ВВЕДЕНИЕ

**Актуальность темы.** В настоящее время достаточно широко распространены услуги информационно-справочного обслуживания, которые нацелены на хранение и обработку каких-либо сведений. Сложно представить успешную компанию, не использующую никаких автоматизированных систем, нацеленных на хранение информации, циркулирующей внутри нее. В эффективной работе компании большую роль играет минимизация времени, затрачиваемого на обработку данных, таких, как например статистика, выполнение тех или иных услуг и так далее. Для этого применяются базы данных, которые, по сути, являются одними из важнейших частей информационных систем, предназначенных для хранения и обработки информации. Для управления же этими данными и обеспечения эффективного доступа к ним, а также для упрощения работы пользователей ещё к началу 1980-х годов были созданы системы управления базами данных и в настоящий момент используются около двадцати таких систем.

**Цель и задачи работы.** Целью данного курсового проекта является разработка программы, которая будет обеспечивать компанию средствами хранения и обработки циркулирующей в ней информации.

Для достижения поставленной цели необходимо решить следующие задачи:

провести анализ предметной области, для которой разрабатывается приложение (выполнить концептуальное моделирование);

разработать логическую модель будущей системы;

разработать физическую модель на основе логической модели;

создать базу данных с использованием выбранной СУБД;

провести тестирование разработанной базы данных посредством создания и выполнения запросов к ней;

разработать клиентское приложение для обеспечения возможности удобной обработки информации.

**Предмет и объект исследования.** Объектом исследования настоящей работы является процесс создания базы данных, предметом исследования служит база данных «Организация пассажирских морских перевозок».

**Структура работы.** Данная работа состоит из пояснительной записки, которая включает в себя введение, три раздела, заключение, список литературы, приложения и программного модуля на электронном носителе.

В первом разделе заключается аналитическая часть проектирования, в рамках которой формулируется постановка задачи и освещается предметная область.

Во втором разделе содержится описание процесса разработки логической модели базы данных. В рамках этого раздела строится три типа логических моделей: в нотации Чена, модель, основанная на ключах и модель в нотации IDEF1X.

Третий раздел освещает процесс построения физической модели, включая обоснования выбора аппаратно-программной платформы, реализацию базы данных с использованием взятого ПО и тестирование разработанной системы.

В четвертом разделе представлено тестирование разработанной системы в целом и описание работы системы

1 АНАЛИЗ ПРОЦЕССА РАБОТЫ МОРСКОЙ КОМПАНИИ ПО ПЕРЕВОЗКЕ ПАРССАЖИРОВ

1.1 Постановка задачи

Требуется разработать приложение (базу данных), которое будет обеспечивать хранение и обработку информации, циркулирующей в компании-заказчике.

Основные требования к разрабатываемому приложению:

возможность ведения данных о суднах, рейсах;

возможность фиксации и редактирования поступающих заказов билетов;

возможность отслеживания статуса заказа;

возможность выборки для пользователя имеющихся рейсов на определенную дату;

возможность выбора пассажиром места на палубе судна;

возможность менеджером выписки забронированных билетов пользователей;

удобный пользовательский интерфейс, понятный для пользователей программы.

Основные ограничения:

один пассажир на одну дату может забронировать только 1 билет;

судно определяется идентификатором, выданным в акватории;

заказ может иметь следующие состояния: забронировано, выписано, отменено;

все заказы привязываются к пользователю. При создании пользователя используется введенные номер телефона и email.

1.2 Анализ предметной области

База данных разрабатывается для предметной области «Работа компании морского порта». Заказчиком является Морской порт, владеющее несколькими суднами и занимающийся перевозкой пассажиров.

База данных предназначена для хранения и автоматизации обработки информации о заказах, поступающих на предприятие. Клиенты оставляют запросы через приложение.

Разрабатываемая база данных должна обеспечивать возможность фиксации заказов, а также наблюдение за выполнением заказов.

Большое внимание в базе данных должно уделяться заказам: времени поступления заказа, времени завершения заказа, действительность заказа.

Таким образом, можно утверждать, что цель создания базы данных – обеспечение предприятия удобной системой для регистрации поступивших заказов, отслеживания и фиксирования этапов их выполнения и ведения статистики предприятия о заказах и выполненных рейсах, проданных билетах и перевезенных пассажирах.

На данном предприятии можно выделить две основные роли пользователей, взаимодействующих с приложением: «Модератор» и «Клиент». Каждый из профилей имеет собственную цель использования разрабатываемого приложения. Человек, входящий в первый профиль, может иметь своими целями обработку заказов и редактирование базы, человек второго профиля – оформлять заказы.

1.3 Построение концептуальной модели

На основе анализа предметной области построим внешнюю модель системы (рисунок 1.1), выделив объекты, которые, в целом, описывают предметную область и задают начальную точку для дальнейшего развития системы.



Рисунок 1.1 – Внешняя модель разрабатываемой системы

Таким образом, сформированы 6 основных сущностей:

Судно;

Рейс;

Пассажир;

Билет;

Заказ;

Пользователь.

С сущности «Заказ» начинается весь поток информации, и без нее система не будет отвечать ИПП.

Сущность «Пассажир» хранит всю информацию о пассажире, отправляющемся в рейс.

Сущность «Билет» является связывающей сущностью для пассажира, пейса и заказа. В ней хранятся данные о том какой пассажир будет перевезен на выбранном рейсе.

Сущность «Клиент» требуется для регистрации Заказа, хранит идентификатор клиента и его номер телефона.

Сущность «Судно» располагает информации о судне, перевозящим пассажиров.

Сущность «Рейс» хранит данные о рейсе, в который отправятся пассажиры.

На основе анализа внешней модели, произведенного ранее, расширим её путем введения дополнительных сущностей «Заказ-Билет» и «Рейс-Билет». Расширенная внешняя модель представлена на рисунке 1.2.



Рисунок 1.2 – Расширенная внешняя модель

Из диаграммы на рисунке 1.2 видно, что сущность «Билет» имеет двух предков, что соответствует определению сетевой структуры. Для того чтобы преобразовать структуру к древовидной прибегнем к введению избыточности в виде дублирования таблицы «Билет» и распространению ключей. Вид модифицированной расширенной внешней модели можно увидеть на рисунке 1.3.



Рисунок 1.3 – Расширенная внешняя модель без элементов сетевой структуры

Выводы раздела 1

В ходе анализа предметной области «Работа морского порта» первоначально были установлены ограничения к разрабатываемой системе, которые будут необходимы при дальнейшей разработке. Также было определено основное назначение системы, её некоторые характерные особенности (в частности, наличие в ней темпоральных данных) и выделены основные роли пользователей, которые будут эксплуатировать разрабатываемое приложение. В рамках концептуального моделирования была построена внешняя модель базы данных, первоначальный вид которой предполагал наличие множественных связей между сущностями, что, в свою очередь, указывало на сложную сетевую структуру. С целью приведения базы данных к иерархическому виду были проведены следующие действия:

к шести базовым таблицам были добавлены дополнительные, после чего структура приняла простой сетевой вид (за счет наличия сущности с тремя предками);

была введена избыточность путем дублирования таблицы, имеющей двух «родителей».

Таким образом, можно заключить, что по итогу завершения первого раздела была построена концептуальная модель базы данных, на основе которой можно продолжить проектирование, спускаясь на всё более низкие уровни.

2 РАЗРАБОТКА ЛОГИЧЕСКОЙ МОДЕЛИ ДАННЫХ

2.1 Построение диаграммы «сущность-связь» в нотации П.Чена

Для начала необходимо проанализировать внешнюю модель (рисунок 1.3) и на её основе перейти к логической модели. Вид ERD-диаграммы представлен на рисунке А.1 приложения А.

Заказ, совмещает себе основные сущности, которые оперируются в базе данных.

Также подразумевается, что Заказ имеет и «физическую интерпретацию», то есть этап «материализации» или, проще говоря, его реализации.

В итоге, можно сказать, что заказ имеет сложную структуру: предполагается, что для выполнения заказа требуется информация о клиенте, билете, рейсе и пассажире. По сути, сущность «Билет» может рассматриваться как развязка связи «многие ко многим» между объектами «Заказ» и «Пассажир» и «Рейс».

2.2 Нормализация БД до первой нормальной формы

На данном этапе структура базы данных является древовидной, поэтому, полагаясь на данное условие, можно начать анализ на её нахождение в нормальных формах.

На основе диаграммы в нотации Чена, представленной на рисунке А.1, построим модель основанную на ключах (рисунок А.2), на которой можно увидеть то, как происходит «стекание» ключей, т.е., иными словами, их распространение по дочерним сущностям, начиная с родительской (которой, как это видно, является сущность «Заказ»).

Первая нормальная форма предполагает наличие первичных ключей и отсутствие неатомарных полей. Видим, что на данный момент второе условие нарушено, так как в некоторых отношениях имеются неатомарные атрибуты: в сущностях «Пассажир» есть поле «ФИО»; «Пассажир», «Заказ» и «Рейс» имеют неатомарное поле «Билет». Необходимо устранить эту проблему для дальнейшего анализа, поэтому выделим отдельные сущности («Заказ-Билет» и «Рей-Билет»), которые будут являться справочными. Также необходимо добавить таблицы «развязки», что, собственно говоря, и позволит избавиться от неатомарности. Модифицированный вариант диаграммы, основанной на ключах, отображен на рисунке А.3. Как видим, теперь выполняются оба условия, поэтому можно утверждать, что БД находится в первой нормальной форме. Однако сущности «Рейс-Билет» и «Заказ-Билет» можно упразднить, так как эти отношения является развязками «многие ко многим».

3 РАЗРАБОТКА ФИЗИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ

3.1 Выбор аппаратной и программной платформы для реализации БД

Для того, чтобы сформировать некоторое множество возможных вариантов СУБД, следует определить в какой мере существующие системы удовлетворяют основным требованиям разрабатываемого приложения. Для того, чтобы ограничить круг возможных систем, будем опираться на следующие критерии отбора:

моделирование данных: используемая модель данных, предусмотренные типы данных, реализация языка запросов;

особенности архитектуры и функциональные возможности: возможные принципы организации архитектуры, масштабируемость;

особенности разработки приложений: средства проектирования, поддерживаемые языки программирования;

надежность: резервное копирование, восстановление после сбоев, откат изменений, многоуровневая система защиты;

требования к рабочей среде: поддерживаемые аппаратные платформы; поддерживаемые операционные системы.

Исходя из перечисленных выше критериев, определим требования к СУБД, которая будет использоваться в дальнейшем.

**1. Моделирование данных.** Из рисунка А.6 можно увидеть, что модель БД – иерархическая. Однако на полученной диаграмме есть две сущности, которые играют в системе одну и ту же роль. В физической реализации планируется «слить» эти таблицы воедино, а в следствие этого возникает ситуация, когда сущность имеет двух предков. Таким образом, модель БД будет являться иерархической, но с элементами сетевой структуры, поэтому одно из ключевых требований к СУБД – возможность поддержания такой структуры базы данных.

Требования к типам данных сводятся к тому, что в БД предполагается наличие темпоральных свойств и, как следствие, темпоральных данных. В остальном достаточно использования базовых типов данных.

Также требуется, чтобы СУБД была совместима со стандартным языком доступа к данным SQL-92, и при этом желательно расширение данного стандарта в виде средств для работы с темпоральными данными (без готового расширения, вероятно, будет требоваться темпоральная надстройка).

**2. Особенности архитектуры и функциональные возможности.** Предполагается использование локальной СУБД с архитектурой «клиент-сервер». Схема данной архитектуры отображена на рисунке 3.1.



Рисунок 3.1 – Схема клиент-серверной архитектуры

Базовый принцип такой архитектуры – централизация хранения и обработки данных. Для его реализации используется так называемый сервер баз данных, выполненный как приложение или сервис операционной системы, и только он может реально манипулировать файлами. В качестве рабочего места пользователя может быть использован обычный персональный компьютер, что позволяет не отказываться от привычной рабочей среды.

Таким образом, в простейшем случае клиент-серверная информационная система состоит из двух основных компонентов:

1) сервера баз данных, управляющего данными и выполняющего запросы клиентских приложений;

2) клиентских приложений, предоставляющих интерфейс пользователя и посылающих запросы к серверу.

Информационные системы, использующие такую архитектур, обладают серьезными преимуществами:

являются дешевым решением проблемы коллективного доступа к базам данных в локальной сети;

снижение сетевого трафика при выполнении запросов за счет того, что в клиентское приложение передается только результат запроса ( в этом случае сетевой трафик не зависит ни от ни от числа записей в таблицах, ни от способа реализации поиска нужных записей);

на компьютерах-клиентах освобождается значительный объем дискового пространства за счет отсутствия нужды хранения большого количества данных;

сниженные требования к оперативной памяти клиентских компьютеров за счет того, что весь «груз» файловых операций лежит, в основном, на сервере;

высокую степень безопасности за счет возможности управления пользовательскими привилегиями и правами доступа к различным объектам базы данных;

 возможность параллельной обработки данных.

Приложение разрабатывается для достаточно серьезного предприятия, в котором, вероятно, будет проявляться рост информации в виде увеличения числа пользователей, объема хранимых данных и, безусловно, объема обрабатываемой информации. Поэтому, при выборе СУБД необходимо учитывать, сможет ли данная система соответствовать росту информационной системы.

**3. Особенности разработки приложений.** При выборе СУБД большой акцент будет сделан на предоставляемые средства проектирования: требуется, чтобы в системе поддерживалась автоматическая генерация скриптов, автоматическая генерацию внешних и первичных ключей, автоматическая генерация триггеров, также желательна возможность осуществления не только проектирования данных, но и структурного моделирования.

Также требуется, чтобы в СУБД поддерживался объектно-ориентированный язык программирования C#.

**4. Надежность.** Требуется обеспечение высокой надежности СУБД в виде возможности восстановления поврежденных или утерянных данных за счет реализации механизма резервного копирования. Данное требование возникает из уровня организации, для которой разрабатывается приложение: потеря некоторых данных может привести к достаточно серьезным финансовым убыткам. Также требуется реализация в СУБД механизма отката изменений, который позволил бы аннулировать недоведенные до конца транзакции (к примеру, в случае программных или аппаратных сбоев).

В СУБД также должна быть обеспечена многоуровневая система защиты, так как на предприятии заказчика имеются профили категорий пользователей, и, как следствие, у рабочих есть те или иные обязанности (в зависимости от их профилей). В соответствии с этим, требуется обеспечить непосредственную идентификацию пользователей при входе в систему.

**5. Требования к рабочей среде.** Основное требование в данном направлении – возможность запуска СУБД под ОС Windows.

Рассмотрим СУБД MySQL и PostgreSQL. MySQL существует достаточно давно и зарекомендовала себя как отличное решение, PostgreSQL пришла на рынок приблизительно в то же самое время, но предоставляет достаточно много интересных функций и возможностей, благодаря чему стремительно набирает популярность.

**Стандарт SQL.** Для того, чтобы сохранить MySQL достаточно простым для использования, разработчики выбрали путь развития, в котором поддерживаются далеко не все новые возможности стандарта SQL.

 PostgreSQL разрабатывается командой энтузиастов, которые пытаются максимально соответствовать стандарту SQL. Однако это идет в ущерб простоте: данная СУБД достаточно сложная.

**Типы данных.** Типов данных в PostgreSQL гораздо больше, они более разнообразны, есть свои типы полей для определенных видов данных (в том числе темпоральных), которых нет MySQL. В данном аспекте отличие СУБД PostgreSQL от MySQL очевидно: первая из них в полной мере удовлетворяет требованиям к типам данных.

**Управление учетными записями пользователей.** Как MySQL, так и Postgres позволяют системным администраторам создавать пользователей, а также предоставлять права пользователям или лишать их прав на нескольких уровнях привилегий.

**Архитектура.** Архитектура MySQL является клиент-серверной, однако её организация очень сильно отличается от архитектур других систем. MySQL не универсальна, но обладает достаточной гибкостью, чтобы можно было работать в среде разрабатываемого приложения.

**PostgreSQL также** же использует клиент-серверную модель, но при этом является унифицированным сервером баз данных.

Подводя итоги, можно сказать, что и MySQL, и PostgreSQL вполне могли бы подойти для реализации разрабатываемой системы. MySQL, по большей части, привлекает своей простотой, которой не особо может похвастаться Postgres. Поэтому выбор в данном случае падает на СУБД MySQL, которая в полной мере удовлетворяет требованиям, описанным ранее.

На рисунке 3.2 представлена архитектура PostgreSQL.

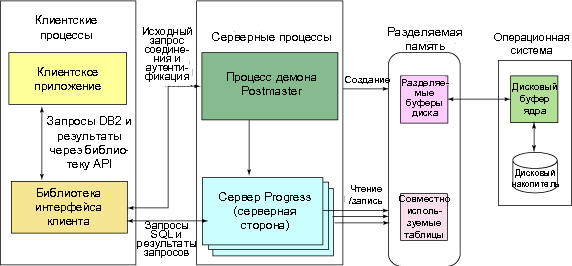


Рисунок 3.2 – Архитектура СУБД MySQL

 MySQL использует простую клиент-серверную модель «процесс/пользователь», в которой к одному серверному процессу присоединяется один процесс-клиент.

Работающий PostgreSQL состоит из следующих процессов:

контролирующий процесс (postmaster);

клиентское приложение пользователя (например, программа pgAdmin);

серверная часть базы данных (процесс postgres).

Главный процесс «слушает» порты TCP/IP в ожидании входящих соединений и запускает новый серверный процесс при каждом запросе соединения. Процессы, в свою очередь, взаимодействуют друг с другом при помощи семафоров и совместно используют память, чтобы обеспечить целостность данных в процессе параллельного доступа к данным. Доступ к файлам базы данных осуществляется через разделяемые дисковые буферы.

[https://www.ibm.com/developerworks/ru/library/dm-0603wasserman2/index.html].

3.2 Реализация базы данных

Для построения физической модели базы данных (по сути, реализации базы данных) будем использовать среду IBExpert. Для её визуального отображения в виде диаграммы (рисунок 3.3) воспользуемся инструментарием MySQLWorkbench. Таким образом, логическая модель переводится в физическую путем дополнения её типами данных, размерами полей, индексами и т.д.

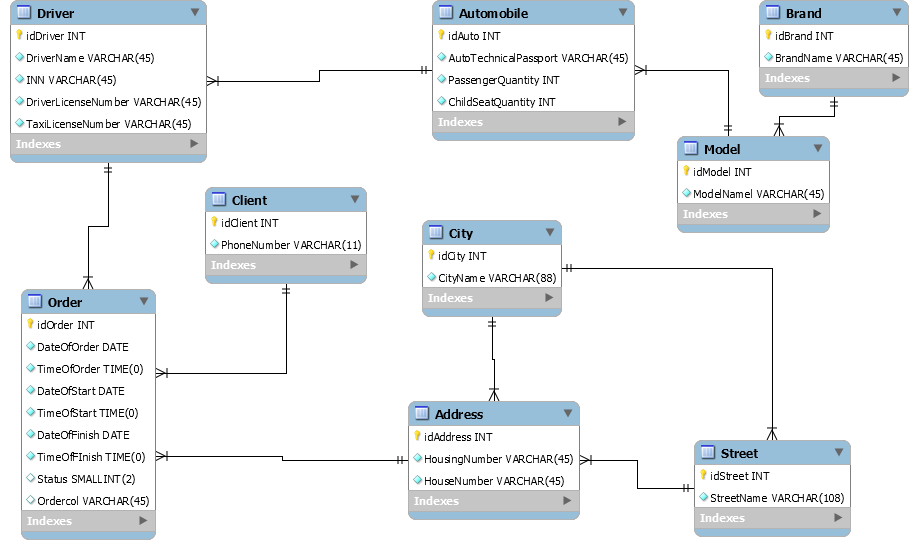


Рисунок 3.3 – Физическая модель базы данных

3.3 Тестирование базы данных

Для проверки корректной работы спроектированной базы внесем в неё некоторые осмысленные данные и построим разноплановые запросы. Формулировка запросов на естественном языке приведена ниже по тексту, листинг соответствующих запросов на языке SQL приведен в приложении Б. Результаты обработки запроса представлены на рисунках 3.4-3..

1. Выбрать информацию о заказах, поступивших на предприятие 16 января 2018 года.

2. Просмотр всех рейсов

3. Просмотр всех суден

4. Добавление моделей

5. Попробовать удалить рейс

6. Просмотр заказов в день, когда их нет

7. Выбрать информацию об активных заказах.

8. Завершение заказа

9. Добавление судна

10. Просмотр списка портов

11. Создание записи без введения обязательных полей

Рисунок 3.4 – Результат выполнения первого запроса

Рисунок 3.5 – Результат выполнения второго запроса

Рисунок 3.6 – Результат выполнения третьего запроса

Рисунок 3.7 – Результат выполнения четвертого запроса

Рисунок 3.8 – Результат выполнения пятого запроса

Рисунок 3.9 – Результат выполнения шестого запроса

Рисунок 3.10 – Результат выполнения седьмого запроса

Рисунок 3.11 – Результат выполнения восьмого запроса

Рисунок 3.12 – Результат выполнения девятого запроса

Рисунок 3.13 – Результат выполнения десятого запроса

Рисунок 3.14 – Результат выполнения одиннадцатого запроса

3.4 Разграничение прав доступа

В разделе 1 были определены следующие профили рабочих (которые можно интерпретировать как роли):

Администратор;

Менеджер;

Клиент.

Описание профилей можно увидеть в таблице 1.1 «Профили категорий пользователей ИС» раздела 1. В соответствии с данной таблицей определим таблицу 3.1, в которой опишем разграничение прав доступа.

Таблица 3.1 – Разграничение прав доступа к базе данных

|  |  |  |  |
| --- | --- | --- | --- |
| Пользователи  Действия | Администратор | Менеджер | Клиент |
| Вид доступа к БД | Полный доступ | Частичный доступ | Ограниченный доступ |
| Добавление данных | Есть возможность добавлять информацию во все таблицы | Есть возможность управлять таблицами «Порт», «Судно» и «Рейс» | Нет прав на добавление |
| Выборка данных | Есть доступ к информации всех таблиц | Есть доступ к информации всех таблиц | Есть доступ к информации таблиц «Рейс», «Порт» |
| Редактирование данных | Есть возможность редактировать информацию во всех таблицах | Есть возможность редактировать информацию в таблицах «Заказ» и «Билет», «Порт», «Судно» и «Рейс» | В личном разделе можно редактировать собственную запись в таблице «Пользователь» |
| Удаление данных | Есть возможность удаления данных из всех таблиц | Нет прав на удаление | Нет прав на удаление |

Команды объявления ролей и привилегий описаны в приложении Б.

Выводы раздела 3

В рамках данного раздела была сформирована физическая модель базы данных, реализация которой была проведена с использованием клиент-серверной СУБД Xampp. Затем созданная база данных была подвергнута тестированию посредством создания разноплановых запросов (в том числе запросов с учетом ролей пользователей) и отправки их на сервер. Листинг запросов приведен в Приложении Б, результат их выполнения – в разделе 3.3 «Тестирование базы данных». Обработка запросов, учитывающих роли пользователей, описывается в разделе 3.4 «Разграничение прав доступа».

В результате тестирования было установлено следующее:

в базе данных выполняются ограничения целостности: первичные ключи не допускают дублирования данных; поддерживаются связи по внешним ключам, за счет чего также выполняется» каскадирование» при удалении/модификации данных;

разграничение прав доступа работает корректно: пользователи не могут выполнять действия, превышающие их права.

4 РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

4.1 Обоснование выбора языка программирования

Клиентское приложение разрабатывается с использованием языка высокого уровня PHP. Для построения визуальных компонент применяется технология Yii. Среда разработки – Xampp.

**Преимущества PHP.** Данный язык вобрал в себя достоинства языков WebUI, но к тому же он специализируется на работами с базами данных.

**Преимущества YII.** Огромным плюсом данной подсистемы для построения графических интерфейсов является использование языка декларативной разметки интерфейса Yii, основанного на XML, благодаря чему можно создавать насыщенный графический интерфейс, используя или декларативное объявление интерфейса, или код на css и html, либо совмещать и то, и другое.

В основе YII лежит векторная система визуализации, не зависящая от разрешения и созданная с расчетом на возможности современного графического оборудования.

4.2 Интерфейс пользователя

Прикладная программа разрабатывается с целью обеспечить удобным интерфейсом пользователей, работающих на предприятии. С данным приложением отпадает нужда в непосредственном использовании СУБД: логика формирования запросов реализуется внутри приложения.

Структура приложения следующая:

главное окно (рисунок В.2 Приложения В);

список заказов (рисунок В.3 Приложения В);

список Автомобилей (рисунок В.4 Приложения В);

список водителей (рисунок В.5 Приложения В);

список адресов (рисунок В.6 Приложения В);

список моделей (рисунок В.7 Приложения В);

Как было описано в предыдущем разделе, на уровне базы данных были выделены роли, имеющие те или иные ограничения. Исходя из этого, в приложении была предусмотрена форма авторизации, где пользователю необходимо вводить логин и пароль. Вид данного окна можно увидеть на рисунке 4.1.

Главное окно представляет из себя форму, с размещенным на ней меню. Именно эта форма стоит в вершине иерархии, то есть, иными словами, работая с данной формой можно организовать переход к другим.

В каждой форме, содержащей тот или иной список, может быть получена более детальная информация путем нажатия на соответствующую строку. Например, в списке заказов указывается ID заказа и его стауту, однако в базе также содержится его описание.

Выводы раздела 4

С целью обеспечить пользователей удобным интерфейсом для работы с базой данных было разработано соответствующее приложение.

Возможности, предоставляемые разработанным приложением:

просмотр списка заказов;

просмотр списка авто;

просмотр списка водителей;

просмотр списка адресов;

просмотр списка моделей;

формирование SQL-запросов в явном виде и последующее получение результата выполнения запроса.

ПРИЛОЖЕНИЕ А

**Схемы и диаграммы**



Рисунок А.1 – Логическая модель разрабатываемой системы в нотации Чена

Рисунок А.2 – Модель базы данных, основанная на ключах. Первоначальный вид



Рисунок А.3 – Полная атрибутивная модель БД

ПРИЛОЖЕНИЕ Б

Текст SQL для создания БД

-- --------------------------------------------------------

-- Хост: 127.0.0.1

-- Версия сервера: 5.6.37 - MySQL Community Server (GPL)

-- Операционная система: Win32

-- HeidiSQL Версия: 9.4.0.5125

-- --------------------------------------------------------

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!50503 SET NAMES utf8mb4 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

-- Дамп структуры базы данных seas

CREATE DATABASE IF NOT EXISTS `seas` /\*!40100 DEFAULT CHARACTER SET utf8 \*/;

USE `seas`;

-- Дамп структуры для таблица seas.order

CREATE TABLE IF NOT EXISTS `order` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`user\_id` int(11) NOT NULL,

`date` datetime NOT NULL,

`state` int(11) NOT NULL,

PRIMARY KEY (`id`),

KEY `user\_id` (`user\_id`),

CONSTRAINT `order\_ibfk\_1` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Дамп данных таблицы seas.order: ~0 rows (приблизительно)

/\*!40000 ALTER TABLE `order` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `order` ENABLE KEYS \*/;

-- Дамп структуры для таблица seas.passenger

CREATE TABLE IF NOT EXISTS `passenger` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`firstName` varchar(255) NOT NULL,

`lastName` varchar(255) NOT NULL,

`secondName` varchar(255) NOT NULL,

`documentType` varchar(1) NOT NULL,

`documentNumber` varchar(15) NOT NULL,

`documentExpire` datetime NOT NULL,

`birthday` datetime NOT NULL,

`sex` varchar(1) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Дамп данных таблицы seas.passenger: ~0 rows (приблизительно)

/\*!40000 ALTER TABLE `passenger` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `passenger` ENABLE KEYS \*/;

-- Дамп структуры для таблица seas.port

CREATE TABLE IF NOT EXISTS `port` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL,

`berthCapacity` int(11) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Дамп данных таблицы seas.port: ~0 rows (приблизительно)

/\*!40000 ALTER TABLE `port` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `port` ENABLE KEYS \*/;

-- Дамп структуры для таблица seas.ticket

CREATE TABLE IF NOT EXISTS `ticket` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`passenger\_id` int(11) NOT NULL,

`order\_id` int(11) NOT NULL,

`voyage\_id` int(11) NOT NULL,

`date` datetime NOT NULL,

`time` datetime NOT NULL,

`duration` int(11) NOT NULL,

PRIMARY KEY (`id`),

KEY `passenger\_id` (`passenger\_id`),

KEY `order\_id` (`order\_id`),

KEY `voyage\_id` (`voyage\_id`),

CONSTRAINT `ticket\_ibfk\_1` FOREIGN KEY (`passenger\_id`) REFERENCES `passenger` (`id`),

CONSTRAINT `ticket\_ibfk\_2` FOREIGN KEY (`order\_id`) REFERENCES `order` (`id`),

CONSTRAINT `ticket\_ibfk\_3` FOREIGN KEY (`voyage\_id`) REFERENCES `voyage` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Дамп данных таблицы seas.ticket: ~0 rows (приблизительно)

/\*!40000 ALTER TABLE `ticket` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `ticket` ENABLE KEYS \*/;

-- Дамп структуры для таблица seas.user

CREATE TABLE IF NOT EXISTS `user` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`email` varchar(255) NOT NULL,

`phone` varchar(255) NOT NULL,

`password` varchar(255) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Дамп данных таблицы seas.user: ~0 rows (приблизительно)

/\*!40000 ALTER TABLE `user` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `user` ENABLE KEYS \*/;

-- Дамп структуры для таблица seas.vessel

CREATE TABLE IF NOT EXISTS `vessel` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL,

`number` int(11) NOT NULL,

`capacity` int(11) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Дамп данных таблицы seas.vessel: ~0 rows (приблизительно)

/\*!40000 ALTER TABLE `vessel` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `vessel` ENABLE KEYS \*/;

-- Дамп структуры для таблица seas.voyage

CREATE TABLE IF NOT EXISTS `voyage` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`vessel\_id` int(11) NOT NULL,

`from\_id` int(11) NOT NULL,

`to\_id` int(11) NOT NULL,

`date` datetime NOT NULL,

`time` datetime NOT NULL,

`duration` int(11) NOT NULL,

PRIMARY KEY (`id`),

KEY `vessel\_id` (`vessel\_id`),

KEY `from\_id` (`from\_id`),

KEY `to\_id` (`to\_id`),

CONSTRAINT `voyage\_ibfk\_1` FOREIGN KEY (`vessel\_id`) REFERENCES `vessel` (`id`),

CONSTRAINT `voyage\_ibfk\_2` FOREIGN KEY (`from\_id`) REFERENCES `port` (`id`),

CONSTRAINT `voyage\_ibfk\_3` FOREIGN KEY (`to\_id`) REFERENCES `port` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Дамп данных таблицы seas.voyage: ~0 rows (приблизительно)

/\*!40000 ALTER TABLE `voyage` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `voyage` ENABLE KEYS \*/;

/\*!40101 SET SQL\_MODE=IFNULL(@OLD\_SQL\_MODE, '') \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=IF(@OLD\_FOREIGN\_KEY\_CHECKS IS NULL, 1, @OLD\_FOREIGN\_KEY\_CHECKS) \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;