**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc120179548)

[1 ПОСТРОЕНИЕ ИНФОЛОГИЧЕСКОЙ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ 6](#_Toc120179549)

[1.1 Анализ предметной области и выявление необходимого набора сущностей 6](#_Toc120179550)

[1.2 Обоснование требуемого набора атрибутов для каждой сущности и выделение идентифицирующих атрибутов 7](#_Toc120179551)

[1.3 Определение связей между объектами 10](#_Toc120179552)

[1.4 Описание полученной модели на языке инфологического проектирования 11](#_Toc120179553)

[2 ПОСТРОЕНИЕ СХЕМЫ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ 12](#_Toc120179554)

[2.1 Построение набора необходимых отношений базы данных 12](#_Toc120179555)

[2.2 Задание первичных и внешних ключей определенных отношений 13](#_Toc120179556)

[2.3 Третья нормальная форма 14](#_Toc120179557)

[2.4 Определение ограничений целостности для внешних ключей отношений и для отношений в целом 14](#_Toc120179558)

[2.5 Графическое представление связей между внешними ключами 15](#_Toc120179559)

[3 СОЗДАНИЕ СПРОЕКТИРОВАННОЙ БАЗЫ ДАННЫХ 16](#_Toc120179560)

[4 ЗАПИСЬ ВЫРАЖЕНИЙ, УКАЗАННЫХ В ВАРИАНТЕ ЗАДАНИЯ ТИПОВ ЗАПРОСОВ НА ЯЗЫКЕ SQL 20](#_Toc120179561)

[5 ВЫБОР И ОСНОВАНИЕ СРЕДСТВ РАЗРАБОТКИ ПРИЛОЖЕНИЯ 23](#_Toc120179562)

[6 РЕАЛИЗАЦИЯ ЗАКОНЧЕННОГО ПРИЛОЖЕНИЯ, РАБОТАЮЩЕГО С СОЗДАННОЙ БАЗОЙ ДАННЫХ 25](#_Toc120179563)

[6.1 Разработка и построение интерфейса главной и рабочих форм 25](#_Toc120179564)

[6.2 Построение главного меню и кнопок панели инструментов 25](#_Toc120179565)

[6.3 Выполнение программного кода на языке C# 26](#_Toc120179566)

[ЗАКЛЮЧЕНИЕ 28](#_Toc120179567)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29](#_Toc120179568)

[ПРИЛОЖЕНИЕ А (обязательное) Концептуальная схема БД 30](#_Toc120179569)

[ПРИЛОЖЕНИЕ Б (обязательное) Схема реляционной базы данных 31](#_Toc120179570)

[ПРИЛОЖЕНИЕ В (обязательное) Описание задания курсовой работы 32](#_Toc120179571)

[ПРИЛОЖЕНИЕ Г (обязательное) Главная и рабочие формы приложения 34](#_Toc120179572)

# ВВЕДЕНИЕ

В современном информационном обществе базы данных играют ключевую роль в организации, хранении и управлении огромными объемами данных. Они служат основой для работы многих приложений и систем, обеспечивая эффективное управление информацией для бизнеса, науки, образования и других областей. Эффективное проектирование баз данных играет важную роль в обеспечении их производительности, масштабируемости и надежности.

База данных – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД). Данные вместе с СУБД, а также приложения, которые с ними связаны, называются системой баз данных, или, для краткости, просто базой данных.[1]

Цель данной курсовой работы состоит в изучении основных принципов баз данных, их архитектуры, методов проектирования и запросов, а также применения различных моделей данных в разнообразных сферах деятельности.

SQL – это язык программирования, используемый в большинстве реляционных баз данных для запросов, обработки и определения данных, а также контроля доступа. [1]

В свете быстрого развития информационных технологий и постоянного увеличения объемов данных актуальность изучения баз данных непрерывно возрастает. Понимание принципов и методов работы с базами данных является необходимым для успешной карьеры в области информационных технологий, а также для эффективного управления информацией в современном мире.

Базы данных играют решающую роль в обеспечении эффективного хранения информации, а также в обеспечении ее доступности и безопасности. Они служат основой для хранения пользовательских данных, таких как учетные записи, профили, история действий и многое другое, что делает веб-приложения более функциональными и персонализированными.

Базы данных также необходимы для обработки и анализа данных, собранных из различных источников, таких как формы обратной связи, заказы, транзакции и т. д. Эти данные могут быть использованы для создания отчетов, статистики, персонализированных рекомендаций и других функций, улучшающих пользовательский опыт и помогающих в принятии более информированных решений в веб-разработке.

В данной курсовой работе поставлена задача создания базы данных и информационной системы железнодорожной пассажирской станции.

Для создания информационной базы данных будет использоваться СУДБ phpMyAdmin. Для создания приложения – среда Visual Studio 2022.

# 1 ПОСТРОЕНИЕ ИНФОЛОГИЧЕСКОЙ КОНЦЕПТУАЛЬНОЙ

**МОДЕЛИ**

# 1.1 Анализ предметной области и выявление необходимого набора

**сущностей**

В рамках разработки информационной системы для компании программного обеспечения, важно понимать взаимодействие между ключевыми сущностями, чтобы обеспечить целостность и эффективность работы системы.

Клиенты являются центральным звеном в этой сети взаимоотношений, их обратная связь служит важным инструментом для оценки качества продукции и услуг. Работа с клиентами тесно связана с проектами, над которыми работает компания, и каждый проект может включать различные документы, технологии и задачи, требующие внимания сотрудников.

Сотрудники, в свою очередь, подразделяются по отделам и должностям, что формирует иерархию в компании и определяет потоки управления. Своевременное выполнение задач сотрудниками, отслеживаемое через журналы работ, напрямую влияет на успех проектов.

Менеджеры и руководители отделов играют ключевую роль в координации работы отделов и распределении ресурсов, а также в планировании маршрутов развития компании и её проектов. Встречи служат платформой для обсуждения хода проектов, распределения задач и стратегического планирования.

Система должна обеспечивать интеграцию всех этих элементов в единую сеть взаимодействий, что позволит компании быть гибкой и отзывчивой к изменениям в требованиях клиентов и рынка, а также эффективно управлять внутренними ресурсами и процессами.

# 1.2 Обоснование требуемого набора атрибутов для каждой сущности и

**выделение идентифицирующих атрибутов**

Обоснование требуемого набора атрибутов для каждой сущности в базе данных является важным этапом проектирования, поскольку от правильного выбора атрибутов зависит эффективность и функциональность базы данных. Набор атрибутов должен отражать основные характеристики сущностей и удовлетворять требованиям бизнес-логики приложения.

Для каждой сущности необходимо определить атрибуты, которые наилучшим образом описывают ее свойства и состояние. Например, для сущности "Клиент" могут быть необходимы атрибуты, такие как имя, фамилия, номер телефона и название проекта. Эти атрибуты позволяют эффективно хранить данные о каждом клиенте.

Выделение идентифицирующих атрибутов также крайне важно для определения уникальности каждой записи в базе данных. Данные атрибуты должны быть уникальными для каждой сущности и позволять однозначно идентифицировать записи.

В таблице 1.1 представлены сущности, определенные для них атрибуты, описание атрибутов и ключи.

Таблица 1.1 – Описание сущностей

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица | Поле | Ключ | Описание |
| technologies | TechnologyID | PK | Идентификационный номер таблицы technologies |
| Description |  | Описание технологии |
| Name |  | Название технологии |
| taskstatuses | StatusID | PK | Идентификационный номер таблицы taskstatuses |
| Name |  | Название статуса задачи |
| projects | ProjectID | PK | Идентификационный номер таблицы projects |
| Name |  | Название проекта |
| StartDate |  | Дата начала проекта |
| EndDate |  | Дата окончания проекта |
| Budget |  | Бюджет проекта |
| ClientID | FK | Идентификационный номер таблицы clients |
| ManagerID | FK | Идентификационный номер таблицы managers |

Продолжение таблицы 1.1

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица | Поле | Ключ | Описание |
| positions | PositionID | PK | Идентификационный номер таблицы positions |
| Title |  | Название должности |
| SalaryGrade |  | Количество окладов |
| employees | EmployeeID | PK | Идентификационный номер таблицы employees |
| Name |  | Имя сотрудника |
| PositionID | FK | Идентификационный номер таблицы positions |
| DepartmentID | FK | Идентификационный номер таблицы departments |
| Email |  | Адрес электронной почты |
| Phone |  | Номер телефона |
| departments | DepartmentID | PK | Идентификационный номер таблицы departments |
| Name |  | Название отдела |
| ManagerID | FK | Идентификационный номер таблицы managers |
| projecttechnologies | ProjectTechnologyID | PK | Идентификационный номер таблицы projecttechnologies |
| ProjectID | FK | Идентификационный номер таблицы projects |
| TechnologyID | FK | Идентификационный номер таблицы technologies |
| clients | ClientsID | PK | Идентификационный номер таблицы clients |
| Name |  | Название организации |
| Industry |  | Сфера деятельности |
| ContactPerson |  | Представитель компании |
| Email |  | Адрес электронной почты |
| Phone |  | Мобильный телефон |
| meetings | MeetingID | PK | Идентификационный номер таблицы meetings |
| ProjectID | FK | Идентификационный номер таблицы projects |
| MeetingDate |  | Дата собрания |
| Agenda |  | Тема собрания |

Продолжение таблицы 1.1

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица | Поле | Ключ | Описание |
| clientfeedback | FeedbackID | PK | Идентификационный номер таблицы clientfeedback |
| ProjectID | FK | Идентификационный номер таблицы projects |
| ClientID | FK | Идентификационный номер таблицы clients |
| Date |  | Дата получения отзыва |
| Text |  | Текст отзыва |
| employeetasks | EmployeeTaskID | PK | Идентификационный номер таблицы employeetasks |
| TaskID | FK | Идентификационный номер таблицы tasks |
| EmployeeID | FK | Идентификационный номер таблицы employees |
| TimeSpent |  | Затраченное время на задачу |
| Date |  | Дата выполнения задачи |
| projectdocuments | DocumentID | PK | Идентификационный номер таблицы projectdocuments |
| ProjectID | FK | Идентификационный номер таблицы projects |
| Title |  | Название документации |
| DocumentPath |  | Путь к файлу документации |
| CreationDate |  | Дата создания |
| managers | ManagerID | PK | Идентификационный номер таблицы managers |
| Name |  | Имя менеджера |
| Position |  | Должность менеджера |
| worklogs | WorkLogID | PK | Идентификационный номер таблицы worklogs |
| EmployeeID | FK | Идентификационный номер таблицы employees |
| ProjectID | FK | Идентификационный номер таблицы projects |
| HoursWorked |  | Количество отработанных часов |
| Date |  | Дата рабочей сессии |

Продолжение таблицы 1.1

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица | Поле | Ключ | Описание |
| tasks | TaskID | PK | Идентификационный номер таблицы tasks |
| ProjectID | FK | Идентификационный номер таблицы projects |
| EmployeeID | FK | Идентификационный номер таблицы employees |
| Description |  | Описание задания |
| StartDate |  | Дата начала работы над задачей |
| EndDate |  | Дата окончания работы над задачей |
| StatusID | FK | Идентификационный номер таблицы taskstatuses |

# 1.3 Определение связей между объектами

Определение связей между объектами в базе данных играет ключевую роль в создании эффективной и структурированной системы хранения данных. Связи определяют взаимосвязи между различными сущностями и указывают на способы, которыми они взаимодействуют друг с другом. В процессе проектирования базы данных необходимо тщательно определить типы связей и их характеристики для обеспечения целостности данных и эффективного доступа к ним.

Существует несколько типов связей, которые могут быть установлены между объектами в базе данных: один к одному, один ко многим, многие ко многим. Связь один к одному (One-to-One) предполагает, что каждый объект одной сущности соответствует ровно одному объекту другой сущности, и наоборот. Например, каждый пользователь может иметь только один профиль, а каждый профиль принадлежит только одному пользователю. В случае со связью один ко многим (One-to-Many) каждый объект одной сущности может соответствовать нескольким объектам другой сущности, но каждый объект другой сущности связан только с одним объектом первой сущности. Например, один отдел может содержать несколько работников, но каждый работник принадлежит только одному отделу. Связь многие ко многим (Many-to-Many) указывает на то, что каждый объект одной сущности может соответствовать нескольким объектам другой сущности, и наоборот. Например, множество работников могут работать над множеством проектов, и каждый проект может иметь несколько сотрудников, работающих над ним.

Кроме того, связи могут быть определены как обязательные (mandatory) или необязательные (optional), что указывает на то, должна ли каждая сущность в связи иметь соответствующий объект в другой сущности.

Определение связей между объектами в базе данных требует глубокого понимания бизнес-логики приложения и требований к данным. Неправильное определение связей может привести к нарушению целостности данных, избыточности или недостаточности информации. Поэтому важно тщательно анализировать и проектировать связи между объектами, чтобы обеспечить эффективное и надежное функционирование базы данных.

Проследить отношения, в которых состоят таблицы базы данных можно по схеме, изображенной на рисунке A.1 приложения A.

# 1.4 Описание полученной модели на языке инфологического

**проектирования**

Модель, разработанная на языке инфологического проектирования, представляет собой абстрактное описание структуры базы данных, основанное на концепциях и отношениях между сущностями. В этой модели сосредоточено внимание на связях между объектами, их атрибутах и характеристиках. Термины «семантическая модель», «концептуальная модель» и «инфологическая модель» являются синонимами.

Каждая сущность в модели представлена в виде объекта, для которого определены ключевые атрибуты, идентифицирующие его в контексте базы данных. Связи между сущностями отображают взаимосвязи и зависимости между объектами, позволяя понять структуру данных и способы их взаимодействия. В модели также отражены ограничения и правила целостности данных, которые гарантируют соблюдение бизнес-правил и нормализацию данных. Описание полученной модели на языке инфологического проектирования представляет собой документацию, содержащую информацию о сущностях, атрибутах, связях и правилах, необходимую для дальнейшей реализации и развертывания базы данных.

Концептуальная модель проектируемой базы данных представлена на рисунке А.1 приложения А.

# 2 ПОСТРОЕНИЕ СХЕМЫ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

# 2.1 Построение набора необходимых отношений базы данных

Построение набора необходимых отношений базы данных включает определение структуры связей между сущностями. В этом процессе учитываются типы связей и специфика взаимодействия между объектами.

Для начала определяются сущности, которые будут представлены в базе данных, и их ключевые атрибуты. Затем устанавливаются связи между этими сущностями на основе их взаимосвязей в реальном мире. После определения отношений между сущностями разрабатывается схема базы данных, которая включает в себя таблицы для каждой сущности и связей между ними. В этой схеме учитываются ключевые атрибуты, первичные и внешние ключи, что обеспечивает целостность данных и эффективное выполнение запросов к базе данных.

Важно учитывать бизнес-требования и специфику приложения при построении набора необходимых отношений базы данных, чтобы обеспечить соответствие функциональности базы данных требованиям бизнеса и удовлетворить потребности пользователей. На рисунке 2.1 представлены отношения базы данных.

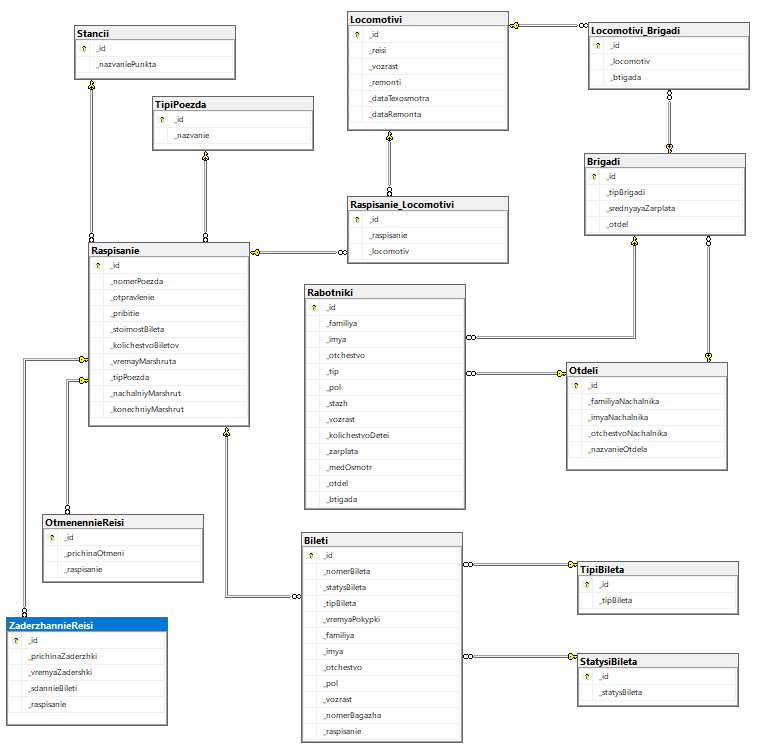


Рисунок 2.1 – Набор необходимых отношений базы данных

# 2.2 Задание первичных и внешних ключей определенных отношений

Определение первичных и внешних ключей для задания отношений в базе данных является одним из главных этапов проектирования, поскольку от правильного выбора ключей зависит целостность и эффективность базы данных. Первичный ключ (Primary Key) обеспечивает уникальную идентификацию каждой записи в таблице, тогда как внешний ключ (Foreign Key) устанавливает связь между двумя таблицами, указывая на ключевой атрибут в другой таблице.

Это позволяет базе данных совершенно точно отслеживать взаимосвязи таблиц и обеспечивает целостность данных. Например, при удалении пользователя из таблицы "Клиенты" можно настроить каскадное удаление, чтобы все его заказы автоматически удалялись из таблицы "Заказ", чтобы избежать оставшихся в базе данных "висячих" записей. Кроме того, правильно заданные ключи обеспечивают эффективность запросов к базе данных, так как они позволяют использовать индексы для быстрого доступа к данным.

Важно также учитывать нормализацию базы данных при выборе первичных и внешних ключей, чтобы избежать избыточности данных и обеспечить их структурированность. Например, можно использовать отдельную таблицу для хранения данных о заказах и ссылаться на нее с помощью внешнего ключа из таблицы "Клиенты".

Первичные и вторичные ключи представлены в таблице 1.1.

В дальнейшем построении схемы реляционной базы данных ключи будут служить для организации связей между отношениями.

# 2.3 Третья нормальная форма

Процесс преобразования базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры базы данных к виду, обеспечивающему минимальную избыточность. Для реляционных баз данных необходимо, чтобы все отношения базы данных обязательно находились в первой нормальной форме. Нормальные формы более высокого порядка могут использоваться разработчиками по своему усмотрению. Однако грамотный специалист стремится к тому, чтобы довести уровень нормализации базы данных хотя бы до третьей нормальной формы, тем самым, исключив из базы данных избыточность и аномалии обновления.

Нормальная форма – требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы данных избыточных функциональных зависимостей между полями таблиц.

Первая нормальная форма требует, чтобы база данных не имела дубликатов и составных данных. Элементы составных данных следует разнести по разным полям, иначе в процессе работы с данными могут появиться ошибки и аномалии.

Повторяющимися являются поля, содержащие одинаковые по смыслу значения. Например, если требуется получить статистику сдачи экзаменов по предметам, можно создать поля для хранения данных об оценке по каждому предмету. Однако в этом случае мы имеем дело с повторяющимися группами.

Вторая нормальная форма требует, чтобы у каждой записи в базе данных должен быть первичный ключ. Первичный ключ — это элемент записи, который не повторяется в других записях, то есть, чтобы первичный ключ однозначно определял запись и не был избыточен. Те поля, которые зависят только от части первично ключа, должны быть выделены в составе отдельных таблиц.

Третья нормальная форма требует, чтобы база данных соответствовала первым двум нормальным формам, а также в записи не должно быть столбцов с неключевыми значениями, которые зависят от других неключевых значений. Берем во внимание то, что все атрибуты наших отношений атомарны, что каждое отношение имеет первичный ключ, что в отношениях базы отсутствуют зависимости неключевых атрибутов от других неключевых атрибутов, а также отсутствуют зависимости неключевых атрибутов от части составного ключа, и делаем вывод, что отношения базы находятся в третьей нормальной форме.

# 2.4 Определение ограничений целостности для внешних ключей

**отношений и для отношений в целом**

Определение ограничений целостности для внешних ключей отношений и для отношений в целом является важной частью проектирования базы данных, направленной на обеспечение целостности и согласованности данных.

Для внешних ключей отношений определяются различные ограничения, включая ограничение на удаление (ON DELETE), ограничение на обновление (ON UPDATE). Ограничение на удаление определяет, что произойдет с записями в зависимой таблице, если запись в основной таблице будет удалена. Варианты включают в себя каскадное удаление (CASCADE), что означает удаление всех связанных записей в зависимой таблице, и установку значения NULL (SET NULL), что означает установку значения NULL в столбце с внешним ключом. Также существует ограничение на обновление (ON UPDATE), которое определяет, что произойдет с записями в зависимой таблице, если значение первичного ключа в основной таблице будет изменено. Аналогично ограничению на удаление, варианты включают в себя каскадное обновление (CASCADE) и установку значения NULL (SET NULL).

Для отношений в целом также могут быть определены ограничения целостности, включая уникальность (UNIQUE), гарантирующее, что все значения в указанных столбцах уникальны, т.е. нет повторяющихся значений; необходимость (NOT NULL) требующее, чтобы значения в указанных столбцах не были пустыми (NULL); проверка (CHECK) позволяющее определить пользовательское условие, которое должно выполняться для каждой строки в таблице.

Вместе эти ограничения обеспечивают целостность данных, защищая базу данных от некорректных операций и обеспечивая согласованность данных в различных таблицах. Правильное определение ограничений целостности является неотъемлемой частью разработки базы данных, чтобы обеспечить ее надежное и эффективное функционирование. Система управления базами данных не может контролировать правильность каждого отдельного значения, вводимого в базу данных. Для этого существует ряд средств, помогающих разработчику минимизировать возможность нарушения целостности данных базы: триггеры, проверки, уникальность и другое.

# 2.5 Графическое представление связей между внешними

**первичными ключами**

Графическое представление связей между внешними и первичными ключами в базе данных часто осуществляется с помощью диаграммы ER (сущность-связь). На диаграмме каждая таблица представлена прямоугольником, в котором указываются ее атрибуты, первичные и вторичные ключи, а связи между таблицами отображаются стрелками, указывающими направление связи и соответствующие ключевые атрибуты. Это позволяет наглядно представить структуру базы данных и взаимосвязи между таблицами. Данная диаграмма представлена на рисунке Б.1 приложения Б.

# 3 СОЗДАНИЕ СПРОЕКТИРОВАННОЙ БАЗЫ ДАННЫХ

Благодаря системе управления базами данных phpMyAdmin была реализована спроектированная ранее база данных компании, разрабатывающей программное обеспечение. Данная система была выбрана по ряду весомых причин: удобный и понятный интерфейс, достаточно широкое распространение данной систему управления базами данных, а также функциональность веб-приложения, которая не уступает привычной Microsoft SQL Server.

Описание структур каждой из таблиц базы данных с описанием типа полей представлено в таблицах ниже.

Таблица 3.1 – Характеристики атрибутов таблицы technologies

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| TechnologyID | INT | Идентификационный номер таблицы technologies |
| Name | NVARCHAR(255) | Название технологии |
| Description | TEXT | Описание технологии |

Таблица 3.2 – Характеристики атрибутов таблицы taskstatuses

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| StatusID | INT | Идентификационный номер таблицы taskstatuses |
| Name | NVARCHAR(255) | Название статуса задачи |

Таблица 3.3 – Характеристики атрибутов таблицы clients

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| ClientID | INT | Идентификационный номер таблицы clients |
| Name | NVARCHAR(255) | Название организации |
| Industry | NVARCHAR(255) | Сфера деятельности организации |
| ContactPerson | NVARCHAR(255) | Контакт менеджера компании |
| Email | NVARCHAR(255) | Адрес электронной почты |
| Phone | NVARCHAR(50) | Номер телефона |

Таблица 3.4 – Характеристики атрибутов таблицы worklog

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| WorkLogID | INT | Идентификационный номер таблицы worklogs |
| EmployeeID | INT | Идентификационный номер таблицы employees |
| ProjectID | INT | Идентификационный номер таблицы projects |
| HoursWorked | INT | Количество отработанных часов |
| Date | DATE | Дата рабочей сессии |

Таблица 3.5 – Характеристики атрибутов таблицы departments

| Поле | Тип данных | Описание |
| --- | --- | --- |
| DepartmentID | INT | Идентификационный номер таблицы departments |
| Name | NVARCHAR(255) | Название отдела |
| ManagerID | INT | Идентификационный номер таблицы managers |

Таблица 3.6 – Характеристики атрибутов таблицы managers

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| ManagerID | INT | Идентификационный номер таблицы managers |
| Name | NVARCHAR(255) | Имя менеджера |
| Position | NVARCHAR(255) | Должность менеджера |

Таблица 3.7 – Характеристики атрибутов таблицы positions

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| PositionID | INT | Идентификационный номер таблицы positions |
| Title | NVARCHAR(255) | Название позиции |
| SalaryGrade | INT | Оклад зарплаты |

Таблица 3.8 – Характеристики атрибутов таблицы projecttechnologies

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| ProjectTechologyID | INT | Идентификационный номер таблицы projecttechnologies |
| ProjectID | INT | Идентификационный номер таблицы projects |
| TechnologyID | INT | Идентификационный номер таблицы technologies |

Таблица 3.9 – Характеристики атрибутов таблицы tasks

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| TaskID | INT | Идентификационный номер таблицы tasks |
| ProjectID | INT | Идентификационный номер таблицы projects |
| EmployeeID | INT | Идентификационный номер таблицы employees |
| Description | TEXT | Описание задачи |
| StartDate | DATE | Дата начала работы над задачей |
| EndDate | DATE | Дата окончания работы над задачей |
| StatusID | INT | Идентификационный номер таблицы taskstatuses |

Таблица 3.10 – Характеристики атрибутов таблицы clientfeedback

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| FeedbackID | INT | Идентификационный номер таблицы clientfeedback |
| ProjectID | INT | Идентификационный номер таблицы projects |
| ClientID | INT | Идентификационный номер таблицы clients |
| Date | DATE | Дата записи ответа |
| Text | TEXT | Текст ответа |

Таблица 3.11 – Характеристики атрибутов таблицы projectdocuments

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| DocumentID | INT | Идентификационный номер таблицы projectsdocuments |
| ProjectID | INT | Идентификационный номер таблицы projects |
| Title | NVARCHAR(255) | Название документации |
| DocumentPath | NVARCHAR(255) | Путь к файлу документации |
| CreationDate | DATE | Дата создания файла |

Таблица 3.12 – Характеристики атрибутов таблицы employeetasks

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| EmployeeTaskID | INT | Идентификационный номер таблицы employeetasks |
| TaskID | INT | Идентификационный номер таблицы tasks |
| EmployeeID | INT | Идентификационный номер таблицы employees |
| TimeSpent | INT | Время, затраченное на выполнение задачи |
| Date | DATE | Дата выполнения задачи |

Таблица 3.13 – Характеристики атрибутов таблицы meetings

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| MeetingID | INT | Идентификационный номер таблицы meetings |
| ProjectID | INT | Идентификационный номер таблицы projects |
| MeetingDate | DATE | Дата собрания |
| Agenda | TEXT | Тема собрания |

Таблица 3.14 – Характеристики атрибутов таблицы projects

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| ProjectID | INT | Идентификационный номер таблицы projects |
| Name | NVARCHAR(255) | Название проекта |
| StartDate | DATE | Начало работы над проектом |
| EndDate | DATE | Окончание работы над проектом |

Продолжение таблицы 3.14

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Budget | DECIMAL(10,2) | Бюджет проекта |
| ClientID | INT | Идентификационный номер таблицы clients |
| ManagerID | INT | Идентификационный номер таблицы managers |

Таблица 3.15 – Характеристики атрибутов таблицы employees

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| EmployeeID | INT | Идентификационный номер таблицы employees |
| Name | NVARCHAR(255) | Имя работника |
| PositionID | INT | Идентификационный номер таблицы positions |
| DepartmentID | INT | Идентификационный номер таблицы departments |
| Email | NVARCHAR(255) | Электронная почта |
| Phone | NVARCHAR(50) | Номер телефона |

# 4 ЗАПИСЬ ВЫРАЖЕНИЙ, УКАЗАННЫХ В ВАРИАНТЕ ЗАДАНИЯ

**ТИПОВ ЗАПРОСОВ НА ЯЗЫКЕ SQL**

Описание запросов и их реализация на языке SQL для информационной системы компании, занимающейся разработкой программного обеспечения, представлены ниже. Эти запросы предназначены для обработки и анализа различных аспектов операций компании, включая данные о проектах, сотрудниках, клиентах и их взаимодействиях.

Листинг 4.1 – Получение списка всех сотрудников и их общего числа:

SELECT COUNT(\*) AS TotalEmployees, e.\* FROM employees e;

Листинг 4.2 – Вычисление общего числа сотрудников по каждому отделу:

SELECT d.name AS Department, COUNT(e.id) AS NumberOfEmployees

FROM departments d

JOIN employees e ON e.department\_id = d.id

GROUP BY d.name;

Листинг 4.3 – Определение сотрудников, участвующих в определённых проектах:

SELECT p.name AS Project, COUNT(et.employee\_id) AS Participants

FROM projects p

JOIN employeetasks et ON p.id = et.project\_id

GROUP BY p.name;

Листинг 4.4 – Получение списка проектов с участием конкретного сотрудника:

SELECT p.name

FROM projects p

JOIN employeetasks et ON p.id = et.project\_id

WHERE et.employee\_id = ?; -- здесь ? - это параметр с ID сотрудника

Листинг 4.5 – Вычисление среднего времени, затраченного на задачи по каждому проекту:

SELECT p.name AS Project, AVG(wl.hours) AS AverageTimeSpent

FROM projects p

JOIN employeetasks et ON p.id = et.project\_id

JOIN worklogs wl ON et.id = wl.employeetask\_id

GROUP BY p.name;

Листинг 4.6 – Список клиентов и количество их проектов:

SELECT c.name AS Client, COUNT(p.id) AS NumberOfProjects

FROM clients c

JOIN projects p ON c.id = p.client\_id

GROUP BY c.name;

Листинг 4.7 – Отображение обратной связи от клиентов по завершённым проектам:

SELECT p.name AS Project, cf.feedback, cf.rating

FROM projectdocuments pd

JOIN clientfeedback cf ON pd.id = cf.projectdocument\_id

JOIN projects p ON pd.project\_id = p.id;

Листинг 4.8 – Выборка встреч, запланированных на текущий месяц:

SELECT m.subject, m.date

FROM meetings m

WHERE MONTH(m.date) = MONTH(CURRENT\_DATE())

AND YEAR(m.date) = YEAR(CURRENT\_DATE());

Листинг 4.9 – Подсчёт количества технологий, использованных в проектах:

SELECT t.name AS Technology, COUNT(pt.project\_id) AS UsageCount

FROM technologies t

JOIN projecttechnologies pt ON t.id = pt.technology\_id

GROUP BY t.name;

Каждый из этих запросов SQL ориентирован на получение конкретной информации, которая может быть использована для составления отчётов, анализа производительности или планирования ресурсов в информационной системе компании. Запросы предназначены для выполнения администратором базы данных или через внутренний интерфейс приложения, обеспечивающего доступ к данным для управленческого персонала.

# 5 ВЫБОР И ОСНОВАНИЕ СРЕДСТВ РАЗРАБОТКИ

**ПРИЛОЖЕНИЯ**

В рамках проекта по разработке приложения были внимательно выбраны ключевые инструменты и программные средства, что обеспечило наиболее эффективное взаимодействие и производительность при создании и поддержке программного продукта. Одним из основных инструментов стала интегрированная среда разработки Microsoft Visual Studio 2022. Этот выбор был сделан благодаря комплексной поддержке языка C# и тесной интеграции с Entity Framework, что позволяет значительно ускорить процесс разработки серверной части приложения, облегчить отладку и упростить управление зависимостями.

Кроме того, для разработки клиентской части приложения был выбран IDE JetBrains WebStorm, который предоставляет обширные возможности для работы с TypeScript и библиотекой React. WebStorm выделяется среди других редакторов своей оптимизацией под веб-разработку, обеспечивая автоматическое дополнение кода, интуитивно понятные инструменты для отладки и удобные средства для навигации по проекту, что значительно упрощает процесс разработки интерактивных пользовательских интерфейсов.

Выбор MySQL в качестве системы управления базами данных был обусловлен её заслуженной репутацией надежной и эффективной СУБД, поддерживающей транзакции и обеспечивающей высокую производительность при работе с крупными объемами данных. MySQL обладает гибкостью в настройке и масштабировании, а также предлагает различные опции для оптимизации производительности, что критически важно для обеспечения стабильности и быстродействия приложений с большим числом пользователей.

Соединение базы данных с серверным приложением было реализовано с использованием Entity Framework, что позволяет разработчикам сосредоточиться на бизнес-логике, а не на деталях реализации доступа к данным. Entity Framework, работающий по принципу ORM (Object-Relational Mapping), автоматизирует многие процессы, связанные с обработкой данных, и предоставляет разработчикам мощные средства для работы с базой данных, используя объектно-ориентированный подход. Это не только ускоряет разработку, но и повышает уровень безопасности приложения за счет предоставления стандартизированных и проверенных решений для взаимодействия с данными.

Общий подход к выбору технологий для проекта был направлен на создание устойчивой архитектуры, которая может масштабироваться в соответствии с растущими требованиями и изменяющимися условиями использования. Использование продвинутых инструментов разработки, таких как Visual Studio и WebStorm, в комбинации с мощной базой данных и современными фреймворками, позволяет обеспечить высокую скорость разработки и поддержки, а также гарантировать, что приложение будет отвечать самым высоким стандартам производительности и безопасности. Эта стратегия обеспечивает, что разработанное приложение не только эффективно решает поставленные задачи, но и легко адаптируется под новые бизнес-требования и технологические тренды.

Дополнительно, при выборе технологий особое внимание уделялось поддержке и развитию сообщества разработчиков, которое стоит за каждым инструментом. Огромные сообщества пользователей и разработчиков Microsoft Visual Studio и JetBrains WebStorm предоставляют обширные ресурсы для обучения и поддержки, включая документацию, учебные курсы и активные форумы. Эти ресурсы значительно облегчают процесс внедрения новых функций и устранения возникающих проблем, что сокращает время на обучение новых сотрудников и ускоряет решение технических задач. MySQL, как одна из наиболее популярных систем управления базами данных, также поддерживается многочисленными ресурсами и опытным сообществом, что обеспечивает доступ к проверенным практикам и оптимизации производительности.

Безопасность является другим ключевым аспектом, который учитывался при выборе технологий. Entity Framework предоставляет множество встроенных механизмов безопасности, таких как предотвращение инъекций SQL, что повышает уровень защиты приложения от внешних угроз. Visual Studio и WebStorm поддерживают различные плагины и инструменты для проведения кодового аудита и безопасного программирования, которые помогают обнаруживать уязвимости на ранних этапах разработки. Таким образом, внедрение этих инструментов позволяет не только повысить эффективность работы разработчиков, но и значительно улучшить безопасность конечного продукта, что критически важно для поддержания доверия пользователей и соблюдения нормативных требований.

# 6 РЕАЛИЗАЦИЯ ЗАКОНЧЕННОГО ПРИЛОЖЕНИЯ,

**РАБОТАЮЩЕГО С СОЗДАННОЙ БАЗОЙ ДАННЫХ**

# 6.1 Разработка и построение интерфейса главной и рабочих форм

Разработка и построение интерфейса главной и рабочих форм приложения была выполнена с акцентом на удобство и интуитивность навигации. Программа включает в себя пять ключевых страниц: Workflow (главная), Projects (проекты), Employee (сотрудники), Clients (клиенты) и Company (компания), каждая из которых содержит таблицы, предоставляющие всю необходимую информацию в удобном для пользователя виде.

Главная страница, "Workflow", функционирует как центральный узел системы, обеспечивая прямой доступ к основным функциональным модулям приложения, включая проекты, персонал, клиентов и корпоративную информацию. Визуально она представлена в виде организованной панели навигации, облегчающей доступ к разделам и предоставляющей общий обзор системы.

Асайд-меню слева служит для легкого переключения между различными разделами приложения, позволяя пользователям быстро перемещаться между рабочими столами, соответствующими различным аспектам управления проектами и ресурсами компании. Это меню структурировано таким образом, чтобы отражать всю комплексность управления данными, сохраняя при этом простоту визуального восприятия и взаимодействия.

Каждая страница разработана с учетом уникальных требований, связанных с её функционалом, и включает интерактивные поля ввода, что позволяет пользователям эффективно и безошибочно управлять данными. Все интерфейсные решения были протестированы на соответствие требованиям пользователя, а возможные сценарии ошибок предусмотрены и снабжены ясными уведомлениями, которые информируют пользователя о характере и шагах для устранения проблемы.

Помимо функциональности, особое внимание было уделено эстетической стороне приложения. Современный дизайн, использование спокойной цветовой палитры и отсутствие перегруженных элементов создают приятное впечатление от работы с системой. Интерфейс разрабатывался с использованием библиотеки React и её встроенных средств графического дизайна, что позволило не только ускорить процесс разработки, но и достичь высокого качества визуализации элементов управления.

Детальная проработка интерфейса, а также представление скриншотов и описания работы главной и рабочих окон приложения подробно задокументированы в приложении Г документации проекта, обеспечивая полное представление о пользовательском опыте и навигации по программе.

# 6.2 Построение главного меню и кнопок панели инструментов

Навигационная панель программы представлена пунктами: Workflow (главная), Projects (проекты), Employee (сотрудники), Clients (клиенты) и Company (компания). Данные пункты выполнены в виде тематических иконок.

# 6.3 Выполнение программного кода на языке C#

Далее следует описание работы программы с базой данных. Все необходимые методы для работы с базами данных описаны в классе DBMethods и связываются с базой данных при помощи класса SqlConnection. Подключение к базе данных начинается с формирования строки подключения. Код класса DBMethods представлен в листинге 6.1.

Листинг 6.1 – Код контроллера для работы с базой данных

using api.Context;

using api.Models;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Globalization;

using System.Linq.Expressions;

using System.Reflection;

namespace api.Controllers

{

[Route("aucusoft/[controller]")]

public abstract class BaseController<TEntity, TContext> : ControllerBase

where TEntity : class

where TContext : DbContext

{

protected readonly TContext \_context;

private readonly DbSet<TEntity> \_dbSet;

public BaseController(TContext context)

{

\_context = context;

\_dbSet = context.Set<TEntity>();

}

// Абстрактное свойство для указания связей

protected abstract Expression<Func<TEntity, object>>[] Includes { get; }

// Абстрактный метод для определения проекции

Продолжение листинга 6.1

protected abstract Expression<Func<TEntity, object>> Projection { get; }

// Абстрактный метод для получения дополнительных данных

protected abstract Task<object> GetAdditionalDataAsync();

[HttpGet("{id}")]

public async Task<IActionResult> GetItem(int id)

{

var keyName = \_context.Model.FindEntityType(typeof(TEntity)).FindPrimaryKey().Properties

.Select(x => x.Name).Single();

IQueryable<TEntity> query = \_dbSet;

// Применение связей

foreach (var include in Includes)

{

query = query.Include(include);

}

// Выполнение проекции

var result = await query.Select(Projection)

.FirstOrDefaultAsync(e => EF.Property<int>(e, keyName) == id);

if (result == null)

{

return NotFound();

}

return Ok(new { data = new[] { result } });

}

[HttpGet]

public async Task<ActionResult<object>> GetAll(

[FromQuery] int pageIndex = 1,

[FromQuery] int pageSize = 1,

[FromQuery] string sortBy = null,

[FromQuery] string sortDirection = "asc",

[FromQuery] string searchQuery = null,

[FromQuery] string fieldName = null,

[FromQuery] string fieldValue = null)

{

IQueryable<TEntity> ApplySorting(IQueryable<TEntity> query, string sortBy, string sortDirection)

{

Продолжение листинга 6.1

var propertyInfo = typeof(TEntity).GetProperty(sortBy, BindingFlags.IgnoreCase | BindingFlags.Public | BindingFlags.Instance);

if (propertyInfo == null)

{

throw new ArgumentException($"Key '{sortBy}' is not a property of type '{typeof(TEntity).Name}'.");

}

var parameter = Expression.Parameter(typeof(TEntity), "x");

var property = Expression.Property(parameter, propertyInfo);

var lambda = Expression.Lambda<Func<TEntity, object>>(Expression.Convert(property, typeof(object)), parameter);

query = sortDirection.ToLower() == "asc" ? query.OrderBy(lambda) : query.OrderByDescending(lambda);

return query;

}

IQueryable<TEntity> ApplySearch(IQueryable<TEntity> query, string searchQuery)

{

var properties = typeof(TEntity).GetProperties().Where(p => p.PropertyType == typeof(string));

var searchPredicates = properties.Select(prop => {

var parameterExp = Expression.Parameter(typeof(TEntity), "type");

var propertyExp = Expression.Property(parameterExp, prop.Name);

var method = typeof(string).GetMethod("Contains", new[] { typeof(string) });

var someValue = Expression.Constant(searchQuery, typeof(string));

var containsMethodExp = Expression.Call(propertyExp, method, someValue);

return Expression.Lambda<Func<TEntity, bool>>(containsMethodExp, parameterExp);

});

var combinedExpression = searchPredicates.Aggregate((Expression<Func<TEntity, bool>>)null, (current, predicate) =>

current == null ? predicate : Expression.Lambda<Func<TEntity, bool>>(Expression.OrElse(current.Body, Expression.Invoke(predicate, current.Parameters.Single())), current.Parameters));

Продолжение листинга 6.1

if (combinedExpression != null)

query = query.Where(combinedExpression);

return query;

}

IQueryable<TEntity> ApplyFieldFilter(IQueryable<TEntity> query, string fieldName, string fieldValue)

{

if (!string.IsNullOrEmpty(fieldName) && !string.IsNullOrEmpty(fieldValue))

{

var parameterExp = Expression.Parameter(typeof(TEntity), "type");

var propertyExp = Expression.Property(parameterExp, fieldName);

var propertyType = ((PropertyInfo)propertyExp.Member).PropertyType;

if (propertyType == typeof(string))

{

MethodInfo method = typeof(string).GetMethod("Contains", new[] { typeof(string) });

var someValue = Expression.Constant(fieldValue, typeof(string));

var containsMethodExp = Expression.Call(propertyExp, method, someValue);

var lambda = Expression.Lambda<Func<TEntity, bool>>(containsMethodExp, parameterExp);

query = query.Where(lambda);

}

else if (propertyType == typeof(Nullable<decimal>) || propertyType == typeof(decimal))

{

var toStringExp = Expression.Call(propertyExp, "ToString", null, null);

MethodInfo method = typeof(string).GetMethod("Contains", new[] { typeof(string) });

var someValue = Expression.Constant(fieldValue, typeof(string));

var containsMethodExp = Expression.Call(toStringExp, method, someValue);

var lambda = Expression.Lambda<Func<TEntity, bool>>(containsMethodExp, parameterExp);

query = query.Where(lambda);

Продолжение листинга 6.1

}

}

return query;

}

IQueryable<TEntity> query = \_dbSet;

// Apply includes

foreach (var include in Includes)

{

query = query.Include(include);

}

// Apply sorting

if (!string.IsNullOrEmpty(sortBy))

{

query = ApplySorting(query, sortBy, sortDirection);

}

// Apply search and field filtering

if (!string.IsNullOrEmpty(searchQuery))

{

query = ApplySearch(query, searchQuery);

}

else if (!string.IsNullOrEmpty(fieldName) && !string.IsNullOrEmpty(fieldValue))

{

query = ApplyFieldFilter(query, fieldName, fieldValue);

}

// Apply projection

var projectedQuery = query.Select(Projection);

// Apply pagination

var data = await projectedQuery.Skip((pageIndex - 1) \* pageSize).Take(pageSize).ToListAsync();

var additional = await GetAdditionalDataAsync();

var totalItems = await query.CountAsync();

return Ok(new

{

data,

additional,

pageIndex = pageIndex,

pageSize = pageSize,

totalPages = (int)Math.Ceiling((double)totalItems / pageSize)

});

}

Продолжение листинга 6.1

[HttpPut("{id}")]

public async Task<IActionResult> Put(int id, TEntity item)

{

if (item == null)

{

return BadRequest("Item is null");

}

var existingItem = await \_dbSet.FindAsync(id);

if (existingItem == null)

{

return NotFound();

}

// Обновляем только те поля, которые были переданы

var itemProperties = item.GetType().GetProperties();

foreach (var property in itemProperties)

{

var newValue = property.GetValue(item);

if (newValue != null) // проверяем, что новое значение поля не null

{

var existingValue = property.GetValue(existingItem);

if (existingValue != newValue)

{

property.SetValue(existingItem, newValue);

}

}

}

\_context.Entry(existingItem).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!await \_dbSet.AnyAsync(e => (int)e.GetType().GetProperty("ID").GetValue(e) == id))

{

return NotFound();

}

else

{

throw;

}

Продолжение листинга 6.1

}

return NoContent();

}

[HttpPost]

public async Task<ActionResult<TEntity>> Post(TEntity item)

{

\_dbSet.Add(item);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetItem", new { id = item.GetType().GetProperty("ID").GetValue(item) }, item);

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

var item = await \_dbSet.FindAsync(id);

if (item == null)

{

return NotFound();

}

\_dbSet.Remove(item);

await \_context.SaveChangesAsync();

return NoContent();

}

}

}

# ЗАКЛЮЧЕНИЕ

В процессе выполнения данной курсовой работы были закреплены навыки проектировании баз данных и реализации их в MS SQL Server 2018.

Были определены основные цели системы в соответствии с выбранным вариантом и выделены требования, которым должна удовлетворять система.

Спроектированная база данных соответствует всем требованиям, которые предъявляются в задании. Данная база позволяет без проблем хранить и извлекать нужную информацию. Разработанная система реагирует на ошибочный ввод данных, а также способна определять возникающие ошибки и уведомлять об этом пользователя, чтобы в любой момент он знал из-за чего или почему произошла ошибка, и устранил её.

Разработанная информационная система железнодорожной пассажирской станции удовлетворяет всем требованиям комфортного использования и обеспечивает беспроблемное хранение и изменение информации.

В результате выполнения курсовой работы были спроектированы и реализованы: база данных информационной системы железнодорожной пассажирской станции, программа, которая эффективно взаимодействует с базой данных. Программное средство реализовано с помощью языка программирования C#.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. TypeScript Documentation [Электронный ресурс]. Режим доступа: https://www.typescriptlang.org/docs/. Дата доступа 12.02.2024.
  2. React documentation [Электронный ресурс]. Режим доступа: https://react.dev/learn. Дата доступа: 12.02.2024.
  3. Понятие первичного и вторичного ключа [Электронный ресурс]. Режим доступа: <http://informatic.ugatu.ac.ru/lib/office/Proekt.htm>. Дата доступа 22.11.2022.
  4. About MobX [Электронный ресурс]. Режим доступа: https://mobx.js.org/README.html. Дата доступа 22.11.2022.
  5. Нормализация [Электронный ресурс]. Режим доступа: https://metanit.com/sql/tutorial/2.1.php/. Дата доступа 19.11.2022.
  6. Entity Framework documentation hub [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/ef/. Дата доступа 20.11.2022.

# ПРИЛОЖЕНИЕ А

(обязательное)

**Концептуальная схема БД**

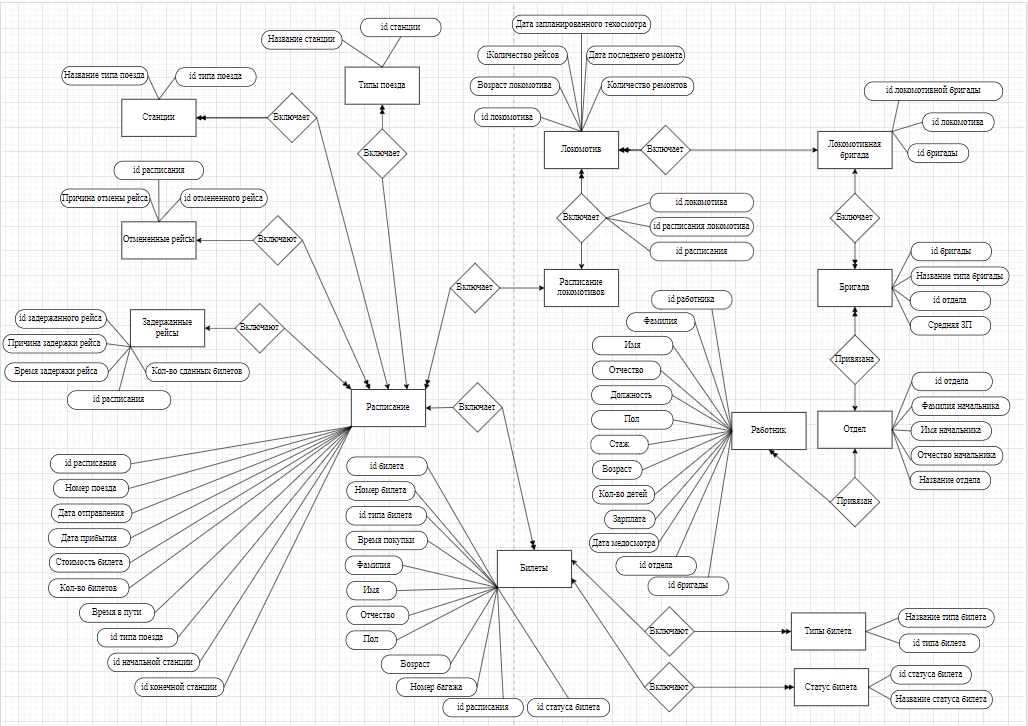


Рисунок А.1 – Концептуальная модель проектируемой базы данных

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Схема реляционной базы данных**

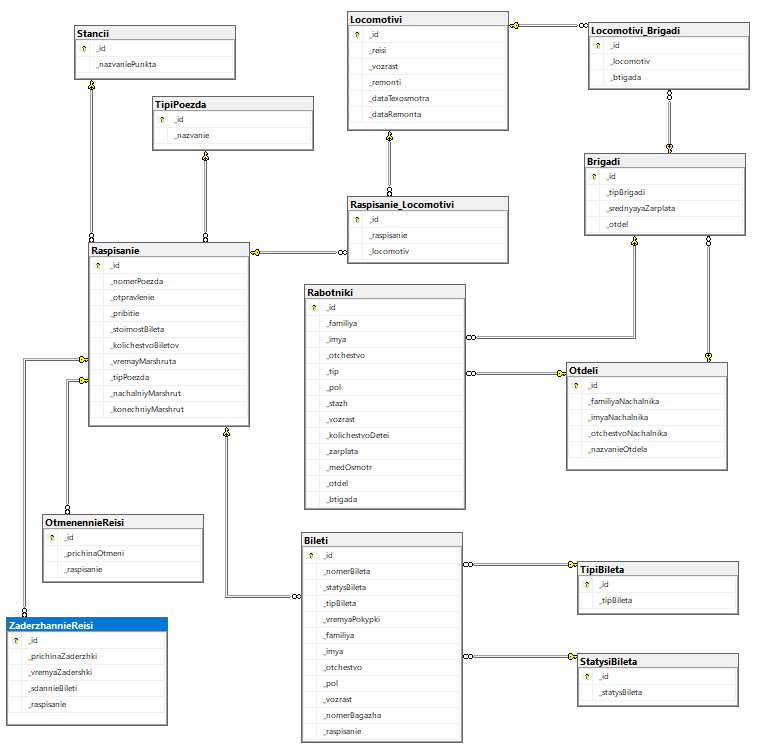


Рисунок Б.1 – Схема реляционной базы данных

# ПРИЛОЖЕНИЕ В

(обязательное)

**Описание задания курсовой работы**

Реализованная база данных должна состоять не менее чем из 15-и таблиц, а также выполнять все указанные в варианте запросы. Кроме того, в программе, при выводе данных, пользователь не должен видеть уникальные идентификаторы. Добавление, удаление, редактирование и поиск данных также не должно осуществляться с помощью уникальных идентификаторов.

Задача проекта заключается в разработке информационной системы для фирмы, занимающейся разработкой программного обеспечения. Для обеспечения целостности и эффективности работы предприятия необходимо реализовать базу данных, состоящую как минимум из пятнадцати таблиц, что позволит отразить всю важную информацию о проектах, клиентах, сотрудниках, задачах, технологиях и прочих связанных с деятельностью компании аспектах.

База данных должна обеспечивать возможность выполнения разнообразных запросов, например, выборку данных для формирования отчетов о проделанной работе, статистики использования технологий в проектах, информации по клиентам и многое другое. Особое внимание следует уделить тому, чтобы при выводе данных пользовательский интерфейс скрывал уникальные идентификаторы (ID) записей. Это требование касается всех операций в системе – от добавления и удаления записей до их редактирования и поиска.

В дополнение к этим требованиям, система должна предоставлять удобные и интуитивно понятные инструменты для работы с данными без прямого использования уникальных идентификаторов. Это подразумевает разработку дружественного пользовательского интерфейса и внедрение логики поиска и фильтрации данных, которые позволят пользователю легко находить, анализировать и обрабатывать информацию, необходимую для его работы.

Цель проекта – не только создание технической структуры базы данных, но и обеспечение высокого уровня удобства для конечных пользователей, что является ключевым фактором успешной эксплуатации информационной системы на предприятии.

# ПРИЛОЖЕНИЕ Г

(обязательное)

**Главная и рабочие формы приложения**

В ходе проектирования было определено, что при запуске приложения будет открываться главное окно с общим элементом для вывода содержимого таблиц. Окно входа в систему представлено на рисунке Г.1.

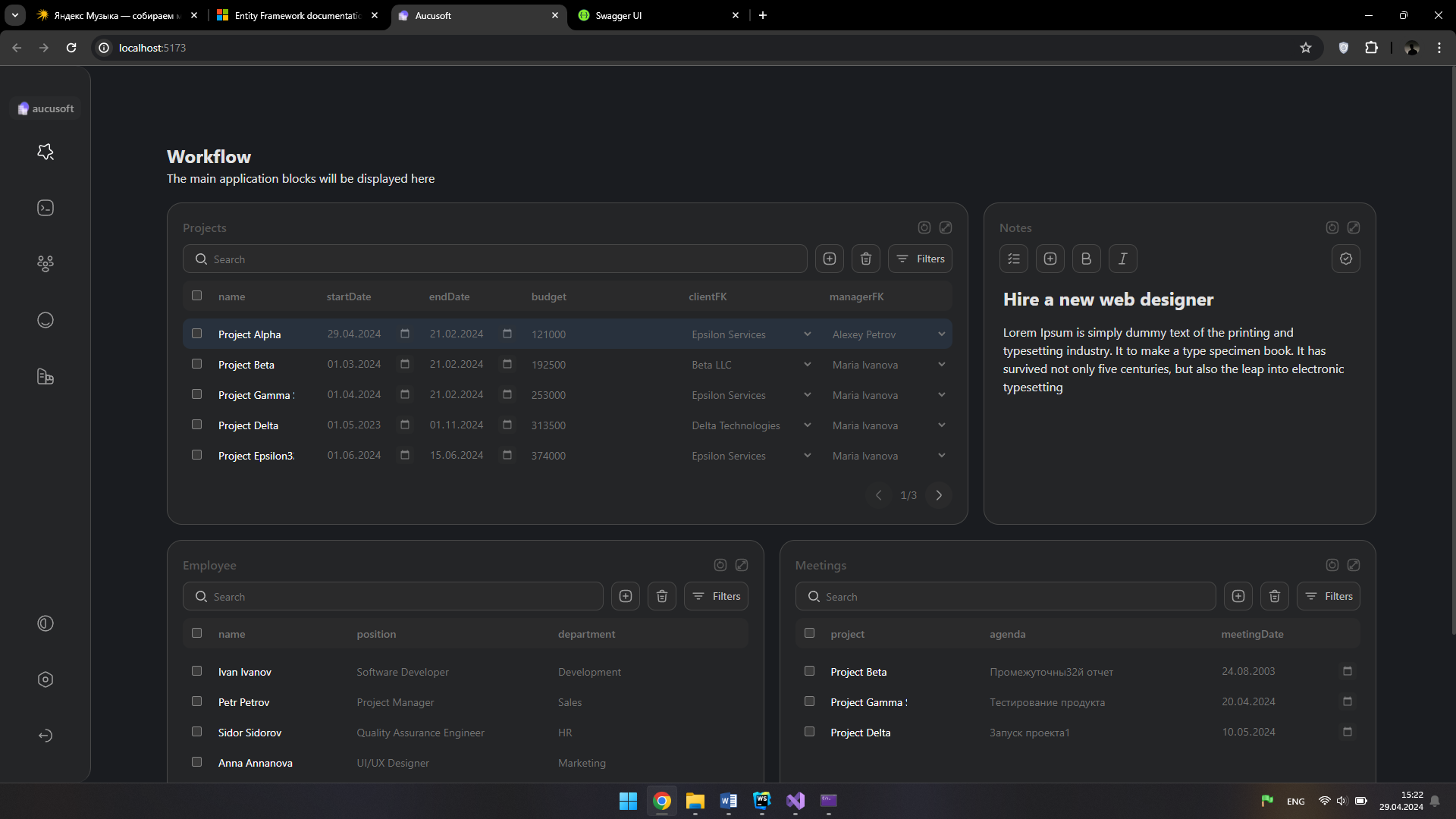


Рисунок Г.1 – Главное окно приложения