.
* Обосновать выбор СУБД.
* Реализовать базу данных в выбранной СУБД (таблицы, связи, представления, права доступа).
* Разработать прототип веб-приложения для взаимодействия с БД и провести его тестирование

Итогом работы стала разработанная база данных и веб-приложение для работы с ней.

Практическая значимость работы заключается в создании прототипа базы данных, который может служить основой для разработки полнофункциональной информационной системы управления ресторанами.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc198092224)

[Раздел 1. Анализ предметной области 6](#_Toc198092225)

[1.1 Определение границ исследования 6](#_Toc198092226)

[1.2 Анализ существующих подходов к автоматизации учета 7](#_Toc198092227)

[1.3 Требования к разрабатываемой базе данных 8](#_Toc198092228)

[1.3.1 Функциональные требования к базе данных 8](#_Toc198092229)

[1.3.2 Нефункциональные требования к базе данных 10](#_Toc198092230)

[1.3.3 Требования к данным 10](#_Toc198092231)

[Раздел 2. Проектирование базы данных 12](#_Toc198092232)

[2.1 Разработка концептуальной модели (ER-диаграмма) 12](#_Toc198092233)

[2.2 Разработка логической модели 13](#_Toc198092234)

[2.3 Выбор системы управления базами данных (СУБД) 16](#_Toc198092235)

[2.4 Разработка физической модели 18](#_Toc198092236)

[2.5 Проведение нормализации 20](#_Toc198092237)

[Раздел 3. Технология реализации и проведение эксперимента 22](#_Toc198092238)

[3.1. Инструментарий и среда разработки 23](#_Toc198092239)

[3.2. Реализация структуры базы данных и бизнес-логики 23](#_Toc198092240)

[3.3. Разработка прототипа веб-приложения 25](#_Toc198092241)

[3.4. Тестирование системы 26](#_Toc198092242)

[ЗАКЛЮЧЕНИЕ 27](#_Toc198092243)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 29](#_Toc198092244)

ВВЕДЕНИЕ

Актуальность автоматизации бизнес-процессов в сфере общественного питания неоспорима. В условиях высокой конкуренции и постоянно растущих требований к качеству обслуживания, эффективное управление ресурсами становится ключевым фактором успеха. Особое значение приобретает точный учет продуктов на складах, калькуляция себестоимости блюд и анализ маржинальности, поскольку эти показатели напрямую влияют на рентабельность заведения. Использование баз данных (БД) предоставляет мощный инструментарий для решения этих задач, обеспечивая структурированное хранение информации, быстрый доступ к ней и возможности для комплексного анализа.

Постановка задачи данной курсовой работы заключается в разработке базы данных для информационной поддержки основных процессов учета и анализа в сети кафе-ресторанов "Sushi-shop". Основной акцент будет сделан на управлении складскими запасами, учете состава блюд, расчете их себестоимости и последующем анализе маржинальности.

Целью работы является создание эффективной и надежной реляционной базы данных, обеспечивающей автоматизацию учета продуктов на складах, точный расчет себестоимости блюд и их маржинальности для сети ресторанов "Sushi-shop".

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ предметной области и сформулировать детальные требования к разрабатываемой базе данных.
2. Разработать концептуальную, логическую и физическую модели данных, отражающие структуру информации в предметной области.
3. Обосновать выбор системы управления базами данных (СУБД) для реализации проекта.
4. Реализовать спроектированную базу данных в выбранной СУБД, включая создание таблиц, связей, представлений (view), а также настройку прав доступа для различных категорий пользователей.
5. Разработать и протестировать набор SQL-запросов для извлечения необходимой информации и формирования отчетов.

Объектом исследования является процесс управления данными в сети кафе-ресторанов.

Предметом исследования выступают методы и средства проектирования, реализации и эксплуатации реляционной базы данных для решения задач учета и анализа в указанной предметной области.

Практическая значимость работы заключается в создании прототипа базы данных, который может быть использован как основа для дальнейшей разработки полнофункциональной информационной системы управления сетью ресторанов "Sushi-shop". Разработанные модели и SQL-запросы могут служить методическим материалом при решении аналогичных задач.

# Раздел 1. Анализ предметной области

Для успешного проектирования и реализации эффективной базы данных необходимо провести тщательный анализ предметной области, определить границы исследования и сформулировать четкие требования к будущей системе.

## 1.1 Определение границ исследования

Предметом исследования данной курсовой работы является разработка базы данных для сети кафе-ресторанов "Sushi-shop". Информационная система, основанная на этой БД, призвана автоматизировать и оптимизировать ключевые процессы, связанные с учетом продуктов, приготовлением блюд и анализом экономической эффективности.

Границы исследования охватывают следующие аспекты деятельности сети "Sushi-shop":

1. Учет продуктов на складах: поступление продуктов от поставщиков, хранение информации о продуктах (наименование, единицы измерения, сроки годности и т.д.), внутреннее перемещение продуктов между складами (если предполагается несколько складов в рамках одной точки или между точками), списание продуктов (по причине порчи, использования в приготовлении блюд).
2. Управление рецептурами (технологическими картами): хранение информации о составе каждого блюда, включая перечень ингредиентов и их точное количество, необходимое для приготовления одной порции.
3. Учет продаж и заказов: фиксация данных о проданных блюдах для последующего расчета расхода продуктов и анализа популярности позиций меню.
4. Расчет себестоимости блюд: автоматический расчет себестоимости каждого блюда на основе актуальных закупочных цен на ингредиенты и их количества согласно рецептуре.
5. Анализ маржинальности: расчет и анализ разницы между ценой продажи блюда и его себестоимостью.
6. Управление сотрудниками в контексте доступа к данным: фиксация информации о сотрудниках для разграничения прав доступа к различным функциям и данным в системе.
7. За рамками данного исследования остаются такие аспекты деятельности ресторанного бизнеса, как управление персоналом (графики работы, начисление заработной платы, кроме как в контексте должности для определения ставки и доступа), маркетинговые кампании, программы лояльности, финансовый учет предприятия в целом, бухгалтерский учет, а также разработка полнофункционального пользовательского интерфейса приложения. Основной упор делается на структуру и логику самой базы данных.

## 1.2 Анализ существующих подходов к автоматизации учета

Автоматизация учета в ресторанном бизнесе является распространенной практикой, и на рынке существует множество программных решений, предлагающих различные функциональные возможности. Среди них можно выделить несколько основных категорий:

* POS-системы (Point of Sale): Основная функция – автоматизация процесса продаж, прием заказов, печать чеков. Многие современные POS-системы также включают базовые функции складского учета, списывая ингредиенты согласно проданным блюдам.
* Системы управления складом (Inventory Management Systems): Специализированные решения для детального учета товарно-материальных ценностей, контроля остатков, управления закупками и инвентаризациями.
* ERP-системы (Enterprise Resource Planning) для ресторанов: Комплексные системы, охватывающие большинство бизнес-процессов предприятия, включая финансы, закупки, склад, производство, продажи и управление персоналом. Примерами могут служить такие системы, как iiko или r\_keeper, которые предоставляют широкие возможности по управлению ресторанным бизнесом, включая сложные схемы учета и анализа[1][2].

Анализ существующих решений показывает, что ключевым элементом любой такой системы является хорошо структурированная база данных. Она позволяет хранить информацию о продуктах, рецептах, продажах, поставщиках и т.д. Однако, готовые решения могут быть избыточны по функционалу для небольшой сети или стартапа, требовать значительных финансовых вложений или не обладать достаточной гибкостью для адаптации под уникальные бизнес-процессы. Разработка собственной базы данных, сфокусированной на конкретных потребностях "Sushi-shop", позволит создать более легковесное, адаптированное и потенциально менее затратное решение для ключевых учетных задач.

## 1.3 Требования к разрабатываемой базе данных

На основе анализа предметной области и существующих подходов можно сформулировать следующие требования к разрабатываемой базе данных для сети "Sushi-shop".

1.3.1 Функциональные требования к базе данных

База данных должна обеспечивать хранение и обработку информации для выполнения следующих функций:

Учет продуктов:

* Хранение информации о продуктах (наименование, единица измерения, категория, описание, поставщик по умолчанию (опционально)).
* Учет поступления продуктов на склад (дата, количество, закупочная цена).
* Учет остатков продуктов на складах (с возможностью ведения учета по нескольким складам).
* Учет списания продуктов (по различным причинам: приготовление блюд, порча).

Управление блюдами и рецептурами:

* Хранение информации о блюдах (наименование, категория, цена продажи, описание, фотография (ссылка)).
* Хранение рецептур (технологических карт) для каждого блюда с указанием ингредиентов и их количества.

Учет продаж:

* Фиксация информации о проданных блюдах (дата, время, количество, блюдо, итоговая сумма по позиции).
* Расчеты и аналитика:
* Возможность расчета себестоимости каждого блюда на основе актуальных цен ингредиентов.
* Возможность расчета маржинальности по каждому блюду и по группам блюд.
* Формирование отчетов об остатках продуктов, движении товаров, продажах.

Управление сотрудниками и доступом:

* Хранение информации о сотрудниках (ФИО, должность).
* Реализация механизма разграничения прав доступа к данным на основе ролей (например, администратор, управляющий, повар).

Справочная информация:

* Хранение справочников (категории продуктов, категории блюд, единицы измерения, должности, причины списания и т.д.).

### 1.3.2 Нефункциональные требования к базе данных

Надежность: Обеспечение сохранности данных при сбоях. Использование механизмов транзакций для атомарности операций.

Целостность данных: Поддержание логической согласованности и достоверности данных с помощью первичных и внешних ключей, ограничений (constraints).

Производительность: Обеспечение приемлемого времени отклика на типовые запросы пользователей.

Безопасность: Защита данных от несанкционированного доступа.

Масштабируемость: Возможность увеличения объема хранимых данных и интенсивности запросов без значительного падения производительности (в разумных пределах для курсового проекта).

Простота администрирования: Относительная простота обслуживания и внесения изменений в структуру БД (в рамках возможностей выбранной СУБД)[5].

### 1.3.3 Требования к данным

Актуальность: Данные, хранящиеся в БД (особенно цены на продукты, остатки), должны отражать текущее состояние дел.

Непротиворечивость: Данные в различных частях БД не должны противоречить друг другу.

Полнота: БД должна содержать все необходимые данные для выполнения заявленных функций.

Точность: Данные должны быть корректными и точно отражать реальные значения (например, количество ингредиентов, цены)[5].

Вывод:

В данной главе был проведен анализ предметной области – деятельности сети кафе-ресторанов "Sushi-shop" – и определены границы исследования для разработки базы данных. Рассмотрены существующие подходы к автоматизации учета в ресторанном бизнесе, что позволило выявить как преимущества, так и потенциальные ограничения готовых решений. На основе этого анализа были сформулированы подробные функциональные и нефункциональные требования к разрабатываемой базе данных, а также требования к самим данным. Эти требования послужат основой для последующих этапов проектирования и реализации базы данных "Sushi-shop".

# Раздел 2. Проектирование базы данных

После проведения анализа предметной области и формулировки требований к базе данных, следующим ключевым этапом является ее проектирование. Процесс проектирования включает разработку концептуальной, логической и физической моделей данных, выбор системы управления базами данных (СУБД) и проведение нормализации для обеспечения целостности и эффективности хранения информации.

## 2.1 Разработка концептуальной модели (ER-диаграмма)

Концептуальное моделирование является первым шагом в проектировании базы данных и направлено на создание высокоуровневого представления предметной области. На этом этапе определяются основные сущности (информационные объекты), их ключевые атрибуты и типы связей между ними, без учета деталей реализации и специфики конкретной СУБД[3].

Для информационной системы сети кафе-ресторанов "Sushi-shop" были выделены следующие основные сущности:

* Сотрудник: Информация о работниках заведения.
* Должность: Справочник должностей и связанных с ними ставок.
* Касса: Информация о кассовых аппаратах.
* Смена: Данные о рабочих сменах сотрудников.
* Склад: Информация о местах хранения продуктов.
* Продукт: Детализированная информация об ингредиентах и товарах.
* Категория Продукта: Классификатор продуктов.
* Единица Измерения: Справочник единиц измерения продуктов.
* Поступление: Документы, фиксирующие приход продуктов на склад.
* Списание: Документы, фиксирующие расход или утилизацию продуктов и блюд.
* Причина Списания: Справочник причин списания.
* Блюдо: Информация о позициях меню.
* Категория Блюда: Классификатор блюд.
* Состав Блюда (Рецептура): Связь между блюдами и продуктами, определяющая ингредиенты.
* Заказ: Информация о заказах клиентов.
* Позиция Заказа: Детализация заказа по конкретным блюдам.
* Остаток Продукта на Складе: Учет текущего количества каждого продукта на каждом складе.

Атрибуты для каждой сущности на концептуальном уровне отражают их основные характеристики. Например, для сущности "Продукт" это могут быть "Наименование", "Категория", "Единица измерения". Для сущности "Заказ" – "Дата заказа", "Сотрудник оформивший", "Общая сумма".

Связи между сущностями отражают бизнес-правила. Например, между "Сотрудник" и "Должность" существует связь "один-ко-многим" (одна должность может быть у многих сотрудников, но каждый сотрудник занимает одну должность). Связь между "Заказ" и "Блюдо" является "многие-ко-многим" и реализуется через промежуточную сущность "Позиция Заказа"[1].

Результатом концептуального моделирования является ER-диаграмма (Entity-Relationship Diagram), которая наглядно представляет структуру данных. ER-диаграмма для разрабатываемой базы данных приведена в Приложении А.

## 2.2 Разработка логической модели

Логическая модель данных является следующим этапом проектирования и представляет собой преобразование концептуальной модели в реляционную структуру, не зависящую от физических аспектов хранения, но уже ориентированную на использование реляционной СУБД. На этом этапе определяются таблицы, их атрибуты, первичные и внешние ключи, а также детально прорабатываются связи между таблицами с указанием их кардинальности и опциональности. Для представления логической модели была использована нотация "вороньи лапы" (Crow's Foot).

В ходе разработки логической модели сущности из концептуальной модели были преобразованы в таблицы, а атрибуты сущностей – в поля таблиц. Связи "многие-ко-многим", такие как между "Заказом" и "Блюдом" или "Блюдом" и "Продуктом" (в контексте рецептуры), были реализованы через создание промежуточных (ассоциативных) таблиц: "ПозицияЗаказа" и "СоставБлюда" соответственно.

Ниже приведено описание основных таблиц логической модели (полная схема представлена в Приложении Б)

Таблица "Сотрудник": Хранит данные о сотрудниках. Первичный ключ – ID\_сотрудника. Внешний ключ ID\_должности ссылается на таблицу "Должность".

Таблица "Должность": Справочник должностей. Первичный ключ – ID\_должности.

Таблица "Продукт": Информация об ингредиентах. Первичный ключ – ID\_продукта. Внешние ключи ID\_категории\_продукта и ID\_единицы\_измерения ссылаются на соответствующие справочники.

Таблица "Блюдо": Позиции меню. Первичный ключ – ID\_блюда. Внешний ключ ID\_категории\_блюда ссылается на справочник категорий блюд.

Таблица "СоставБлюда": Промежуточная таблица для связи "многие-ко-многим" между "Блюдо" и "Продукт". Составной первичный ключ (ID\_блюда, ID\_продукта).

Таблица "Заказ": Информация о заказах. Первичный ключ – ID\_заказа. Внешние ключи ID\_кассы и ID\_сотрудника\_оформившего.

Таблица "ПозицияЗаказа": Промежуточная таблица для связи "многие-ко-многим" между "Заказ" и "Блюдо". Составной первичный ключ (ID\_заказа, ID\_блюда).

Таблица "Поступление": Учет прихода товаров. Первичный ключ – ID\_поступления. Внешние ключи ID\_склада и ID\_сотрудника\_принявшего.

Таблица "ПозицияПоступления": Детализация поступления. Составной первичный ключ (ID\_поступления, ID\_продукта).

Таблица "Списание": Базовая таблица для операций списания. Первичный ключ – ID\_списания.

Таблицы "СписаниеПродуктов" и "СписаниеБлюд": Реализуют механизм наследования от таблицы "Списание", детализируя списание конкретных объектов. Первичный ключ ID\_списания является также внешним ключом, ссылающимся на таблицу "Списание".

Таблица "ОстатокПродуктаНаСкладе": Учет текущих остатков. Составной первичный ключ (ID\_склада, ID\_продукта).

Связи между таблицами определены с указанием кардинальности. Например, связь между "Сотрудник" и "Смена" – один ко многим (один сотрудник может иметь много смен, каждая смена принадлежит одному сотруднику), обязательная со стороны "Смены". Связь между "Заказ" и "ПозицияЗаказа" – один ко многим, где один заказ должен содержать как минимум одну позицию, и каждая позиция принадлежит ровно одному заказу.

Логическая модель (Приложение Б) обеспечивает структурированное представление данных, минимизирует избыточность и создает основу для физической реализации базы данных.

## 2.3 Выбор системы управления базами данных (СУБД)

Выбор подходящей СУБД является важным шагом, влияющим на производительность, надежность, масштабируемость и стоимость разработки и эксплуатации информационной системы. Для проекта "Sushi-shop" рассматривались реляционные СУБД, так как структура данных предметной области хорошо ложится на реляционную модель.

Основные кандидаты для выбора и их сравнительная характеристика по ключевым для проекта критериям представлены в Таблице 2.1.

Таблица 2.1 – Сравнительная характеристика реляционных СУБД

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | PostgreSQL | MySQL | MS SQL Server Express Edition |
| Тип лицензии | Открытое ПО (лицензия PostgreSQL, аналогична MIT) | Открытое ПО (GPL) / Коммерческая | Бесплатная версия коммерческого ПО |
| Поддержка стандартов SQL | Высокая, одно из лучших соответствий | Хорошая, с некоторыми особенностями | Хорошая, с T-SQL как диалектом |
| Объектно-реляционные возможности | Расширенные (наследование таблиц, пользовательские типы данных и операторы) | Ограниченные (в основном реляционная) | Присутствуют, но менее выражены, чем в PostgreSQL |
| Сложные запросы и аналитика | Очень хорошо подходит, мощный оптимизатор | Хорошо, но может уступать PostgreSQL на очень сложных запросах | Хорошо, особенно с инструментами аналитики Microsoft (в платных версиях) |
| Масштабируемость | Высокая (вертикальная и горизонтальная) | Хорошая, особенно для операций чтения | Ограничена в Express Edition (по ресурсам) |
| Надежность и стабильность | Высокая, проверена временем | Высокая (особенно с InnoDB) | Высокая |
| Сообщество и поддержка | Активное и большое международное сообщество | Очень большое сообщество, обширная документация | Хорошая поддержка от Microsoft, активное сообщество |
| Простота использования и администрирования | Средняя, требует определенных знаний | Относительно простая, особенно для стандартных задач | Относительно простая, особенно с GUI-инструментами от Microsoft |
| Специфичные функции для проекта (наследование таблиц) | Полная поддержка | Отсутствует на уровне таблиц (можно эмулировать) | Отсутствует на уровне таблиц (можно эмулировать) |
| Стоимость | Бесплатно | Бесплатно (Community Edition) / Платно (Enterprise Edition) | Бесплатно (Express Edition) / Платно (Standard, Enterprise) |

Анализ сравнительной таблицы:

Из таблицы видно, что все три рассматриваемые СУБД являются зрелыми и надежными решениями. Однако, для проекта "Sushi-shop" PostgreSQL выделяется по нескольким ключевым параметрам:

Объектно-реляционные возможности и наследование таблиц: PostgreSQL нативно поддерживает наследование таблиц, что является важным аспектом для реализации механизма учета списаний (таблицы "СписаниеПродуктов" и "СписаниеБлюд" наследуют от "Списание"), как это было запланировано на этапе логического проектирования. В MySQL и MS SQL Server подобную функциональность пришлось бы эмулировать, что усложнило бы структуру и запросы.

Поддержка сложных запросов и аналитики: Для задач расчета себестоимости, маржинальности и формирования различных аналитических отчетов PostgreSQL предоставляет мощный инструментарий и эффективный оптимизатор запросов.

Открытый исходный код и отсутствие лицензионных ограничений: Это делает PostgreSQL привлекательным выбором с точки зрения затрат и гибкости использования.

Высокая степень соответствия стандартам SQL: Обеспечивает лучшую переносимость знаний и кода.

MySQL, будучи очень популярной и быстрой для многих веб-приложений, уступает PostgreSQL в поддержке расширенных реляционных и объектных функций, которые могут быть полезны для данного проекта. MS SQL Server Express Edition, хотя и является качественным продуктом, имеет ограничения по ресурсам (размер базы данных, использование процессора и памяти), что может стать препятствием при росте системы "Sushi-shop".

Таким образом, для реализации базы данных информационной системы "Sushi-shop" была выбрана СУБД PostgreSQL. Данный выбор обусловлен ее богатой функциональностью, включая поддержку наследования таблиц, высокой надежностью, соответствием стандартам SQL, отсутствием лицензионных затрат и хорошей масштабируемостью, что наилучшим образом соответствует требованиям проекта.

## 2.4 Разработка физической модели

Физическая модель данных описывает конкретную реализацию логической модели в выбранной СУБД, в данном случае – PostgreSQL. На этом этапе определяются точные типы данных для каждого поля, создаются индексы для оптимизации запросов, определяются правила ссылочной целостности и другие специфичные для СУБД параметры.

Структура таблиц, их поля и первичные/внешние ключи в физической модели соответствуют ранее разработанной логической модели. Ниже приведено описание ключевых аспектов физической реализации для некоторых таблиц с указанием типов данных PostgreSQL:

Таблица "Сотрудник":

* ID\_сотрудника: SERIAL PRIMARY KEY (автоинкрементный целочисленный идентификатор).
* ID\_должности: INTEGER NOT NULL REFERENCES "Должность"(ID\_должности).
* Фамилия, Имя, Отчество: VARCHAR(100).
* Телефон: VARCHAR(20) UNIQUE.
* Дата\_приема: DATE NOT NULL DEFAULT CURRENT\_DATE.
* Активен: BOOLEAN NOT NULL DEFAULT TRUE.

Таблица "Продукт":

* ID\_продукта: SERIAL PRIMARY KEY.
* ID\_категории\_продукта: INTEGER NOT NULL REFERENCES "КатегорияПродукта"(ID\_категории\_продукта).
* ID\_единицы\_измерения: INTEGER NOT NULL REFERENCES "ЕдиницаИзмерения"(ID\_единицы\_измерения).
* Название: VARCHAR(255) NOT NULL UNIQUE.
* Срок\_годности\_дни: INTEGER CHECK("Срок\_годности\_дни" > 0).

Таблица "ПозицияЗаказа":

* ID\_заказа: INTEGER NOT NULL REFERENCES "Заказ"(ID\_заказа) ON DELETE CASCADE.
* ID\_блюда: INTEGER NOT NULL REFERENCES "Блюдо"(ID\_блюда).
* Количество: INTEGER NOT NULL CHECK("Количество" > 0).
* Цена\_на\_момент\_заказа: NUMERIC(10, 2) NOT NULL.
* PRIMARY KEY (ID\_заказа, ID\_блюда).

Для обеспечения ссылочной целостности активно используются внешние ключи с указанием соответствующих действий (ON DELETE CASCADE  в таблицах "ПозицияЗаказа", "СоставБлюда", "ОстатокПродуктаНаСкладе" для автоматического удаления связанных записей при удалении основной). Ограничения NOT NULL, UNIQUE и CHECK применяются для гарантии корректности и непротиворечивости данных.

Для повышения производительности запросов создаются **индексы**. Помимо индексов, автоматически создаваемых для первичных и уникальных ключей, дополнительные индексы создаются для всех внешних ключей, а также для полей, часто используемых в условиях выборки (WHERE) или сортировки (ORDER BY), например, для полей с датами (Дата\_создания в таблице "Заказ", Дата\_поступления в таблице "Поступление").

Физическая модель также включает определение таких объектов базы данных, как пользователи (роли), представления (views), триггеры и хранимые процедуры. Их назначение и логика работы будут рассмотрены в Разделе 3, а полный SQL-код для их создания представлен в Приложениях Г-З

## 2.5 Проведение нормализации

Нормализация – это процесс организации данных в базе данных с целью уменьшения избыточности и улучшения целостности данных. Он включает применение набора правил, называемых нормальными формами. Для разрабатываемой базы данных "Sushi-shop" был проведен анализ на соответствие основным нормальным формам.

Первая нормальная форма (1НФ): Требует, чтобы все атрибуты таблиц были атомарными, то есть не содержали повторяющихся групп или составных значений в одной ячейке. Все таблицы в спроектированной базе данных соответствуют этому требованию, так как каждое поле хранит только одно значение.

Вторая нормальная форма (2НФ): Требует, чтобы таблица находилась в 1НФ и все неключевые атрибуты полностью функционально зависели от всего первичного ключа. Это требование особенно актуально для таблиц с составными первичными ключами.

Например, в таблице "ПозицияЗаказа" первичный ключ составной (ID\_заказа, ID\_блюда). Атрибуты Количество и Цена\_на\_момент\_заказа полностью зависят от комбинации конкретного заказа и конкретного блюда в этом заказе, а не от какой-либо части ключа по отдельности. Аналогично для таблиц "СоставБлюда","ПозицияПоступления", "ОстатокПродуктаНаСкладе".

Третья нормальная форма (3НФ): Требует, чтобы таблица находилась в 2НФ и не содержала транзитивных зависимостей неключевых атрибутов от первичного ключа (то есть, неключевой атрибут не должен зависеть от другого неключевого атрибута).

Например, в таблице "Сотрудник" атрибут Ставка\_в\_час не хранится напрямую, так как он зависит от должности сотрудника. Вместо этого, ID\_должности является внешним ключом, а сама ставка хранится в таблице "Должность". Это предотвращает транзитивную зависимость (Сотрудник -> Должность -> Ставка).

Аналогично, наименования категорий, единиц измерения вынесены в отдельные справочные таблицы ("КатегорияПродукта", "ЕдиницаИзмерения", "КатегорияБлюда", "ПричинаСписания"), и в основных таблицах используются только их идентификаторы, что устраняет транзитивные зависимости и избыточность.

Нормальная форма Бойса-Кодда (БКНФ): Является усиленной версией 3НФ. В большинстве случаев, если база данных находится в 3НФ и каждый детерминант (атрибут или набор атрибутов, однозначно определяющий другие атрибуты) является потенциальным ключом, то она находится и в БКНФ. Спроектированная структура стремится к соответствию БКНФ[4].

Вывод:

В рамках данного раздела была проделана работа по проектированию базы данных для информационной системы "Sushi-shop". Разработана концептуальная модель, отражающая основные сущности и связи предметной области. На ее основе создана логическая модель в нотации "вороньи лапы", детализирующая структуру таблиц, атрибутов и ключей. Был обоснован выбор СУБД PostgreSQL для реализации проекта. Далее была разработана физическая модель, описывающая реализацию таблиц с указанием типов данных PostgreSQL, ограничений и индексов. Проведенный анализ нормализации подтвердил, что структура базы данных соответствует как минимум третьей нормальной форме, что обеспечивает минимизацию избыточности и поддержку целостности данных. Созданная таким образом структура базы данных является надежной основой для последующей реализации и эксплуатации информационной системы.

# Раздел 3. Технология реализации и проведение эксперимента

Данный раздел посвящен описанию технологического стека, использованного для воплощения спроектированной базы данных и прототипа информационной системы "Sushi-shop", а также результатам проведенного тестирования.

## 3.1. Инструментарий и среда разработки

Для реализации проекта были выбраны современные и широко распространенные технологии. Основой для хранения данных послужила объектно-реляционная система управления базами данных PostgreSQL (версия 14), выбранная за ее надежность, соответствие стандартам SQL, расширяемость и поддержку сложных структур данных, таких как наследование таблиц, что было использовано при моделировании операций списания[7].

Взаимодействие с СУБД, проектирование схемы, написание и отладка SQL-скриптов (для создания таблиц, представлений, функций, триггеров и управления правами доступа) осуществлялись с помощью интегрированной среды pgAdmin 4.

Прототип веб-приложения, обеспечивающего пользовательский интерфейс для взаимодействия с базой данных, был разработан на языке программирования  Python (версия 3.9+)  с использованием микрофреймворка Flask (версия 2.x+). Flask был выбран за его легковесность, гибкость и простоту освоения, что оптимально для создания прототипов. Связь между Python-приложением и базой данных PostgreSQL обеспечивалась через адаптер psycopg2. Разработка велась в среде Visual Studio Code.

## 3.2. Реализация структуры базы данных и бизнес-логики

Процесс реализации базы данных включал несколько ключевых этапов, выполненных посредством SQL-скриптов (полные листинги приведены в Приложениях X, Y, Z – *здесь и далее укажите реальные номера приложений*).

Создание таблиц и связей: На основе разработанной физической модели (Приложение В) были созданы все необходимые таблицы, определены типы данных для атрибутов, первичные и внешние ключи для обеспечения ссылочной целостности, а также ограничения (NOT NULL, UNIQUE, CHECK) для поддержания корректности данных. Индексы были созданы для всех первичных и внешних ключей, а также для часто используемых в выборках полей с целью оптимизации производительности запросов[7].

Реализация бизнес-логики на уровне БД:

Представления (Views): Создан набор представлений (Приложение Г) для агрегации данных и упрощения формирования отчетов (например, VТекущиеОстатки, VСписокЗаказов, VСводкаПродажПоБлюдам).

Хранимые функции: Разработаны функции (Приложение Д) для инкапсуляции часто используемых операций и бизнес-правил, таких как создание нового заказа (СоздатьНовыйЗаказ), добавление позиции в заказ (ДобавитьПозициюВЗаказ), создание документов списания продуктов и блюд (создать\_списание\_продукта, создать\_списание\_блюда).

Триггеры: Реализованы триггерные функции (Приложение Е) для автоматического выполнения действий при модификации данных. Ключевыми являются триггеры, обеспечивающие автоматическое обновление остатков продуктов на складе при поступлении (TG\_ОбновитьОстатокПриПоступлении), списании продуктов (TG\_ОбновитьОстатокПриСписанииПродукта), списании блюд (с пересчетом ингредиентов – TG\_ОбновитьОстатокПриСписанииБлюда) и при продаже блюд (также с пересчетом ингредиентов – TG\_ОбновитьОстаткиПриПродажеБлюда).

Управление доступом: Настроена ролевая модель безопасности (Приложение Ж). Созданы три основные роли пользователей (владелец, администраторсклада, работник), каждой из которых предоставлены гранулированные права доступа (на выполнение операций SELECT, INSERT, UPDATE, DELETE с таблицами и представлениями, а также EXECUTE для функций) в строгом соответствии с их предполагаемыми обязанностями в системе.

Заполнение начальными данными: База данных была заполнена начальными справочными данными и тестовыми операционными данными (Приложение И) для обеспечения возможности демонстрации и тестирования системы.

## 3.3. Разработка прототипа веб-приложения

Прототип веб-приложения был реализован для демонстрации взаимодействия с базой данных и выполнения основных пользовательских сценариев. Приложение построено по трехуровневой архитектуре (клиент-сервер-БД) с использованием Flask.

Маршрутизация и логика: Определены маршруты (URL) для каждой функциональной страницы. Логика обработки запросов, взаимодействия с БД (выполнение SQL-запросов, вызов хранимых процедур) и подготовки данных для отображения реализована в Python-функциях-обработчиках.

Пользовательский интерфейс: Для визуализации данных и взаимодействия с пользователем использовались HTML-шаблоны с применением шаблонизатора Jinja2 и базовых стилей Bootstrap для оформления.

Функционал по ролям:

* Работник: Реализован интерфейс для просмотра меню, создания новых заказов (с выбором кассы и себя как оформителя), добавления блюд в активный заказ с указанием количества, изменения типа оплаты заказа.
* Администратор Склада: Доступен просмотр актуальных остатков продуктов на складах. Реализован функционал для оформления поступлений продуктов (создание документа поступления и добавление позиций с указанием количества и цены закупки) и оформления списаний (выбор типа объекта – продукт или блюдо, указание склада, причины, количества и комментария).
* Владелец: Предоставлен доступ к набору агрегированных отчетов, формируемых на основе представлений в БД (текущие остатки, список заказов, детализация блюд, сводка продаж по блюдам, полная информация по списаниям).

## 3.4. Тестирование системы

Тестирование разработанной системы проводилось комплексно и включало несколько направлений:

1. Запросы базы данных:

* Получение текущих остатков продуктов на всех складах (лежит в основе представления VТекущиеОстатки).
* Получение остатков продуктов на конкретном складе.
* Просмотр списка всех заказов с основной информацией (лежит в основе представления VСписокЗаказов).
* Детализация конкретного заказа с перечнем блюд и их стоимостью (лежит в основе представления VДетализацияЗаказов).
* Получение информации о поступлениях продуктов (список документов поступления, детализация конкретного поступления).
* Просмотр полной информации по всем списаниям продуктов и блюд (лежит в основе представления VПолнаяИнформацияПоСписаниям).
* Получение списка сотрудников и их должностей.
* Просмотр меню блюд с ценами и категориями.
* Расчет себестоимости конкретного блюда на основе средних цен закупки ингредиентов (использует функцию "РассчитатьСреднююЦенуЗакупки").
* Расчет маржинальности по конкретному блюду и в процентах.
* Получение сводки продаж по блюдам: количество проданных порций и общая сумма продаж по каждому блюду (лежит в основе представления VСводкаПродажПоБлюдам).
* Определение общей суммы продаж за выбранный период (например, день, неделя, месяц).
* Расчет общих затрат на закупку продуктов за период.
* Выявление блюд, не продававшихся в течение определенного периода.
* Определение топ-N самых продаваемых блюд или самых активных сотрудников.
* Получение информации о рецептуре блюд: список ингредиентов и их количество для каждого блюда (лежит в основе представления VДетализацияБлюдСИнгредиентами).
* Запрос на получение средней цены закупки для конкретного продукта (вызов функции "РассчитатьСреднююЦенуЗакупки").

1. Модульное тестирование объектов БД:  Проверялась корректность создания всех объектов базы данных (таблиц, представлений, функций, триггеров) путем выполнения соответствующих SQL-скриптов и анализа их результатов. Тестировалась логика работы хранимых функций путем их вызова с различными наборами входных параметров. Особое внимание уделялось проверке корректности срабатывания триггеров: отслеживалось автоматическое обновление остатков при поступлениях, продажах и списаниях, а также корректность каскадных операций при необходимости.
2. Тестирование прав доступа:  Осуществлялось подключение к базе данных под каждой из созданных ролей (владелец, администраторсклада, работник). Проверялась возможность выполнения разрешенных операций и невозможность выполнения запрещенных операций в соответствии с назначенной матрицей прав.
3. Функциональное тестирование веб-приложения:  Проводилось тестирование всех реализованных пользовательских сценариев для каждой роли. Проверялась корректность отображения данных, работа форм ввода, обработка пользовательских действий (например, создание заказа, добавление продукта в поступление, оформление списания), а также корректность сообщений об успехе или ошибках операций.
4. Интеграционное тестирование:  Проверялось корректное взаимодействие между веб-приложением и базой данных, правильность передачи данных и вызова функций БД.
5. Тестирование на основе данных:  Выполнялись различные SQL-запросы (примеры приведены в Приложении Ж) для проверки способности системы извлекать требуемую аналитическую и оперативную информацию, а также для контроля целостности и непротиворечивости данных после выполнения различных операций[6].

По результатам тестирования можно сделать вывод о работоспособности основных модулей системы. Функционал базы данных и веб-приложения соответствует поставленным задачам для прототипа. Были выявлены и устранены ошибки, связанные с правами доступа пользователей к объектам базы данных. Система корректно обрабатывает основные операции учета и предоставляет информацию для анализа.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была успешно спроектирована и реализована реляционная база данных для информационной поддержки ключевых процессов учета и анализа в сети кафе-ресторанов "Sushi-shop". Также был разработан прототип веб-приложения, демонстрирующий взаимодействие с созданной базой данных и предоставляющий базовый функционал для различных категорий пользователей.

Основные достигнутые результаты:

1. Проведен детальный анализ предметной области, на основе которого были сформулированы требования к информационной системе и ее базе данных.
2. Разработаны концептуальная, логическая и физическая модели данных, обеспечивающие структурированное и нормализованное (достигнута как минимум третья нормальная форма) хранение информации, что минимизирует избыточность и поддерживает целостность данных.
3. Обоснован выбор СУБД PostgreSQL, и в ее среде реализована структура базы данных, включающая таблицы, связи, индексы, представления, хранимые функции и триггеры, автоматизирующие часть бизнес-логики (например, учет остатков).
4. Реализована ролевая модель доступа, обеспечивающая безопасное взаимодействие с данными для различных категорий пользователей (владелец, администраторсклада, работник).
5. Создан прототип веб-приложения на Flask, позволяющий выполнять основные операции: для работника – управление заказами; для администратора склада – учет поступлений, списаний и просмотр остатков; для владельца – доступ к аналитическим отчетам.
6. Проведено тестирование системы, подтвердившее ее работоспособность и соответствие основным требованиям.

Разработанная база данных и прототип приложения могут служить надежной основой для дальнейшего развития полнофункциональной информационной системы управления сетью ресторанов "Sushi-shop". Практическая значимость работы заключается в создании конкретного инструментария, адаптированного под нужды ресторанного бизнеса, а также в демонстрации применения современных подходов к проектированию и реализации баз данных.

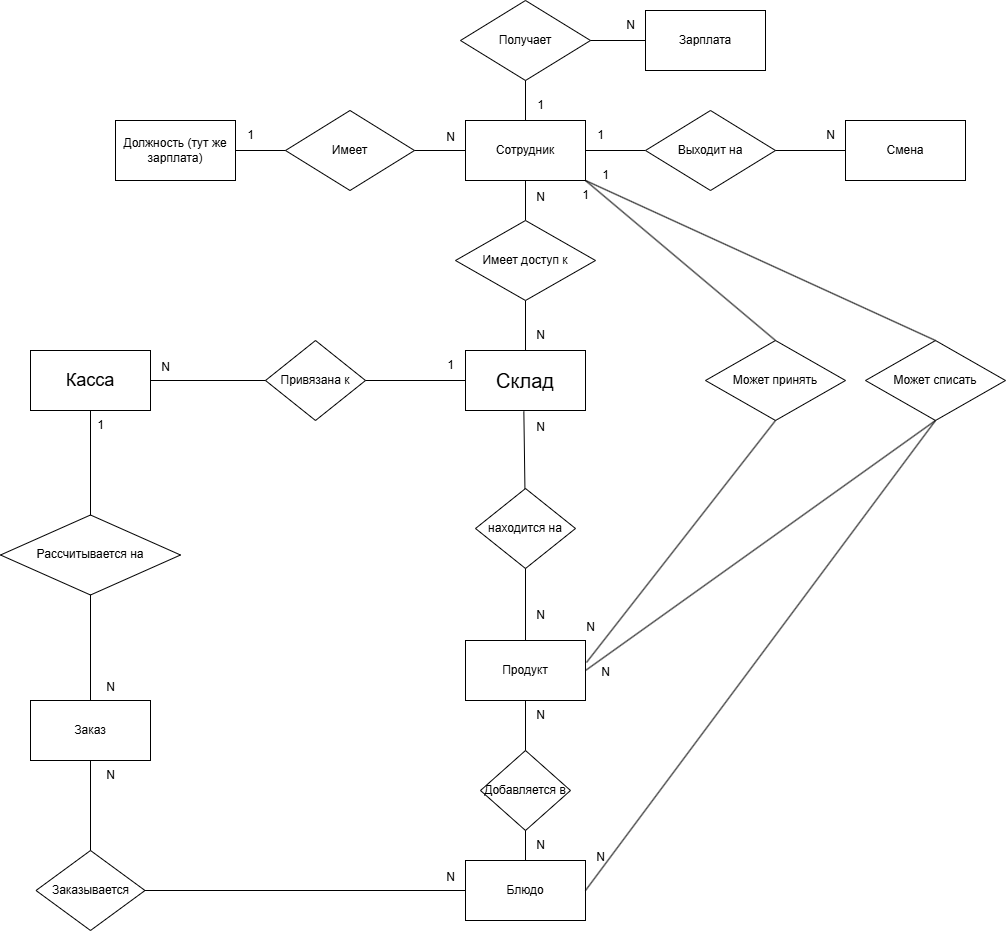
Направления для возможного дальнейшего развития проекта включают: расширение модели данных (например, учет поставщиков, партионный учет продуктов), детализация расчета себестоимости, разработка более продвинутого пользовательского интерфейса с расширенным функционалом для всех ролей, интеграция с внешними системами (бухгалтерия, онлайн-заказы), а также внедрение более сложных аналитических инструментов и средств визуализации данных.

Таким образом, поставленные в курсовой работе цели и задачи были выполнены в полном объеме.

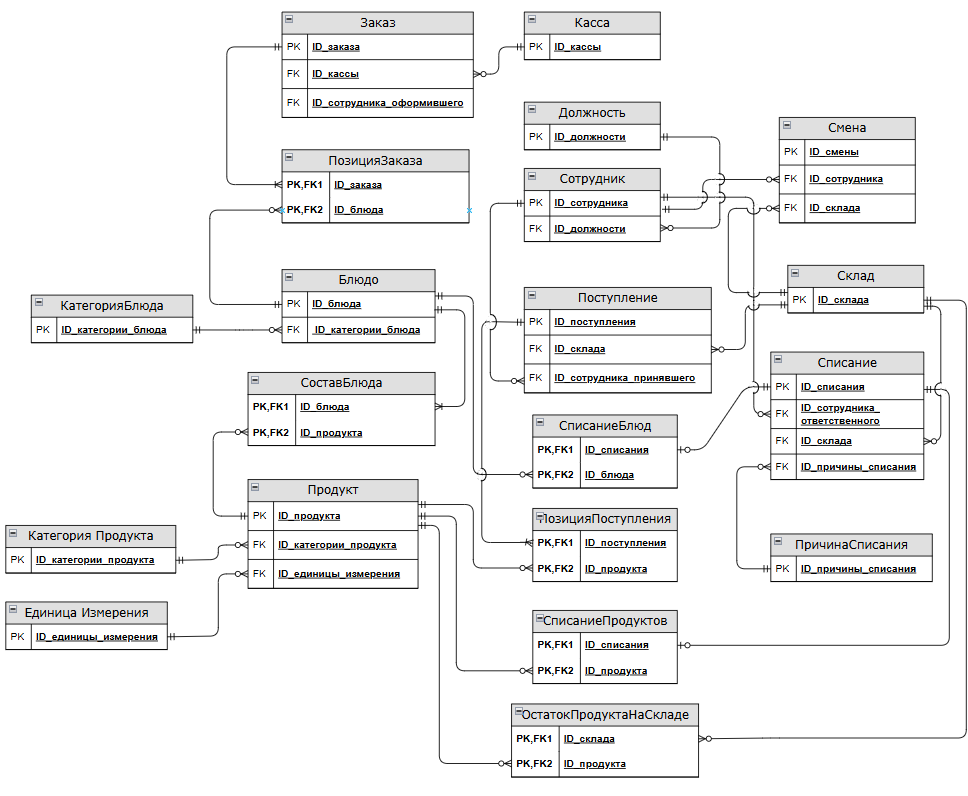
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Чен П. П.–Ш. Модель "сущность–связь" — шаг к единому представлению данных // СУБД. — 1995. — № 3. — С. 131–137.
2. POS vs ERP Program System: Which One is Better? [Электронный ресурс]. — URL: [https://www.hashmicro.com/blog/pos-erp-program/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.hashmicro.com%2Fblog%2Fpos-erp-program%2F) (дата обращения: 10.05.2025).
3. Системы управления запасами (Inventory Management Systems, IMS) [Электронный ресурс]. — URL: [https://soware.ru/categories/inventory-management-systems#definition](https://www.google.com/url?sa=E&q=https%3A%2F%2Fsoware.ru%2Fcategories%2Finventory-management-systems%23definition) (дата обращения: 10.05.2025).
4. Кара-Ушанов, В. Ю. Базы данных : учебное пособие / В. Ю. Кара-Ушанов ; М-во науки и высш. образования Рос. Федерации, Урал. федер. ун-т. – Екатеринбург : Изд-во Урал. ун-та, 2018. – 108 с. – ISBN 978-5-7996-2553-3. – URL: https://study.urfu.ru/Aid/Publication/13604/1/Kara-Ushanov.pdf (дата обращения: 13.05.2025).
5. Кузнецов, С. Д. Основы баз данных: учебное пособие / С. Д. Кузнецов. – 2-е изд., испр. – Москва: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2007. – 484 с. – (Основы информационных технологий). – ISBN 978-5-94774-736-2.
6. Майерс, Г. Искусство тестирования программ / Г. Майерс, К. Сандлер, Т. Баджетт ; пер. с англ. – 3-е изд. – Москва : Диалектика, 2012. – 272 с. – ISBN 978-5-8459-1796-6.
7. PostgreSQL: Документация: Часть I. Учебник [Электронный ресурс] // PostgreSQL Global Development Group. – URL: https://postgrespro.ru/docs/postgresql/13/tutorial (дата обращения: 13.05.2025).

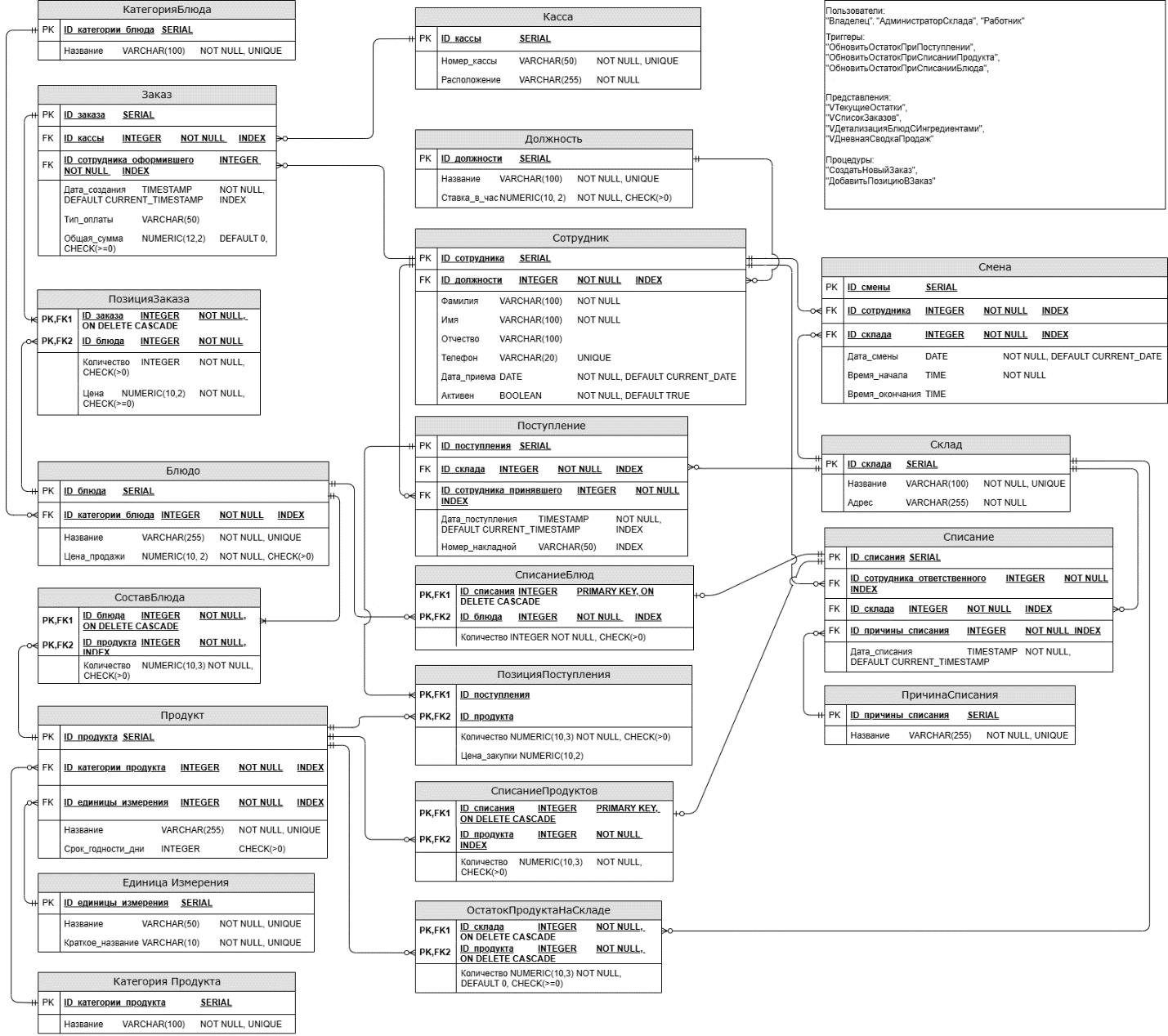
ПРИЛОЖЕНИЕ А



ПРИЛОЖЕНИЕ Б



ПРИЛОЖЕНИЕ В



ПРИЛОЖЕНИЕ Г

-- 06\_create\_views.sql

-- Скрипт для создания представлений (views)

BEGIN;

-- 1. Представление: Текущие остатки продуктов на складах

DROP VIEW IF EXISTS "VТекущиеОстатки" CASCADE;

CREATE VIEW "VТекущиеОстатки" AS

SELECT

s."Название" AS "Склад",

p."Название" AS "Продукт",

kp."Название" AS "Категория\_продукта",

ops."Количество",

ei."Краткое\_название" AS "Ед\_изм"

FROM

"ОстатокПродуктаНаСкладе" ops

JOIN

"Склад" s ON ops."ID\_склада" = s."ID\_склада"

JOIN

"Продукт" p ON ops."ID\_продукта" = p."ID\_продукта"

JOIN

"ЕдиницаИзмерения" ei ON p."ID\_единицы\_измерения" = ei."ID\_единицы\_измерения"

JOIN

"КатегорияПродукта" kp ON p."ID\_категории\_продукта" = kp."ID\_категории\_продукта"

ORDER BY

s."Название", p."Название";

-- 2. Представление: Список заказов с основной информацией

DROP VIEW IF EXISTS "VСписокЗаказов" CASCADE;

CREATE VIEW "VСписокЗаказов" AS

SELECT

z."ID\_заказа",

z."Дата\_создания",

s."Фамилия" || ' ' || LEFT(s."Имя", 1) || '.' || COALESCE(LEFT(s."Отчество", 1) || '.', '') AS "Сотрудник\_оформивший",

k."Номер\_кассы",

k."Расположение" AS "Расположение\_кассы",

z."Тип\_оплаты",

z."Общая\_сумма"

FROM

"Заказ" z

JOIN

"Сотрудник" s ON z."ID\_сотрудника\_оформившего" = s."ID\_сотрудника"

JOIN

"Касса" k ON z."ID\_кассы" = k."ID\_кассы"

ORDER BY

z."Дата\_создания" DESC;

-- 3. Представление: Детализация заказов с блюдами

DROP VIEW IF EXISTS "VДетализацияЗаказов" CASCADE;

CREATE VIEW "VДетализацияЗаказов" AS

SELECT

z."ID\_заказа",

z."Дата\_создания" AS "Дата\_заказа",

b."Название" AS "Блюдо",

kb."Название" AS "Категория\_блюда",

pz."Количество",

pz."Цена\_на\_момент\_заказа" AS "Цена\_за\_единицу",

(pz."Количество" \* pz."Цена\_на\_момент\_заказа") AS "Сумма\_позиции"

FROM

"ПозицияЗаказа" pz

JOIN

"Заказ" z ON pz."ID\_заказа" = z."ID\_заказа"

JOIN

"Блюдо" b ON pz."ID\_блюда" = b."ID\_блюда"

JOIN

"КатегорияБлюда" kb ON b."ID\_категории\_блюда" = kb."ID\_категории\_блюда"

ORDER BY

z."ID\_заказа", b."Название";

-- 4. Представление: Детализация блюд с ингредиентами (Рецептуры) - ИСПРАВЛЕНО (версия 4, с DROP)

DROP VIEW IF EXISTS "VДетализацияБлюдСИнгредиентами" CASCADE;

CREATE VIEW "VДетализацияБлюдСИнгредиентами" AS

SELECT

b."Название" AS "Блюдо",

kb."Название" AS "Категория\_блюда",

b."Цена\_продажи" AS "Цена\_блюда", -- Новый столбец

p."Название" AS "Ингредиент",

kp."Название" AS "Категория\_ингредиента",

sb."Количество" AS "Количество\_ингредиента",

ei."Краткое\_название" AS "Ед\_изм\_ингредиента"

FROM

"СоставБлюда" sb

JOIN

"Блюдо" b ON sb."ID\_блюда" = b."ID\_блюда"

JOIN

"КатегорияБлюда" kb ON b."ID\_категории\_блюда" = kb."ID\_категории\_блюда"

JOIN

"Продукт" p ON sb."ID\_продукта" = p."ID\_продукта"

JOIN

"КатегорияПродукта" kp ON p."ID\_категории\_продукта" = kp."ID\_категории\_продукта"

JOIN

"ЕдиницаИзмерения" ei ON p."ID\_единицы\_измерения" = ei."ID\_единицы\_измерения"

ORDER BY

b."Название", p."Название";

-- 5. Представление: Сводка продаж по блюдам

DROP VIEW IF EXISTS "VСводкаПродажПоБлюдам" CASCADE;

CREATE VIEW "VСводкаПродажПоБлюдам" AS

SELECT

b."Название" AS "Блюдо",

kb."Название" AS "Категория\_блюда",

SUM(pz."Количество") AS "Продано\_порций",

SUM(pz."Количество" \* pz."Цена\_на\_момент\_заказа") AS "Общая\_сумма\_продаж"

FROM

"ПозицияЗаказа" pz

JOIN

"Блюдо" b ON pz."ID\_блюда" = b."ID\_блюда"

JOIN

"КатегорияБлюда" kb ON b."ID\_категории\_блюда" = kb."ID\_категории\_блюда"

GROUP BY

b."Название", kb."Название"

ORDER BY

"Общая\_сумма\_продаж" DESC;

-- 6. Представление: Полная информация по списаниям

DROP VIEW IF EXISTS "VПолнаяИнформацияПоСписаниям" CASCADE;

CREATE VIEW "VПолнаяИнформацияПоСписаниям" AS

SELECT

s."ID\_списания",

s."Дата\_списания",

sk."Название" AS "Склад",

st."Фамилия" || ' ' || LEFT(st."Имя", 1) || '.' AS "Сотрудник\_ответственный",

ps."Название" AS "Причина\_списания",

'Продукт' AS "Тип\_объекта",

p."Название" AS "Наименование\_объекта",

sp."Количество",

ei."Краткое\_название" AS "Ед\_изм",

s."Комментарий"

FROM

"СписаниеПродуктов" sp

JOIN

"Списание" s ON sp."ID\_списания" = s."ID\_списания"

JOIN

"Продукт" p ON sp."ID\_продукта" = p."ID\_продукта"

JOIN

"ЕдиницаИзмерения" ei ON p."ID\_единицы\_измерения" = ei."ID\_единицы\_измерения"

JOIN

"Склад" sk ON s."ID\_склада" = sk."ID\_склада"

JOIN

"Сотрудник" st ON s."ID\_сотрудника\_ответственного" = st."ID\_сотрудника"

JOIN

"ПричинаСписания" ps ON s."ID\_причины\_списания" = ps."ID\_причины\_списания"

UNION ALL

SELECT

s."ID\_списания",

s."Дата\_списания",

sk."Название" AS "Склад",

st."Фамилия" || ' ' || LEFT(st."Имя", 1) || '.' AS "Сотрудник\_ответственный",

ps."Название" AS "Причина\_списания",

'Блюдо' AS "Тип\_объекта",

b."Название" AS "Наименование\_объекта",

sb."Количество",

'шт.' AS "Ед\_изм",

s."Комментарий"

FROM

"СписаниеБлюд" sb

JOIN

"Списание" s ON sb."ID\_списания" = s."ID\_списания"

JOIN

"Блюдо" b ON sb."ID\_блюда" = b."ID\_блюда"

JOIN

"Склад" sk ON s."ID\_склада" = sk."ID\_склада"

JOIN

"Сотрудник" st ON s."ID\_сотрудника\_ответственного" = st."ID\_сотрудника"

JOIN

"ПричинаСписания" ps ON s."ID\_причины\_списания" = ps."ID\_причины\_списания"

ORDER BY

"Дата\_списания" DESC, "ID\_списания" DESC;

COMMIT;

ПРИЛОЖЕНИЕ Д

-- 03\_create\_functions.sql

-- Скрипт для создания хранимых функций и процедур

BEGIN;

-- Функция для создания списания ПРОДУКТА (оставляем как было)

CREATE OR REPLACE FUNCTION создать\_списание\_продукта (

p\_id\_склада INTEGER,

p\_id\_сотрудника\_ответственного INTEGER,

p\_id\_причины\_списания INTEGER,

p\_комментарий TEXT,

p\_id\_продукта INTEGER,

p\_количество NUMERIC

)

RETURNS INTEGER -- Возвращаем ID созданного списания (из родительской таблицы "Списание")

AS $$

DECLARE

inserted\_sписание\_id INTEGER;

BEGIN

INSERT INTO "Списание" (

"ID\_склада", "ID\_сотрудника\_ответственного", "ID\_причины\_списания", "Комментарий", "Дата\_списания"

) VALUES (

p\_id\_склада, p\_id\_сотрудника\_ответственного, p\_id\_причины\_списания, p\_комментарий, CURRENT\_TIMESTAMP

) RETURNING "ID\_списания" INTO inserted\_sписание\_id;

INSERT INTO "СписаниеПродуктов" (

"ID\_списания",

"ID\_склада", "ID\_сотрудника\_ответственного", "ID\_причины\_списания", "Комментарий", "Дата\_списания",

"ID\_продукта", "Количество"

) VALUES (

inserted\_sписание\_id,

p\_id\_склада, p\_id\_сотрудника\_ответственного, p\_id\_причины\_списания, p\_комментарий, (SELECT "Дата\_списания" FROM "Списание" WHERE "ID\_списания" = inserted\_sписание\_id),

p\_id\_продукта, p\_количество

);

RETURN inserted\_sписание\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE WARNING 'Ошибка в функции создать\_списание\_продукта: % (%)', SQLERRM, SQLSTATE;

RAISE;

END;

$$ LANGUAGE plpgsql;

-- Функция для создания списания БЛЮДА (оставляем как было)

CREATE OR REPLACE FUNCTION создать\_списание\_блюда (

p\_id\_склада INTEGER,

p\_id\_сотрудника\_ответственного INTEGER,

p\_id\_причины\_списания INTEGER,

p\_комментарий TEXT,

p\_id\_блюда INTEGER,

p\_количество INTEGER

)

RETURNS INTEGER -- Возвращаем ID созданного списания

AS $$

DECLARE

inserted\_sписание\_id INTEGER;

BEGIN

INSERT INTO "Списание" (

"ID\_склада", "ID\_сотрудника\_ответственного", "ID\_причины\_списания", "Комментарий", "Дата\_списания"

) VALUES (

p\_id\_склада, p\_id\_сотрудника\_ответственного, p\_id\_причины\_списания, p\_комментарий, CURRENT\_TIMESTAMP

) RETURNING "ID\_списания" INTO inserted\_sписание\_id;

INSERT INTO "СписаниеБлюд" (

"ID\_списания",

"ID\_склада", "ID\_сотрудника\_ответственного", "ID\_причины\_списания", "Комментарий", "Дата\_списания",

"ID\_блюда", "Количество"

) VALUES (

inserted\_sписание\_id,

p\_id\_склада, p\_id\_сотрудника\_ответственного, p\_id\_причины\_списания, p\_комментарий, (SELECT "Дата\_списания" FROM "Списание" WHERE "ID\_списания" = inserted\_sписание\_id),

p\_id\_блюда, p\_количество

);

RETURN inserted\_sписание\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE WARNING 'Ошибка в функции создать\_списание\_блюда: % (%)', SQLERRM, SQLSTATE;

RAISE;

END;

$$ LANGUAGE plpgsql;

-- НОВАЯ ФУНКЦИЯ: Рассчитать среднюю цену закупки продукта

CREATE OR REPLACE FUNCTION "РассчитатьСреднююЦенуЗакупки"(id\_продукта\_вход INTEGER)

RETURNS NUMERIC AS $$

SELECT AVG("Цена\_закупки") FROM "ПозицияПоступления" WHERE "ID\_продукта" = id\_продукта\_вход;

$$ LANGUAGE sql; -- Для чистого SQL не нужен BEGIN/END, если одно выражение

-- НОВАЯ ПРОЦЕДУРА: Создать новый заказ

CREATE OR REPLACE FUNCTION "СоздатьНовыйЗаказ"(

id\_кассы\_вход INTEGER,

id\_сотрудника\_вход INTEGER,

OUT id\_нового\_заказа\_выход INTEGER

)

LANGUAGE plpgsql AS $$

BEGIN

INSERT INTO "Заказ"("ID\_кассы", "ID\_сотрудника\_оформившего")

VALUES (id\_кассы\_вход, id\_сотрудника\_вход)

RETURNING "ID\_заказа" INTO id\_нового\_заказа\_выход;

END;

$$;

-- НОВАЯ ПРОЦЕДУРА: Добавить позицию в заказ и обновить общую сумму заказа

CREATE OR REPLACE FUNCTION "ДобавитьПозициюВЗаказ"(

id\_заказа\_вход INTEGER,

id\_блюда\_вход INTEGER,

количество\_вход INTEGER

)

RETURNS VOID -- Процедура не возвращает значение напрямую, она модифицирует данные

LANGUAGE plpgsql AS $$

DECLARE

цена\_блюда\_текущая NUMERIC;

BEGIN

-- Получаем текущую цену блюда

SELECT "Цена\_продажи" INTO цена\_блюда\_текущая

FROM "Блюдо"

WHERE "ID\_блюда" = id\_блюда\_вход;

IF NOT FOUND THEN

RAISE EXCEPTION 'Блюдо с ID % не найдено.', id\_блюда\_вход;

END IF;

-- Добавляем позицию в заказ

-- Используем ON CONFLICT для случая, если такая позиция уже есть (например, увеличиваем количество)

INSERT INTO "ПозицияЗаказа"("ID\_заказа", "ID\_блюда", "Количество", "Цена\_на\_момент\_заказа")

VALUES (id\_заказа\_вход, id\_блюда\_вход, количество\_вход, цена\_блюда\_текущая)

ON CONFLICT ("ID\_заказа", "ID\_блюда") DO UPDATE SET

"Количество" = "ПозицияЗаказа"."Количество" + excluded."Количество",

"Цена\_на\_момент\_заказа" = excluded."Цена\_на\_момент\_заказа"; -- Обновляем цену, если она могла измениться

-- Обновляем общую сумму заказа

UPDATE "Заказ"

SET "Общая\_сумма" = (

SELECT COALESCE(SUM("Количество" \* "Цена\_на\_момент\_заказа"), 0)

FROM "ПозицияЗаказа"

WHERE "ID\_заказа" = id\_заказа\_вход

)

WHERE "ID\_заказа" = id\_заказа\_вход;

END;

$$;

COMMIT;

ПРИЛОЖЕНИЕ Е

-- 07\_create\_triggers.sql

-- Скрипт для создания триггеров

BEGIN;

-- 1. Триггер для обновления остатков продуктов при ПОСТУПЛЕНИИ (без изменений)

CREATE OR REPLACE FUNCTION "TG\_ОбновитьОстатокПриПоступлении"()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO "ОстатокПродуктаНаСкладе" ("ID\_склада", "ID\_продукта", "Количество")

VALUES (

(SELECT "ID\_склада" FROM "Поступление" WHERE "ID\_поступления" = NEW."ID\_поступления"),

NEW."ID\_продукта",

NEW."Количество"

)

ON CONFLICT ("ID\_склада", "ID\_продукта") DO UPDATE SET

"Количество" = "ОстатокПродуктаНаСкладе"."Количество" + EXCLUDED."Количество";

RAISE NOTICE 'Триггер TG\_ОбновитьОстатокПриПоступлении: Остаток продукта ID % на складе ID % обновлен на +%.',

NEW."ID\_продукта", (SELECT "ID\_склада" FROM "Поступление" WHERE "ID\_поступления" = NEW."ID\_поступления"), NEW."Количество";

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER "trg\_after\_insert\_ПозицияПоступления\_update\_остаток"

AFTER INSERT ON "ПозицияПоступления"

FOR EACH ROW

EXECUTE FUNCTION "TG\_ОбновитьОстатокПриПоступлении"();

-- 2. Триггер для обновления остатков продуктов при СПИСАНИИ ПРОДУКТА (без изменений)

CREATE OR REPLACE FUNCTION "TG\_ОбновитьОстатокПриСписанииПродукта"()

RETURNS TRIGGER AS $$

DECLARE

v\_id\_склада\_списания INTEGER;

BEGIN

SELECT s."ID\_склада" INTO v\_id\_склада\_списания

FROM "Списание" s WHERE s."ID\_списания" = NEW."ID\_списания";

IF v\_id\_склада\_списания IS NULL THEN

RAISE WARNING 'Триггер TG\_ОбновитьОстатокПриСписанииПродукта: Не удалось найти склад для списания ID %.', NEW."ID\_списания";

RETURN NEW;

END IF;

UPDATE "ОстатокПродуктаНаСкладе"

SET "Количество" = "Количество" - NEW."Количество"

WHERE "ID\_склада" = v\_id\_склада\_списания AND "ID\_продукта" = NEW."ID\_продукта";

IF NOT FOUND THEN

RAISE WARNING 'Триггер TG\_ОбновитьОстатокПриСписанииПродукта: Продукт ID % не найден на складе ID % для списания или остаток уже 0. Списание произведено.',

NEW."ID\_продукта", v\_id\_склада\_списания;

-- Можно добавить логику для создания записи с отрицательным остатком, если это допустимо,

-- или возбудить исключение, если остаток не может быть отрицательным и проверка на это не сделана ранее.

-- Для простоты оставляем предупреждение.

END IF;

RAISE NOTICE 'Триггер TG\_ОбновитьОстатокПриСписанииПродукта: Остаток продукта ID % на складе ID % обновлен на -%.',

NEW."ID\_продукта", v\_id\_склада\_списания, NEW."Количество";

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER "trg\_after\_insert\_СписаниеПродуктов\_update\_остаток"

AFTER INSERT ON "СписаниеПродуктов"

FOR EACH ROW

EXECUTE FUNCTION "TG\_ОбновитьОстатокПриСписанииПродукта"();

-- 3. НОВЫЙ ТРИГГЕР: Обновление остатков ИНГРЕДИЕНТОВ при СПИСАНИИ БЛЮДА

-- (когда блюдо списывается целиком по причине порчи, проработки и т.д.)

CREATE OR REPLACE FUNCTION "TG\_ОбновитьОстатокПриСписанииБлюда"()

RETURNS TRIGGER AS $$

DECLARE

v\_id\_склада\_списания INTEGER;

rec\_состав RECORD;

BEGIN

-- Получаем ID\_склада из родительской таблицы "Списание"

SELECT s."ID\_склада" INTO v\_id\_склада\_списания

FROM "Списание" s WHERE s."ID\_списания" = NEW."ID\_списания"; -- NEW - это строка из "СписаниеБлюд"

IF v\_id\_склада\_списания IS NULL THEN

RAISE WARNING 'Триггер TG\_ОбновитьОстатокПриСписанииБлюда: Не удалось найти склад для списания ингредиентов блюда (списание ID %).', NEW."ID\_списания";

RETURN NEW;

END IF;

-- Перебираем все ингредиенты списываемого блюда

FOR rec\_состав IN

SELECT "ID\_продукта", "Количество" AS "Количество\_в\_рецепте"

FROM "СоставБлюда"

WHERE "ID\_блюда" = NEW."ID\_блюда" -- NEW."ID\_блюда" из таблицы "СписаниеБлюд"

LOOP

UPDATE "ОстатокПродуктаНаСкладе"

SET "Количество" = "Количество" - (rec\_состав."Количество\_в\_рецепте" \* NEW."Количество") -- NEW."Количество" - это кол-во списываемых блюд

WHERE "ID\_склада" = v\_id\_склада\_списания AND "ID\_продукта" = rec\_состав."ID\_продукта";

IF NOT FOUND THEN

RAISE WARNING 'Триггер TG\_ОбновитьОстатокПриСписанииБлюда: Ингредиент ID % не найден на складе ID % для списания или остаток уже 0. Списание произведено.',

rec\_состав."ID\_продукта", v\_id\_склада\_списания;

END IF;

RAISE NOTICE 'Триггер TG\_ОбновитьОстатокПриСписанииБлюда: Остаток ингредиента ID % на складе ID % обновлен на -% (для списанного блюда ID %).',

rec\_состав."ID\_продукта", v\_id\_склада\_списания, (rec\_состав."Количество\_в\_рецепте" \* NEW."Количество"), NEW."ID\_блюда";

END LOOP;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER "trg\_after\_insert\_СписаниеБлюд\_update\_остатки\_ингредиентов"

AFTER INSERT ON "СписаниеБлюд"

FOR EACH ROW

EXECUTE FUNCTION "TG\_ОбновитьОстатокПриСписанииБлюда"();

-- 4. ДОРАБОТАННЫЙ ТРИГГЕР: Обновление остатков ИНГРЕДИЕНТОВ при ПРОДАЖЕ БЛЮДА

-- (срабатывает при добавлении позиции в заказ)

CREATE OR REPLACE FUNCTION "TG\_ОбновитьОстаткиПриПродажеБлюда"()

RETURNS TRIGGER AS $$

DECLARE

rec\_состав RECORD;

v\_id\_склада\_списания\_ингредиентов INTEGER; -- Склад по умолчанию для списания ингредиентов

BEGIN

-- Определяем склад для списания ингредиентов.

-- ВАЖНОЕ ДОПУЩЕНИЕ ДЛЯ КУРСОВОЙ: Все ингредиенты для проданных блюд

-- списываются с основного кухонного склада с ID = 1 (или выберите другой подходящий ID).

-- В реальной системе эта логика должна быть более гибкой.

v\_id\_склада\_списания\_ингредиентов := 1; -- ЗАМЕНИТЕ НА ID ВАШЕГО ОСНОВНОГО КУХОННОГО СКЛАДА

RAISE NOTICE 'Триггер TG\_ОбновитьОстаткиПриПродажеБлюда: Попытка списания ингредиентов для блюда ID % со склада ID %.',

NEW."ID\_блюда", v\_id\_склада\_списания\_ингредиентов;

FOR rec\_состав IN

SELECT "ID\_продукта", "Количество" AS "Количество\_в\_рецепте"

FROM "СоставБлюда"

WHERE "ID\_блюда" = NEW."ID\_блюда" -- NEW - это строка из "ПозицияЗаказа"

LOOP

UPDATE "ОстатокПродуктаНаСкладе"

SET "Количество" = "Количество" - (rec\_состав."Количество\_в\_рецепте" \* NEW."Количество") -- NEW."Количество" - это кол-во проданных блюд

WHERE "ID\_склада" = v\_id\_склада\_списания\_ингредиентов AND "ID\_продукта" = rec\_состав."ID\_продукта";

IF NOT FOUND THEN

RAISE WARNING 'Триггер TG\_ОбновитьОстаткиПриПродажеБлюда: Ингредиент ID % не найден на складе ID % для списания при продаже или остаток уже 0. Списание произведено.',

rec\_состав."ID\_продукта", v\_id\_склада\_списания\_ингредиентов;

-- ВАЖНО: Здесь должна быть логика обработки ситуации, когда ингредиента нет на складе!

-- Например, запрет добавления в заказ, откат транзакции или специальное уведомление.

-- Для курсовой можно оставить предупреждение, но в описании указать на это ограничение.

END IF;

RAISE NOTICE 'Триггер TG\_ОбновитьОстаткиПриПродажеБлюда: Остаток ингредиента ID % на складе ID % обновлен на -% (для проданного блюда ID % в заказе ID %).',

rec\_состав."ID\_продукта", v\_id\_склада\_списания\_ингредиентов,

(rec\_состав."Количество\_в\_рецепте" \* NEW."Количество"), NEW."ID\_блюда", NEW."ID\_заказа";

END LOOP;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Заменяем старый триггер или создаем новый, если его не было

CREATE OR REPLACE TRIGGER "trg\_after\_insert\_ПозицияЗаказа\_update\_остатки\_ингредиентов"

AFTER INSERT ON "ПозицияЗаказа"

FOR EACH ROW

EXECUTE FUNCTION "TG\_ОбновитьОстаткиПриПродажеБлюда"();

COMMIT;

ПРИЛОЖЕНИЕ Ж

-- 08\_create\_roles\_and\_permissions.sql

-- ВАЖНО: Выполнять под суперпользователем (например, postgres)!

-- Этот скрипт может потребовать ДВУХ запусков.

-- Первый запуск очистит максимум зависимостей и удалит роли.

-- Второй запуск создаст новые роли и выдаст права.

-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

-- ЭТАП 1: ОЧИСТКА И УДАЛЕНИЕ СТАРЫХ РОЛЕЙ (ЗАПУСТИТЬ ПЕРВЫМ)

-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

DO $$

DECLARE

role\_name\_quoted TEXT;

role\_name\_lower TEXT;

roles\_to\_clean\_quoted TEXT[] := ARRAY['"Владелец"', '"АдминистраторСклада"', '"Работник"'];

roles\_to\_clean\_lower TEXT[] := ARRAY['владелец', 'администраторсклада', 'работник'];

BEGIN

RAISE NOTICE '--- STARTING CLEANUP PHASE ---';

-- Очистка ролей, созданных В КАВЫЧКАХ

FOREACH role\_name\_quoted IN ARRAY roles\_to\_clean\_quoted

LOOP

-- Используем trim для получения чистого имени для проверки в pg\_roles

IF EXISTS (SELECT 1 FROM pg\_roles WHERE rolname = trim(BOTH '"' FROM role\_name\_quoted)) THEN

RAISE NOTICE 'Cleaning up quoted role: %', role\_name\_quoted;

EXECUTE 'REASSIGN OWNED BY ' || role\_name\_quoted || ' TO postgres;';

EXECUTE 'DROP OWNED BY ' || role\_name\_quoted || ';';

EXECUTE 'DROP ROLE IF EXISTS ' || role\_name\_quoted || ';';

ELSE

RAISE NOTICE 'Quoted role % does not exist, skipping.', role\_name\_quoted;

END IF;

END LOOP;

-- Очистка ролей, созданных БЕЗ КАВЫЧЕК (в нижнем регистре)

FOREACH role\_name\_lower IN ARRAY roles\_to\_clean\_lower

LOOP

IF EXISTS (SELECT 1 FROM pg\_roles WHERE rolname = role\_name\_lower) THEN

RAISE NOTICE 'Cleaning up lower-case role: %', role\_name\_lower;

EXECUTE 'REASSIGN OWNED BY ' || quote\_ident(role\_name\_lower) || ' TO postgres;';

EXECUTE 'DROP OWNED BY ' || quote\_ident(role\_name\_lower) || ';';

EXECUTE 'DROP ROLE IF EXISTS ' || quote\_ident(role\_name\_lower) || ';';

ELSE

RAISE NOTICE 'Lower-case role % does not exist, skipping.', role\_name\_lower;

END IF;

END LOOP;

RAISE NOTICE '--- CLEANUP PHASE COMPLETE ---';

END$$;

-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

-- ЭТАП 2: СОЗДАНИЕ НОВЫХ РОЛЕЙ И ВЫДАЧА ПРАВ (ЗАПУСТИТЬ ВТОРЫМ, ЕСЛИ ПЕРВЫЙ ПРОШЕЛ УСПЕШНО)

-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

DO $$

BEGIN

IF EXISTS (SELECT 1 FROM pg\_roles WHERE rolname IN ('Владелец', 'АдминистраторСклада', 'Работник', 'владелец', 'администраторсклада', 'работник')) THEN

RAISE WARNING 'Old roles might still exist. Please ensure cleanup was successful or run cleanup again.';

END IF;

END$$;

BEGIN; -- Начинаем новую транзакцию для создания ролей и прав

-- 1. Роль: владелец (без изменений в правах)

DROP ROLE IF EXISTS владелец; -- Добавляем DROP IF EXISTS и сюда для идемпотентности второго этапа

CREATE ROLE владелец WITH LOGIN PASSWORD 'owner\_secure\_password123';

GRANT CONNECT ON DATABASE sushi\_shop\_db TO владелец;

GRANT USAGE ON SCHEMA public TO владелец;

GRANT SELECT ON

"VТекущиеОстатки", "VСписокЗаказов", "VДетализацияЗаказов",

"VДетализацияБлюдСИнгредиентами", "VСводкаПродажПоБлюдам", "VПолнаяИнформацияПоСписаниям"

TO владелец;

GRANT SELECT ON ALL TABLES IN SCHEMA public TO владелец;

GRANT EXECUTE ON FUNCTION создать\_списание\_продукта(INTEGER, INTEGER, INTEGER, TEXT, INTEGER, NUMERIC) TO владелец;

GRANT EXECUTE ON FUNCTION создать\_списание\_блюда(INTEGER, INTEGER, INTEGER, TEXT, INTEGER, INTEGER) TO владелец;

GRANT EXECUTE ON FUNCTION "РассчитатьСреднююЦенуЗакупки"(INTEGER) TO владелец;

GRANT EXECUTE ON FUNCTION "СоздатьНовыйЗаказ"(INTEGER, INTEGER, OUT INTEGER) TO владелец;

GRANT EXECUTE ON FUNCTION "ДобавитьПозициюВЗаказ"(INTEGER, INTEGER, INTEGER) TO владелец;

-- 2. Роль: администраторсклада

DROP ROLE IF EXISTS администраторсклада;

CREATE ROLE администраторсклада WITH LOGIN PASSWORD 'admin\_sklad\_secure\_pass456';

GRANT CONNECT ON DATABASE sushi\_shop\_db TO администраторсклада;

GRANT USAGE ON SCHEMA public TO администраторсклада;

GRANT SELECT ON "VТекущиеОстатки", "VПолнаяИнформацияПоСписаниям" TO администраторсклада;

GRANT SELECT ON TABLE

"Склад", "Продукт", "КатегорияПродукта", "ЕдиницаИзмерения",

"ПричинаСписания", "Сотрудник", "ОстатокПродуктаНаСкладе", -- SELECT уже есть, это хорошо

"Поступление", "ПозицияПоступления", "Списание", "СписаниеПродуктов", "СписаниеБлюд",

"СоставБлюда", "Блюдо"

TO администраторсклада;

GRANT INSERT ON TABLE "Поступление", "ПозицияПоступления",

"Списание", "СписаниеПродуктов", "СписаниеБлюд",

"ОстатокПродуктаНаСкладе" -- <<< ДОБАВЬТЕ INSERT сюда (если его не было)

TO администраторсклада;

GRANT UPDATE ON TABLE "Поступление", "ПозицияПоступления",

"ОстатокПродуктаНаСкладе" -- UPDATE уже был, это хорошо

TO администраторсклада;

GRANT EXECUTE ON FUNCTION создать\_списание\_продукта(INTEGER, INTEGER, INTEGER, TEXT, INTEGER, NUMERIC) TO администраторсклада;

GRANT EXECUTE ON FUNCTION создать\_списание\_блюда(INTEGER, INTEGER, INTEGER, TEXT, INTEGER, INTEGER) TO администраторсклада;

-- 3. Роль: работник (ИЗМЕНЕНИЯ ЗДЕСЬ)

DROP ROLE IF EXISTS работник;

CREATE ROLE работник WITH LOGIN PASSWORD 'worker\_secure\_pass789'; -- ЗАМЕНИТЕ ПАРОЛЬ!

GRANT CONNECT ON DATABASE sushi\_shop\_db TO работник;

GRANT USAGE ON SCHEMA public TO работник;

GRANT SELECT ON "VСписокЗаказов", "VДетализацияЗаказов", "VДетализацияБлюдСИнгредиентами" TO работник;

GRANT SELECT ON TABLE

"Касса", "Смена", "Блюдо", "КатегорияБлюда", "Сотрудник",

"Заказ", "ПозицияЗаказа", "СоставБлюда"

TO работник;

GRANT INSERT ON TABLE "Заказ", "ПозицияЗаказа" TO работник;

-- РАСШИРЯЕМ ПРАВА НА UPDATE для таблицы "Заказ"

GRANT UPDATE ("Тип\_оплаты", "ID\_кассы", "ID\_сотрудника\_оформившего", "Общая\_сумма") ON "Заказ" TO работник;

GRANT UPDATE ("Количество", "Цена\_на\_момент\_заказа") ON "ПозицияЗаказа" TO работник;

GRANT DELETE ON "ПозицияЗаказа" TO работник;

GRANT EXECUTE ON FUNCTION "СоздатьНовыйЗаказ"(INTEGER, INTEGER, OUT INTEGER) TO работник;

GRANT EXECUTE ON FUNCTION "ДобавитьПозициюВЗаказ"(INTEGER, INTEGER, INTEGER) TO работник;

-- Права на последовательности (SERIAL)

DO $$

DECLARE

r RECORD;

rolename\_var TEXT;

BEGIN

FOR rolename\_var IN SELECT unnest(ARRAY['владелец', 'администраторсклада', 'работник'])

LOOP

FOR r IN (SELECT sequence\_name FROM information\_schema.sequences WHERE sequence\_schema = 'public')

LOOP

EXECUTE 'GRANT USAGE, SELECT ON SEQUENCE public.' || quote\_ident(r.sequence\_name) || ' TO ' || quote\_ident(rolename\_var) || ';';

END LOOP;

END LOOP;

END$$;

COMMIT;

ПРИЛОЖЕНИЕ И

-- 04\_insert\_initial\_data.sql

-- Скрипт для заполнения базы данных начальными данными (справочники, начальные остатки)

BEGIN;

-- 1. Заполнение основных справочников

INSERT INTO "КатегорияБлюда" ("Название") VALUES

('Роллы Классические'),

('Роллы Запеченные'),

('Роллы Фирменные'),

('Суши и Гунканы'),

('Наборы и Сеты'),

('Салаты'),

('Супы'),

('Горячие блюда'),

('Напитки'),

('Десерты'),

('Соусы и Добавки');

INSERT INTO "КатегорияПродукта" ("Название") VALUES

('Рыба свежая/охлажденная'),

('Рыба соленая/копченая'),

('Морепродукты (креветки, мидии, гребешки)'),

('Икра'),

('Рис и крупы'),

('Овощи свежие'),

('Фрукты свежие'),

('Зелень'),

('Сыры (сливочный, твердый)'),

('Молочные продукты (сливки, молоко)'),

('Соусы (соевый, унаги, спайси, терияки)'),

('Приправы и специи (васаби, имбирь)'),

('Нори и бумага для роллов'),

('Мука и панировка'),

('Масла растительные'),

('Бакалея (сахар, соль, уксус)'),

('Замороженные полуфабрикаты'),

('Напитки (ингредиенты для приготовления)'),

('Упаковка и расходники');

INSERT INTO "ЕдиницаИзмерения" ("Название", "Краткое\_название") VALUES

('Килограмм', 'кг'),

('Грамм', 'г'),

('Литр', 'л'),

('Миллилитр', 'мл'),

('Штука', 'шт.'),

('Порция', 'порц.'),

('Упаковка', 'упак.'),

('Лист', 'лст.');

INSERT INTO "Должность" ("Название", "Ставка\_в\_час") VALUES

('Шеф-повар', 500.00),

('Повар-сушист', 380.00),

('Помощник повара', 280.00),

('Кассир-администратор', 350.00),

('Менеджер зала', 420.00),

('Курьер', 250.00),

('Уборщик(ца)', 200.00);

INSERT INTO "ПричинаСписания" ("Название") VALUES

('Порча продукта'),

('Истек срок годности'),

('Бой/повреждение при транспортировке'),

('Ошибка приготовления'),

('Проработка нового меню/блюда'),

('Представительские расходы/дегустация'),

('Внутреннее потребление (питание персонала)'),

('Недостача при инвентаризации');

INSERT INTO "Касса" ("Номер\_кассы", "Расположение") VALUES

('K01-MAIN', 'Основной зал, центральная стойка'),

('K02-TAKEAWAY', 'Зона выдачи заказов на вынос');

INSERT INTO "Склад" ("Название", "Адрес") VALUES

('Основной продуктовый склад (Сухой)', 'Подсобное помещение №1'),

('Холодильный склад (-5/+5 C)', 'Холодильная камера №1'),

('Морозильный склад (-18C)', 'Морозильная камера №1'),

('Склад напитков и бара', 'Зона бара, помещение №2'),

('Склад хозтоваров и упаковки', 'Подсобное помещение №3');

-- 2. Заполнение зависимых справочников и основных сущностей

-- Блюда (ID категорий предполагаются по порядку вставки выше)

-- КатегорияБлюда: 1-Роллы Классические, 4-Суши и Гунканы, 9-Напитки, 11-Соусы и Добавки

INSERT INTO "Блюдо" ("ID\_категории\_блюда", "Название", "Цена\_продажи") VALUES

(1, 'Ролл Филадельфия', 480.00),

(1, 'Ролл Калифорния с крабом', 420.00),

(1, 'Ролл Канада с угрем', 550.00),

(4, 'Суши Лосось', 130.00),

(4, 'Суши Угорь', 160.00),

(4, 'Гункан с икрой лосося', 180.00),

(2, 'Ролл Темпура с креветкой', 450.00), -- Категория 2 - Запеченные

(9, 'Морс клюквенный (0.5л)', 150.00),

(9, 'Чай зеленый Сенча (чайник)', 200.00),

(11, 'Соевый соус (порция)', 30.00),

(11, 'Имбирь маринованный (порция)', 30.00),

(11, 'Васаби (порция)', 30.00);

-- Продукты (ID категорий и единиц измерения предполагаются по порядку вставки выше)

-- КатегорииПродукта: 1-Рыба св/охл, 5-Рис, 6-Овощи, 9-Сыры, 11-Соусы, 12-Приправы, 13-Нори

-- ЕдиницыИзмерения: 1-кг, 2-г, 4-мл, 5-шт, 8-лст

INSERT INTO "Продукт" ("ID\_категории\_продукта", "ID\_единицы\_измерения", "Название", "Срок\_годности\_дни") VALUES

(1, 1, 'Лосось (филе охлажденное)', 3),

(5, 1, 'Рис для суши "Нишики"', 365),

(9, 1, 'Сыр сливочный "Cream Cheese" 70%', 45),

(13, 5, 'Листы нори "Gold" (100 шт/упак)', 730), -- Ед.изм. Штука, но срок на упаковку

(6, 1, 'Огурец свежий длинноплодный', 10),

(6, 1, 'Авокадо "Хасс" спелое', 7),

(1, 1, 'Угорь Унаги (филе замороженное)', 180),

(3, 1, 'Креветка тигровая с/м (16/20)', 180),

(4, 2, 'Икра лосося (кета/горбуша)', 5), -- Ед.изм. Грамм

(11, 4, 'Соус соевый "Yamasa" (концентрат)', 365),

(12, 2, 'Имбирь маринованный розовый (корень)', 90),

(12, 2, 'Васаби (порошок)', 365),

(18, 4, 'Сироп клюквенный (для морса)', 180);

-- Сотрудники (ID должностей предполагаются по порядку вставки выше)

-- Должности: 1-Шеф, 2-Повар-сушист, 4-Кассир-админ

INSERT INTO "Сотрудник" ("ID\_должности", "Фамилия", "Имя", "Отчество", "Телефон", "Активен") VALUES

(1, 'Иванов', 'Сергей', 'Петрович', '+79011234567', TRUE),

(2, 'Петрова', 'Мария', 'Игоревна', '+79022345678', TRUE),

(2, 'Сидоров', 'Алексей', 'Иванович', '+79033456789', TRUE),

(4, 'Кузнецова', 'Ольга', 'Владимировна', '+79044567890', TRUE),

(6, 'Михайлов', 'Дмитрий', 'Андреевич', '+79055678901', TRUE); -- Курьер

-- 3. Начальные остатки на складах (пример)

-- Продукты ID: 1-Лосось, 2-Рис, 3-Сыр, 4-Нори

-- Склады ID: 1-Основной, 2-Холодильный

INSERT INTO "ОстатокПродуктаНаСкладе" ("ID\_склада", "ID\_продукта", "Количество") VALUES

(2, 1, 5.500), -- Лосось на холодильном складе

(1, 2, 25.000), -- Рис на основном

(2, 3, 10.250), -- Сыр на холодильном

(1, 4, 2.000); -- Нори (2 упаковки по 100 листов) на основном

-- 4. Состав блюд (рецептуры)

-- Блюда ID: 1-Филадельфия, 2-Калифорния, 4-Суши Лосось

-- Продукты ID: 1-Лосось, 2-Рис, 3-Сыр, 4-Нори, 5-Огурец, 6-Авокадо

-- Единицы измерения: г (0.001 кг), шт (для нори - 1 лист)

INSERT INTO "СоставБлюда" ("ID\_блюда", "ID\_продукта", "Количество") VALUES

-- Филадельфия (ID=1)

(1, 1, 0.080), -- Лосось 80г

(1, 2, 0.100), -- Рис 100г

(1, 3, 0.050), -- Сыр 50г

(1, 4, 1.000), -- Нори 1 лист (если ед.изм. продукта "Лист") или 0.0025 (если Нори в кг и 1 лист = 2.5г)

-- У нас Нори в Штуках, так что 1.000 корректно, если "Количество" в ед.изм. продукта.

-- Калифорния (ID=2)

(2, 2, 0.100), -- Рис 100г

(2, 4, 1.000), -- Нори 1 лист

(2, 6, 0.030), -- Авокадо 30г

-- (Нужно добавить крабовое мясо или его имитацию в продукты и в состав)

-- Суши Лосось (ID=4)

(4, 1, 0.035), -- Лосось 35г

(4, 2, 0.020); -- Рис 20г

COMMIT;

ПРИЛОЖЕНИЕ К

-- 02\_create\_tables.sql

-- Скрипт для создания структуры базы данных sushi\_shop\_db

BEGIN;

-- Удаление таблиц в обратном порядке зависимостей, если они существуют

DROP TABLE IF EXISTS "Смена" CASCADE;

DROP TABLE IF EXISTS "ОстатокПродуктаНаСкладе" CASCADE;

DROP TABLE IF EXISTS "СписаниеПродуктов" CASCADE;

DROP TABLE IF EXISTS "СписаниеБлюд" CASCADE;

DROP TABLE IF EXISTS "ПозицияПоступления" CASCADE;

DROP TABLE IF EXISTS "СоставБлюда" CASCADE;

DROP TABLE IF EXISTS "ПозицияЗаказа" CASCADE;

DROP TABLE IF EXISTS "Списание" CASCADE;

DROP TABLE IF EXISTS "Поступление" CASCADE;

DROP TABLE IF EXISTS "Заказ" CASCADE;

DROP TABLE IF EXISTS "Сотрудник" CASCADE;

DROP TABLE IF EXISTS "Продукт" CASCADE;

DROP TABLE IF EXISTS "Блюдо" CASCADE;

DROP TABLE IF EXISTS "Склад" CASCADE;

DROP TABLE IF EXISTS "Касса" CASCADE;

DROP TABLE IF EXISTS "ПричинаСписания" CASCADE;

DROP TABLE IF EXISTS "Должность" CASCADE;

DROP TABLE IF EXISTS "ЕдиницаИзмерения" CASCADE;

DROP TABLE IF EXISTS "КатегорияПродукта" CASCADE;

DROP TABLE IF EXISTS "КатегорияБлюда" CASCADE;

-- 1. Справочные таблицы

CREATE TABLE "КатегорияБлюда" (

"ID\_категории\_блюда" SERIAL PRIMARY KEY,

"Название" VARCHAR(100) NOT NULL UNIQUE

);

CREATE TABLE "КатегорияПродукта" (

"ID\_категории\_продукта" SERIAL PRIMARY KEY,

"Название" VARCHAR(100) NOT NULL UNIQUE

);

CREATE TABLE "ЕдиницаИзмерения" (

"ID\_единицы\_измерения" SERIAL PRIMARY KEY,

"Название" VARCHAR(50) NOT NULL UNIQUE,

"Краткое\_название" VARCHAR(10) NOT NULL UNIQUE

);

CREATE TABLE "Должность" (

"ID\_должности" SERIAL PRIMARY KEY,

"Название" VARCHAR(100) NOT NULL UNIQUE,

"Ставка\_в\_час" NUMERIC(10, 2) NOT NULL CHECK("Ставка\_в\_час" >= 0)

);

CREATE TABLE "ПричинаСписания" (

"ID\_причины\_списания" SERIAL PRIMARY KEY,

"Название" VARCHAR(255) NOT NULL UNIQUE

);

CREATE TABLE "Касса" (

"ID\_кассы" SERIAL PRIMARY KEY,

"Номер\_кассы" VARCHAR(50) NOT NULL UNIQUE,

"Расположение" VARCHAR(255)

);

CREATE TABLE "Склад" (

"ID\_склада" SERIAL PRIMARY KEY,

"Название" VARCHAR(100) NOT NULL UNIQUE,

"Адрес" VARCHAR(255)

);

-- 2. Таблицы, зависящие от справочников

CREATE TABLE "Блюдо" (

"ID\_блюда" SERIAL PRIMARY KEY,

"ID\_категории\_блюда" INTEGER NOT NULL REFERENCES "КатегорияБлюда"("ID\_категории\_блюда"),

"Название" VARCHAR(255) NOT NULL UNIQUE,

"Цена\_продажи" NUMERIC(10, 2) NOT NULL CHECK("Цена\_продажи" >= 0)

);

CREATE TABLE "Продукт" (

"ID\_продукта" SERIAL PRIMARY KEY,

"ID\_категории\_продукта" INTEGER NOT NULL REFERENCES "КатегорияПродукта"("ID\_категории\_продукта"),

"ID\_единицы\_измерения" INTEGER NOT NULL REFERENCES "ЕдиницаИзмерения"("ID\_единицы\_измерения"),

"Название" VARCHAR(255) NOT NULL UNIQUE,

"Срок\_годности\_дни" INTEGER CHECK("Срок\_годности\_дни" > 0)

);

CREATE TABLE "Сотрудник" (

"ID\_сотрудника" SERIAL PRIMARY KEY,

"ID\_должности" INTEGER NOT NULL REFERENCES "Должность"("ID\_должности"),

"Фамилия" VARCHAR(100) NOT NULL,

"Имя" VARCHAR(100) NOT NULL,

"Отчество" VARCHAR(100),

"Телефон" VARCHAR(20) UNIQUE,

"Дата\_приема" DATE NOT NULL DEFAULT CURRENT\_DATE,

"Активен" BOOLEAN NOT NULL DEFAULT TRUE

);

-- 3. Операционные таблицы

CREATE TABLE "Заказ" (

"ID\_заказа" SERIAL PRIMARY KEY,

"ID\_кассы" INTEGER NOT NULL REFERENCES "Касса"("ID\_кассы"),

"ID\_сотрудника\_оформившего" INTEGER NOT NULL REFERENCES "Сотрудник"("ID\_сотрудника"),

"Дата\_создания" TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

"Тип\_оплаты" VARCHAR(50),

"Общая\_сумма" NUMERIC(12, 2) DEFAULT 0 CHECK("Общая\_сумма" >= 0)

);

CREATE TABLE "Поступление" (

"ID\_поступления" SERIAL PRIMARY KEY,

"ID\_склада" INTEGER NOT NULL REFERENCES "Склад"("ID\_склада"),

"ID\_сотрудника\_принявшего" INTEGER NOT NULL REFERENCES "Сотрудник"("ID\_сотрудника"),

"Дата\_поступления" TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

"Номер\_накладной" VARCHAR(50)

);

CREATE TABLE "Списание" (

"ID\_списания" SERIAL PRIMARY KEY,

"ID\_склада" INTEGER NOT NULL REFERENCES "Склад"("ID\_склада"),

"ID\_сотрудника\_ответственного" INTEGER NOT NULL REFERENCES "Сотрудник"("ID\_сотрудника"),

"ID\_причины\_списания" INTEGER NOT NULL REFERENCES "ПричинаСписания"("ID\_причины\_списания"),

"Дата\_списания" TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

"Комментарий" TEXT

);

-- 4. Ассоциативные таблицы, таблицы деталей и наследуемые таблицы

CREATE TABLE "ПозицияЗаказа" (

"ID\_заказа" INTEGER NOT NULL REFERENCES "Заказ"("ID\_заказа") ON DELETE CASCADE,

"ID\_блюда" INTEGER NOT NULL REFERENCES "Блюдо"("ID\_блюда"),

"Количество" INTEGER NOT NULL CHECK("Количество" > 0),

"Цена\_на\_момент\_заказа" NUMERIC(10, 2) NOT NULL,

PRIMARY KEY ("ID\_заказа", "ID\_блюда")

);

CREATE TABLE "СоставБлюда" (

"ID\_блюда" INTEGER NOT NULL REFERENCES "Блюдо"("ID\_блюда") ON DELETE CASCADE,

"ID\_продукта" INTEGER NOT NULL REFERENCES "Продукт"("ID\_продукта") ON DELETE CASCADE,

"Количество" NUMERIC(10, 3) NOT NULL CHECK("Количество" > 0),

PRIMARY KEY ("ID\_блюда", "ID\_продукта")

);

CREATE TABLE "ПозицияПоступления" (

"ID\_поступления" INTEGER NOT NULL REFERENCES "Поступление"("ID\_поступления") ON DELETE CASCADE,

"ID\_продукта" INTEGER NOT NULL REFERENCES "Продукт"("ID\_продукта"),

"Количество" NUMERIC(10, 3) NOT NULL CHECK("Количество" > 0),

"Цена\_закупки" NUMERIC(10, 2) NOT NULL CHECK("Цена\_закупки" >= 0),

PRIMARY KEY ("ID\_поступления", "ID\_продукта")

);

CREATE TABLE "СписаниеБлюд" (

"ID\_блюда" INTEGER NOT NULL REFERENCES "Блюдо"("ID\_блюда"),

"Количество" INTEGER NOT NULL CHECK("Количество" > 0),

FOREIGN KEY ("ID\_списания") REFERENCES "Списание"("ID\_списания") ON DELETE CASCADE

) INHERITS ("Списание");

CREATE TABLE "СписаниеПродуктов" (

"ID\_продукта" INTEGER NOT NULL REFERENCES "Продукт"("ID\_продукта"),

"Количество" NUMERIC(10, 3) NOT NULL CHECK("Количество" > 0),

FOREIGN KEY ("ID\_списания") REFERENCES "Списание"("ID\_списания") ON DELETE CASCADE

) INHERITS ("Списание");

CREATE TABLE "ОстатокПродуктаНаСкладе" (

"ID\_склада" INTEGER NOT NULL REFERENCES "Склад"("ID\_склада") ON DELETE CASCADE,

"ID\_продукта" INTEGER NOT NULL REFERENCES "Продукт"("ID\_продукта") ON DELETE CASCADE,

"Количество" NUMERIC(10, 3) NOT NULL DEFAULT 0 CHECK("Количество" >= 0),

PRIMARY KEY ("ID\_склада", "ID\_продукта")

);

CREATE TABLE "Смена" (

"ID\_смены" SERIAL PRIMARY KEY,

"ID\_сотрудника" INTEGER NOT NULL REFERENCES "Сотрудник"("ID\_сотрудника"),

"ID\_склада" INTEGER REFERENCES "Склад"("ID\_склада"),

"Дата\_смены" DATE NOT NULL DEFAULT CURRENT\_DATE,

"Время\_начала" TIME NOT NULL,

"Время\_окончания" TIME,

CHECK ("Время\_окончания" IS NULL OR "Время\_окончания" > "Время\_начала")

);

-- 5. Создание индексов

CREATE INDEX IF NOT EXISTS "idx\_Блюдо\_ID\_категории\_блюда" ON "Блюдо"("ID\_категории\_блюда");

CREATE INDEX IF NOT EXISTS "idx\_Продукт\_ID\_категории\_продукта" ON "Продукт"("ID\_категории\_продукта");

CREATE INDEX IF NOT EXISTS "idx\_Продукт\_ID\_единицы\_измерения" ON "Продукт"("ID\_единицы\_измерения");

CREATE INDEX IF NOT EXISTS "idx\_Сотрудник\_ID\_должности" ON "Сотрудник"("ID\_должности");

CREATE INDEX IF NOT EXISTS "idx\_Заказ\_ID\_кассы" ON "Заказ"("ID\_кассы");

CREATE INDEX IF NOT EXISTS "idx\_Заказ\_ID\_сотрудника\_оформившего" ON "Заказ"("ID\_сотрудника\_оформившего");

CREATE INDEX IF NOT EXISTS "idx\_Заказ\_Дата\_создания" ON "Заказ"("Дата\_создания");

CREATE INDEX IF NOT EXISTS "idx\_Поступление\_ID\_склада" ON "Поступление"("ID\_склада");

CREATE INDEX IF NOT EXISTS "idx\_Поступление\_ID\_сотрудника\_приняв" ON "Поступление"("ID\_сотрудника\_принявшего");

CREATE INDEX IF NOT EXISTS "idx\_Поступление\_Дата\_поступления" ON "Поступление"("Дата\_поступления");

CREATE INDEX IF NOT EXISTS "idx\_Списание\_ID\_склада" ON "Списание"("ID\_склада");

CREATE INDEX IF NOT EXISTS "idx\_Списание\_ID\_сотрудника\_ответстве" ON "Списание"("ID\_сотрудника\_ответственного");

CREATE INDEX IF NOT EXISTS "idx\_Списание\_ID\_причины\_списания" ON "Списание"("ID\_причины\_списания");

CREATE INDEX IF NOT EXISTS "idx\_Списание\_Дата\_списания" ON "Списание"("Дата\_списания");

CREATE INDEX IF NOT EXISTS "idx\_ПозицияЗаказа\_ID\_блюда" ON "ПозицияЗаказа"("ID\_блюда");

CREATE INDEX IF NOT EXISTS "idx\_СоставБлюда\_ID\_продукта" ON "СоставБлюда"("ID\_продукта");

CREATE INDEX IF NOT EXISTS "idx\_ПозицияПоступления\_ID\_продукта" ON "ПозицияПоступления"("ID\_продукта");

CREATE UNIQUE INDEX IF NOT EXISTS "idx\_unique\_СписаниеБлюд\_ID\_списания" ON "СписаниеБлюд"("ID\_списания");

CREATE INDEX IF NOT EXISTS "idx\_СписаниеБлюд\_ID\_блюда" ON "СписаниеБлюд"("ID\_блюда");

CREATE UNIQUE INDEX IF NOT EXISTS "idx\_unique\_СписаниеПродуктов\_ID\_списания" ON "СписаниеПродуктов"("ID\_списания");

CREATE INDEX IF NOT EXISTS "idx\_СписаниеПродуктов\_ID\_продукта" ON "СписаниеПродуктов"("ID\_продукта");

CREATE INDEX IF NOT EXISTS "idx\_ОстатокПродуктаНаСкладе\_ID\_прод" ON "ОстатокПродуктаНаСкладе"("ID\_продукта");

CREATE INDEX IF NOT EXISTS "idx\_Смена\_ID\_сотрудника" ON "Смена"("ID\_сотрудника");

CREATE INDEX IF NOT EXISTS "idx\_Смена\_ID\_склада" ON "Смена"("ID\_склада");

CREATE INDEX IF NOT EXISTS "idx\_Смена\_Дата\_смены" ON "Смена"("Дата\_смены");

COMMIT;