СОДЕРЖАНИЕ

Введение5

1. Моделирование данных6
2. Платформа программного обеспечения13
3. Теоретическое обоснование разработки программного продукта17
4. Разработка функциональных требований18
5. Проектирование базы данных19
6. Описание функциональных возможностей программного продукта20

Заключение21

Список литературных источников22

Приложение А (обязательное) Листинг программного кода23

**ВВЕДЕНИЕ**

В современном цифровом мире, в котором доступность информации и скорость обработки данных становятся все более важными, внедрение эффективных баз данных становится ключевым компонентом многих веб-приложений. Базы данных служат фундаментальным инструментом для организации информации и обеспечения доступа к ней во всех сферах деятельности, начиная от предприятий и исследовательских организаций и заканчивая личным использованием.

Каршеринг – это современное и популярное явление в транспортной сфере, предоставляющее доступ к автомобилям на временной основе без необходимости их приобретения. Этот вид услуги становится все более популярным в современных городах, предлагая экономически выгодную и удобную альтернативу традиционным способам передвижения.

На сегодняшний день, рынок каршеринга переживает несколько характерных тенденций. Важно отметить устойчивый рост популярности этой услуги среди городских жителей, особенно среди молодого и активного населения, которое ищет более гибкие и экономически выгодные способы передвижения. Важной частью этой популярности является способность каршеринга уменьшать количество автомобилей на дорогах и, следовательно, снижать выбросы вредных веществ, что подчеркивает его экологические преимущества.

Целью данной курсовой работы является проектирование и реализация базы данных для веб-приложения каршеринга, которое уделяет особое внимание владельцам личных транспортных средств, а именно возможность пользователям предоставлять свои транспортные средства в аренду. Эта инновация открывает двери для более широкого круга участников, позволяя каждому владельцу транспортного средства стать потенциальным арендодателем и монетизировать свое имущество.

**1 МОДЕЛИРОВАНИЕ ДАННЫХ**

Моделирование данных  – это процесс создания визуального представления или чертежа, определяющего системы сбора и управления информацией в любой организации. Этот план или модель данных помогает различным заинтересованным сторонам, таким как аналитики данных, ученые и инженеры, создать единое представление о данных организации. Модель описывает, какие данные собирает компания, взаимосвязь между различными наборами данных и методы, которые будут использоваться для хранения и анализа данных.

Модели данных строятся на основе бизнес-потребностей. Правила и требования к модели данных определяются заранее на основе обратной связи с бизнесом, поэтому их можно включить в разработку новой системы или адаптировать к существующей.

**Типы моделей данных.**

Разработка баз данных и информационных систем начинается с высокого уровня абстракции и с каждым шагом становится все точнее и конкретнее. В зависимости от степени абстракции модели данных можно разделить на три категории. Процесс начинается с концептуальной модели, переходит к логической модели и завершается физической моделью.

Концептуальные модели данных (рисунок 1.1), которые также называются моделями предметной области, описывают общую картину: что будет содержать система, как она будет организована и какие бизнес-правила будут задействованы. Концептуальные модели обычно создаются в процессе сбора исходных требований к проекту. Как правило, они включают классы сущностей (вещи, которые бизнесу важно представить в модели данных), их характеристики и ограничения, отношения между сущностями, требования к безопасности и целостности данных. Любые обозначения обычно просты.

Заинтересованные стороны и аналитики обычно создают концептуальную модель. Это простое диаграммное представление, которое не следует формальным правилам моделирования данных. Важно то, что оно помогает как техническим, так и нетехническим заинтересованным сторонам разделить общее видение и согласовать цель, объем и дизайн проекта по работе с данными. Концептуальные модели выступают в качестве моста между бизнес-правилами и лежащей в их основе физической системой управления базами данных.

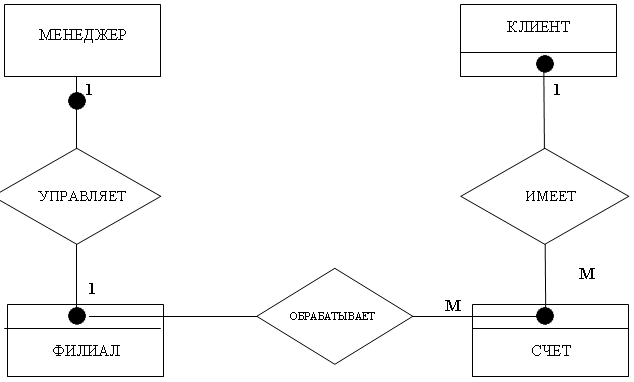


Рисунок 1.1 – Концептуальная модель данных

Логические модели данных отображают концептуальные классы данных на технических структурах данных. Они дают более подробную информацию о концепциях данных и их сложных отношениях, которые были определены в концептуальной модели данных, например:

* типы данных различных атрибутов (например, строка или число);
* взаимосвязи между объектами данных;
* первичные атрибуты или ключевые поля в данных.

Архитекторы данных и аналитики совместно работают над созданием логической модели. Они следуют одной из нескольких формальных систем моделирования данных для создания представления. Иногда гибкие команды могут пропустить этот шаг и перейти от концептуальных к физическим моделям напрямую. Однако эти модели полезны для проектирования больших баз данных, называемых хранилищами данных, и для проектирования систем автоматической отчетности.

Логические модели (рисунок 1.2) служат связующим звеном между концептуальной моделью данных и базовой технологией и языком баз данных, которые разработчики используют для создания базы данных. Однако они не зависят от технологии, и вы можете реализовать их на любом языке баз данных. Инженеры по данным и заинтересованные стороны обычно принимают технологические решения после создания логической модели данных.

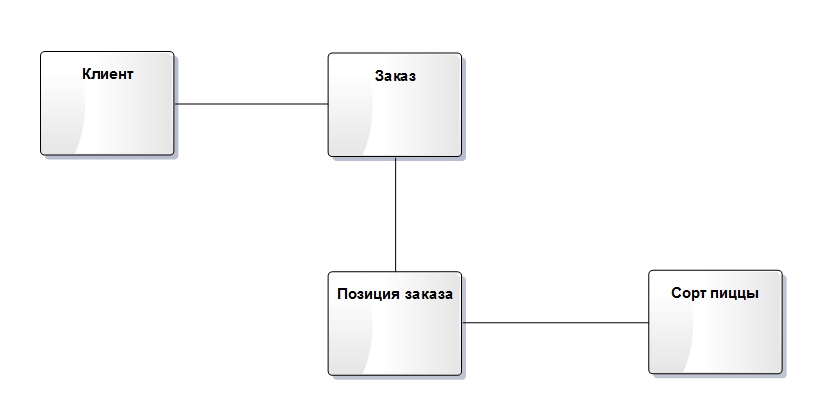


Рисунок 1.2 – Логическая модель данных

Физические модели данных представляют схему того, как данные будут храниться в базе. По сути, это наименее абстрактные из всех моделей. Они предлагают окончательный дизайн, который может быть реализован как реляционная база данных, включающая ассоциативные таблицы, которые иллюстрируют отношения между сущностями, а также первичные и внешние ключи для связи данных. Производительность базы данных, стратегия индексации, физическое хранилище и денормализация – важные параметры физической модели. Физическая модель данных выступает в качестве моста между логической моделью данных и конечной технологической реализацией.

Физическая модель данных (рисунок 1.3) зависит от конкретной системы управления базами данных, фактически являясь отображением системного каталога. В физической модели содержится информация обо всех объектах базы данных. Поскольку стандартов на объекты базах данных не существует, например, нет стандарта на типы данных, физическая модель зависит от конкретной реализации систему управления базами данных. Следовательно, одной и той же логической модели могут соответствовать несколько разных физических моделей. Если в логической модели не имеет значения, какой конкретно тип данных имеет атрибут, то в физической модели важно описать всю информацию о конкретных физических объектах.

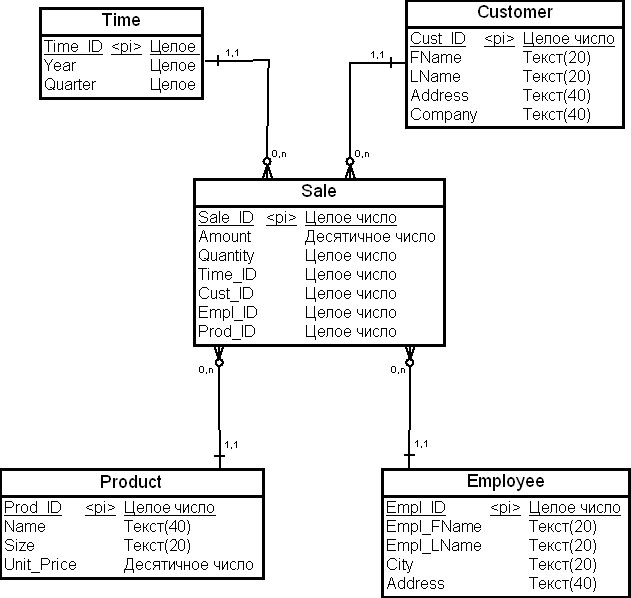


Рисунок 1.2 – Физическая модель данных

**Методы моделирования данных.**

Методы моделирования данных представляют собой различные подходы, которые могут быть использованы для создания различных моделей данных. Эти методы развивались со временем, в ответ на изменения в концепциях баз данных и управления данными.

Иерархические модели данных описывают отношения "один ко многим" в структуре дерева. В такой модели каждая запись имеет свой корневой или родительский элемент, который связан с одной или несколькими дочерними таблицами. Эта модель была впервые представлена в IBM Information Management System в 1966 году и быстро нашла применение, особенно в банковской сфере. Несмотря на то, что этот метод менее эффективен по сравнению с более современными моделями баз данных, он все еще используется в системах с расширяемым языком разметки (XML) и географических информационных системах (ГИС).

Иерархическое моделирование данных позволяет представлять отношения между различными элементами данных в форме дерева. Оно характеризуется отношениями "один ко многим", где родительские или корневые элементы данных могут иметь несколько дочерних элементов.

С течением времени иерархическое моделирование данных эволюционировало в графовые модели данных, которые представляют отношения между данными таким образом, что сущности рассматриваются одинаково, и они могут быть связаны друг с другом отношениями "один ко многим" или "многие ко многим", без понятия "родительский" или "дочерний".

Реляционные модели данных до сих пор широко применяются в реляционных базах данных, особенно в корпоративных вычислениях. Реляционное моделирование не требует глубокого понимания физических характеристик используемого хранилища данных, так как данные сегментируются с использованием таблиц, что упрощает структуру базы данных.

Моделирование данных "сущность-связь" (ER) использует формальные диаграммы для отображения отношений между сущностями в базе данных. Архитекторы данных используют несколько инструментов ER-моделирования для представления данных.

В объектно-ориентированном программировании для хранения данных используются структуры данных, называемые объектами. Эти объекты данных представляют программные абстракции объектов из реального мира. Например, в объектно-ориентированной модели данных автосалон будет содержать объекты данных, такие как "Клиенты", с атрибутами, такими как имя, адрес и номер телефона. Данные о клиентах хранятся таким образом, что каждый реальный клиент представлен в виде объекта данных о клиенте.

В современных корпоративных вычислениях используются технологии хранилищ данных для хранения больших объемов данных, предназначенных для аналитики. Для высокоскоростного хранения и извлечения данных из хранилищ используются проекты многомерного моделирования данных. Многомерные модели используют дублирование данных или избыточные данные с целью повышения производительности, при этом не уделяя особого внимания экономии места для хранения данных.

**Процесс моделирования данных.**

Моделирование данных начинается с договоренности о том, какие символы используются для представления данных, как размещаются модели и как передаются бизнес-требования. Это формализованный рабочий процесс, включающий ряд задач, которые должны выполняться итеративно. Сам процесс обычно выглядят так:

1. Определите сущности. На этом этапе идентифицируем объекты, события или концепции, представленные в наборе данных, который необходимо смоделировать. Каждая сущность должна быть целостной и логически отделенной от всех остальных.
2. Определите ключевые свойства каждой сущности. Каждый тип сущности можно отличить от всех остальных, поскольку он имеет одно или несколько уникальных свойств, называемых атрибутами. Например, сущность «клиент» может обладать такими атрибутами, как имя, фамилия, номер телефона и т.д. Сущность «адрес» может включать название и номер улицы, город, страну и почтовый индекс.
3. Определите связи между сущностями. Самый ранний черновик модели данных будет определять характер отношений, которые каждая сущность имеет с другими. В приведенном выше примере каждый клиент «живет по» адресу. Если бы эта модель была расширена за счет включения сущности «заказы», каждый заказ также был бы отправлен на адрес. Эти отношения обычно документируются с помощью унифицированного языка моделирования (UML).
4. Полностью сопоставьте атрибуты с сущностями. Это гарантирует, что модель отражает то, как бизнес будет использовать данные. Широко используются несколько формальных шаблонов (паттернов) моделирования данных. Объектно-ориентированные разработчики часто применяют шаблоны для анализа или шаблоны проектирования, в то время как заинтересованные стороны из других областей бизнеса могут обратиться к другим паттернам.
5. Назначьте ключи по мере необходимости и определите степень нормализации. Нормализация – это метод организации моделей данных, в которых числовые идентификаторы (ключи) назначаются группам данных для установления связей между ними без повторения данных. Например, если каждому клиенту назначен ключ, этот ключ можно связать как с его адресом, так и с историей заказов, без необходимости повторять эту информацию в таблице с именами клиентов. Нормализация помогает уменьшить объем дискового пространства, необходимого для базы данных, но может сказываться на производительности запросов.
6. Завершите и проверьте модель данных. Моделирование данных – это итеративный процесс, который следует повторять и совершенствовать под потребности бизнеса.

**Системы управления базами данных.**

Система управления базами данных (СУБД) – это программное обеспечение, предназначенное для хранения, извлечения и управления данными. Наиболее распространенной СУБД в системе баз данных предприятия является СУРБД. Полная форма СУБД – это система управления реляционными базами данных.

Согласно реляционной модели Э. Ф. Кодда, СУБД позволяет пользователям создавать, обновлять, управлять реляционной базой данных и взаимодействовать с ней, позволяя хранить данные в табличной форме.

Система управления реляционными базами данных организует данные в таблицах, которые могут быть связаны внутри в зависимости от общих данных. Это позволяет пользователю легко получить одну или несколько таблиц с помощью всего одного запроса. С другой стороны, в плоском файле данные хранятся в единой табличной структуре, что менее эффективно и требует больше места и памяти.

Наиболее коммерчески доступной и общекорпоративной системой управления базами данных или системой управления реляционными базами данных, используемой сегодня, является язык структурированных запросов (база данных SQL) для доступа к базе данных.

Другие широко используемые системы управления реляционными базами данных в компаниях включают базу данных Oracle, MySQL, PostgreSQL (реляционная база данных с открытым исходным кодом) и Microsoft SQL Server. Структуры РСУБД обычно используются для выполнения четырех основных операций: CRUD (создание, чтение, обновление и удаление), которые имеют решающее значение для поддержки согласованного управления данными.

**2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**Характеристика программного обеспечения.**

Платформа – это комплекс аппаратных и программных средств, на котором функционирует программное обеспечение пользователя ЭВМ. Основа аппаратной платформы (hardware-платформы) – процессор. Тип процессора определяет архитектуру аппаратных средств – аппаратную платформу, т. е. тип и характеристики компьютера.

Системное программное обеспечение – это «программная оболочка» аппаратных средств, предназначенная для отделения остальных программ от непосредственного взаимодействия с оборудованием и организации процесса обработки информации в компьютере. Прикладное программное обеспечение предназначено для решения определенных задач пользователя. К системному программному обеспечению относятся такие типы программ, как операционные системы, различные сервисные средства, функционально дополняющие возможности операционных систем, инструментальные средства (системы управления базами данных, программирования, оболочки экспертных систем).

Основная компонента системного программного обеспечения – операционная система выполняет следующие функции:

1. Организация работы компьютера, при которой возможно одновременное выполнение нескольких программ пользователя.
2. Организация хранения программ и данных пользователя на носителях информации и, возможно, санкционирование доступа к этой информации.
3. Обеспечение взаимодействия с пользователем на основе графического интерфейса.
4. Обеспечение сетевых возможностей, т. е. возможности доступа к информации, хранимой в памяти другого компьютера локальной или глобальной сети.

В курсовом проекте платформой программного обеспечения будет использоваться Windows.

**Операционная система Windows.**

Свойства операционной системы Windows:

* + является 32-разрядной;
  + поддерживает вытесняющую многозадачность;
  + работает на разных аппаратных архитектурах и обладает способностью к сравнительно легкому переносу на новые аппаратные архитектуры;
  + поддерживает работу с виртуальной памятью;
  + является полностью реентерабельной.

Компьютерная программа в целом или её отдельная процедура называется реентерабельной (от англ. reentrant – повторно входимый), если она разработана таким образом, что одна и та же копия инструкций программы в памяти может быть совместно использована несколькими пользователями или процессами. При этом второй пользователь может вызвать реентерабельный код до того, как с ним завершит работу первый пользователь и это как минимум не должно привести к ошибке, а в лучшем случае не должно вызвать потери вычислений (то есть не должно появиться необходимости выполнять уже выполненные фрагменты кода).

Реентерабельный код имеет следующие преимущества:

* + хорошо масштабируется с мультипроцессорной обработкой;
  + является распределенной вычислительной платформой;
  + защищена как от внутренних сбоев, так и от внешних;
  + пользовательский интерфейс совместим с предыдущими версиями;
  + обладает высокой производительностью независимо от платформы;
  + обеспечивает простоту адаптации за счет поддержки Unicode;
  + поддерживает многопоточность и объектную модель.

**Общее описание структуры системы.**

Архитектура ОС Windows претерпела ряд изменений в процессе эволюции. Первые версии системы имели микроядерный дизайн, основанный на микроядре Mach, которое было разработано в университете КарнегиМеллона. Архитектура более поздних версий системы микроядерной уже не является. Причина заключается в постепенном преодолении основного недостатка микроядерных архитектур –дополнительных накладных расходов, связанных с передачей сообщений.

По мнению специалистов Microsoft, чисто микроядерный дизайн коммерчески невыгоден, поскольку неэффективен. Поэтому большой объем системного кода, в первую очередь управление системными вызовами и экранная графика, был перемещен из адресного пространства пользователя в пространство ядра и работает в привилегированном режиме.

В результате в ядре ОС Windows переплетены элементы микроядерной архитектуры и элементы монолитного ядра (комбинированная система).

Сегодня микроядро ОС Windows слишком велико (более 1 Мб), чтобы носить приставку «микро». Основные компоненты ядра Windows NT располагаются в вытесняемой памяти и взаимодействуют друг с другом путем передачи сообщений, как и положено в микроядерных операционных системах. В тоже время все компоненты ядра работают в одном адресном пространстве и активно используют общие структуры данных, что свойственно операционным системам с монолитным ядром.

Высокая модульность и гибкость первых версий Windows NT позволила успешно перенести систему на такие отличные от Intel платформы, как Alpha (корпорация DEC), Power PC (IBM) и MIPS (Silicon Graphic). Более поздние версии ограничиваются поддержкой архитектуры Intel x86. Пример упрощенной архитектуры Windows (рисунок 2.1).

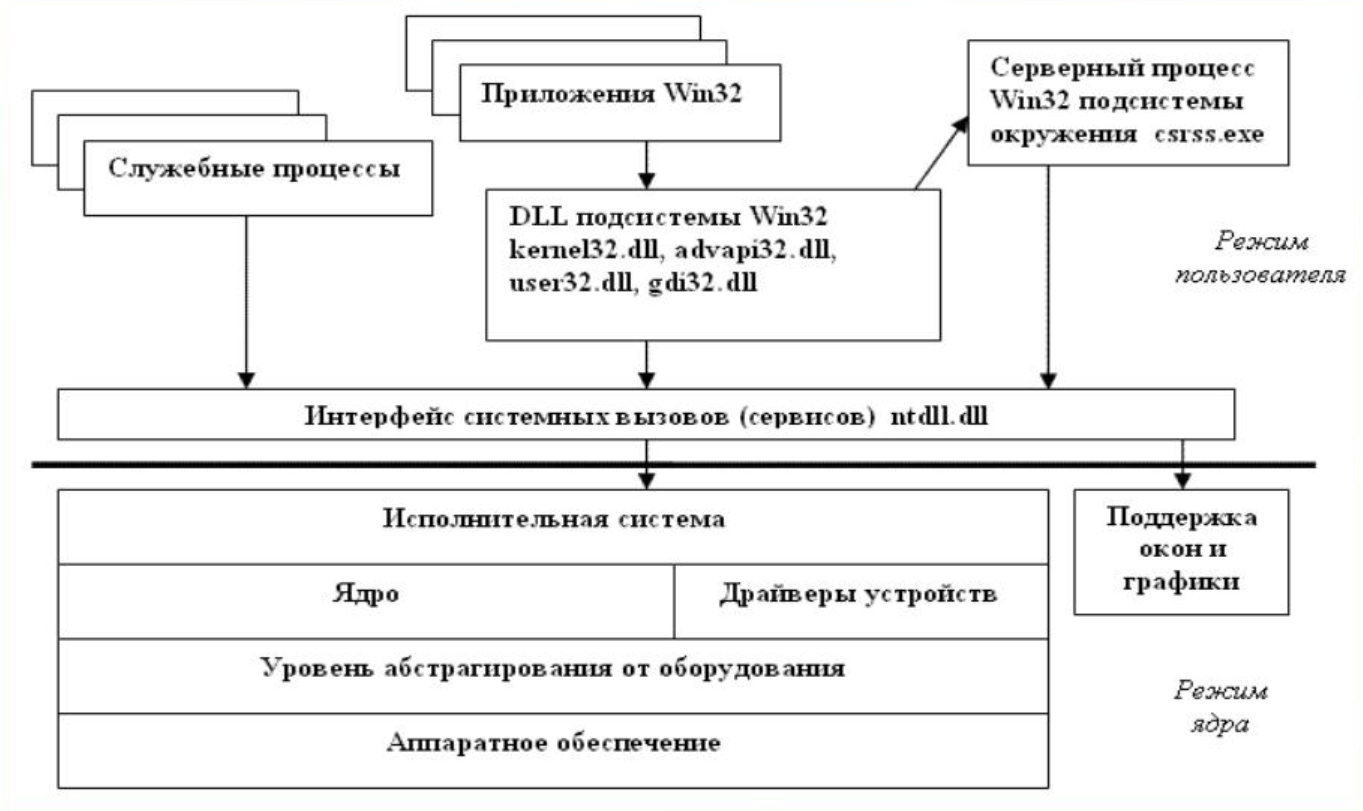


Рисунок 2.1 – Упрощенная архитектурная схема ОС Windows

ОС Windows состоит из компонентов, работающих в режиме ядра и пользователя. Несмотря на миграцию системы в сторону монолитного ядра она сохранила некоторую структуру. В схеме, представленной на рисунке 1.10, отчетливо просматриваются несколько функциональных уровней, каждый из которых пользуется сервисами более низкого уровня.

Задача уровня абстрагирования от оборудования (hardware abstraction layer, HAL) – скрыть аппаратные различия аппаратных архитектур для потенциального переноса системы с одной платформы на другую. HAL предоставляет вышележащим уровням аппаратные устройства в абстрактном виде, свободном от индивидуальных особенностей. Это позволяет изолировать ядро, драйверы и исполнительную систему ОС Windows от специфики оборудования (например, от различий между материнскими платами).

Ядром обычно называют все компоненты ОС, работающие в привилегированном режиме работы процессора или в режиме ядра. Корпорация Microsoft называет ядром (kernel) компонент, находящийся в невыгружаемой памяти и содержащий низкоуровневые функции операционной системы, такие, как диспетчеризация прерываний и исключений, планирование потоков и другие. Оно также предоставляет набор процедур и базовых объектов, применяемых компонентами высших уровней.

Ядро и HAL являются аппаратно-зависимыми и написаны на языках Си и ассемблера.

**3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ**

**ПРОГРАММНОГО ПРОДУКТА**

**4 РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ** **ТРЕБОАВНИЙ**

**5 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ**

**6 ОПИСАНИЕ ФУНКЦИОНАЛЬНЫХ**

**ВОЗМОЖНОСТЕЙ ПРОГРАММНОГО ПРОДУКТА**

**ЗАКЛЮЧЕНИЕ**

**СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ**

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг программного кода**