**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КОМИ**

**Государственное профессиональное образовательное учреждение**

**«Воркутинский арктический горно-политехнический колледж»**

**КУРСОВОЙ ПРОЕКТ**

По дисциплине МДК.05.02 Разработка кода информационных систем

**Архив**

Выполнил студент гр. ИСП-22 /\_\_\_\_\_\_\_\_\_\_\_\_\_/Серегин Николай Витальевич/

(подпись) (Ф.И.О.)

**ОЦЕНКА:\_\_\_\_\_\_\_\_\_\_\_\_\_**

Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ПРОВЕРИЛ**

Научный руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_/ Егоров Данил Павлович/

(подпись) (Ф.И.О.)

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc193880794)

[**ГЛАВА 1. ВЫБОР ИНСТРУМЕНТАРИЯ 4**](#_Toc193880795)

[**1.1 Платформа .NET 4**](#_Toc193880796)

[**1.2 Язык программирования С# 5**](#_Toc193880797)

[**1.3 Windows Presentation Foundation (WPF) 6**](#_Toc193880798)

[**1.4 СУБД SQL server 8**](#_Toc193880799)

[**1.5 Microsoft SQL Server Management Studio 10**](#_Toc193880800)

[**1.6 Entity Framework 12**](#_Toc193880801)

[**ГЛАВА 2. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ 15**](#_Toc193880802)

[**2.1 Разработка диаграммы ERD 15**](#_Toc193880804)

[**2.2 Разработка базы данных 15**](#_Toc193880805)

[**ГЛАВА 3. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ 16**](#_Toc193880806)

[**3.1 Разработка прототипа информационной системы 16**](#_Toc193880808)

[**3.2 Программирование информационной системы 16**](#_Toc193880809)

[**3.2.1 Разработка модуля «Авторизация» 16**](#_Toc193880810)

[**3.2.2 Разработка модуля ….. 16**](#_Toc193880811)

[**3.2.3 Разработка модуля ….. 16**](#_Toc193880812)

[**3.2.4 Разработка модуля ….. 16**](#_Toc193880813)

[**3.2.5 Разработка модуля «Отчет» 16**](#_Toc193880814)

[**ЗАКЛЮЧЕНИЕ 17**](#_Toc193880815)

[**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 18**](#_Toc193880816)

# ВВЕДЕНИЕ

Современные технологии играют все более значимую роль в различных сферах жизни, в том числе в области архивного дел. Создание информационных систем для архива становится все более актуальной задачей в современно мире.

Данная система может значительно упрости работу архивных учреждений. Для более точного и эффективного хранения и поиска документов в данном курсовом проекте будут рассмотрены основные принципы создания информационной системы для архива, ее основные возможности и преимущества, а также практические аспекты ее внедрения и использования.

В современном мире архивы играют важную роль в сохранении и систематизации информации, обеспечивая ее доступность для различных нужд.

**Объект:** информационная система «Архив».

**Предмет:** анализ бизнес-процессов «Архив».

**Цель работы:** разработать информационную систему «Архив».

**Задачи:**

* Выбрать инструментарий;
* Спроектировать базу данных;
* Разработать информационную систему

# ГЛАВА 1. ВЫБОР ИНСТРУМЕНТАРИЯ

* 1. Платформа .NET

Как-то Билл Гейтс сказал, что платформа **.NET** — это лучшее, что создала компания Microsoft. Возможно, он был прав. Фреймворк **.NET** представляет мощную платформу для создания приложений. Можно выделить следующие ее основные черты:

**Поддержка нескольких языков**. Основой платформы является общеязыковая среда исполнения **Common Language Runtime** **(CLR)**, благодаря чему **.NET** поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к **.NET**, например, **Delphi.NET**. При компиляции код на любом из этих языков компилируется в сборку на общем языке **CIL** **(Common Intermediate Language)** - своего рода ассемблер платформы **.NET**. Поэтому при определенных условиях мы можем сделать отдельные модули одного приложения на отдельных языках.

**Кроссплатформенность**. **.NET** является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент - **.NET** **9** поддерживается на большинстве современных ОС Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.

**Мощная библиотека классов**. **.NET** представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# - текстовый редактор, чат или сложный веб-сайт - так или иначе мы задействуем библиотеку классов **.NET**.

**Разнообразие технологий**. Общеязыковая среда исполнения **CLR** и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология **ADO.NET** и **Entity Framework Core**. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF и WinUI, для создания более простых графических приложений - Windows Forms. Для разработки кроссплатформенных мобильных и десктопных приложений - Xamarin/MAUI. Для создания веб-сайтов и веб-приложений - ASP.NET и т.д.

К этому стоит добавить активной развивающийся и набирающий популяность Blazor - фреймворк, который работает поверх **.NET** и который позволяет создавать веб-приложения как на стороне сервера, так и на стороне клиента. А в будущем будет поддерживать создание мобильных приложений и, возможно, десктоп-приложений.

**Производительность**. Согласно ряду тестов веб-приложения на **.NET** в ряде категорий сильно опережают веб-приложения, построенные с помощью других технологий. Приложения на **.NET** в принципе отличаются высокой производительностью.

* 1. Язык программирования С#

**C#** — это кроссплатформенный язык общего назначения, который делает разработчиков продуктивным при написании высокопроизводительного кода. С миллионами разработчиков **C#** является самым популярным языком .NET. **C#** имеет широкую поддержку в экосистеме и всех рабочих нагрузок .NET. На основе объектно-ориентированных принципов он включает множество функций из других парадигм, а не наименее функционального программирования. Низкоуровневые функции поддерживают сценарии высокой эффективности без написания небезопасного кода. Большая часть среды выполнения и библиотек .NET написана на **C#**, и прогресс в **C#** часто используется для всех разработчиков .NET.

**C#** поддерживает кроме примитивных типов данных (int, double, char, bool и т.д) ссылочные типы данных (классы, массивы, делегаты и т.д).

**Инкапсуляция** – скрытие внутренней реализации объекта, доступ только через публичные интерфейсы.

**Наследование** – создание новых классов на основе существующих с наследованием их свойств и методов.

**Полиморфизм –** возможность обработки объектов разных типов через общий интерфейс.

**Абстракция –** скрытие деталей реализации и фокус на важной функциональности.

**Делегат –** типы, представляющие ссылки на методы, позволяя передавать методы как параметры.

**События –** механизм, позволяющий объектам уведомлять другие объекты о произошедших изменениях или действиях.

**LINQ (Language Integrated Query) –** интегрированный механизм запросов для работы с данными в различных источниках с использованием синтаксиса, похожего на SQL.

**Асинхронное программирование –** позволяет выполнять длительные операции без блокировки основного потока программы.

**Сборщик мусора (Garbage Collector) –** система управления памятью, которая автоматически освобождает неиспользуемую память, предотвращая утечки.

Пример кода на **C#**:

using System;

class Hello

{

static void Main()

{

// This line prints "Hello, World"

Console.WriteLine("Hello, World");

}

}

* 1. Windows Presentation Foundation (WPF)

Технология **WPF** (**Windows Presentation Foundation**) является часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов.

Одной из важных особенностей является использование языка декларативной разметки интерфейса **XAML**, основанного на **XML**: вы можете создавать насыщенный графический интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C#, VB.NET и F#, либо совмещать и то, и другое.

Первая версия - **WPF 3.0** вышла вместе с **.NET Framework 3.0** и операционной системой Windows Vista в 2006 году. И с тех пор платформа **WPF** является частью экосистемы **.**NET и развивается вместе с фреймворком **.**NET. Например, на сегодняшний день последней версией фреймворка **.**NET является **.NET 8**, и **WPF** полностью поддерживается этой версией фреймворка.

**Преимущества WPF**

Использование традиционных языков **.NET-платформы** - C#, F# и VB.NET для создания логики приложения

Возможность декларативного определения графического интерфейса с помощью специального языка разметки **XAML**, основанном на xml и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать **XAML** и C#/VB.NET

**Независимость от разрешения экрана:** поскольку в **WPF** все элементы измеряются в независимых от устройства единицах, приложения на **WPF** легко масштабируются под разные экраны с разным разрешением.

**PresentationFramework.dll:** содержит все основные реализации компонентов и элементов управления, которые можно использовать при построении графического интерфейса

**PresentationCore.dll:** содержит все базовые типы для большинства классов из PresentationFramework.dll

**WindowsBase.dll:** содержит ряд вспомогательных классов, которые применяются в WPF, но могут также использоваться и вне данной платформы

Unmanaged API используется для интеграции вышележащего уровня с DirectX:

**milcore.dll:** собственно, обеспечивает интеграцию компонентов **WPF** с DirectX. Данный компонент написан на неуправляемом коде (С/С++) для взаимодействия с **DirectX**.

**WindowsCodecs.dll:** библиотека, которая предоставляет низкоуровневую поддержку для изображений в **WPF**

Еще ниже, собственно, находятся компоненты операционной системы и DirectX, которые производят визуализацию компонентов приложения, либо выполняют прочую низкоуровневую обработку. В частности, с помощью низкоуровневого интерфейса **Direct3D**, который входит в состав **DirectX**, происходит трансляция

Здесь также на одном уровне находится библиотека **user32.dll**. И хотя выше говорилось, что **WPF** не использует эту библиотеку для рендеринга и визуализации, однако для ряда вычислительных задач (не включающих визуализацию) данная библиотека продолжает использоваться.

* 1. СУБД SQL server

**SQL Server** является одной из наиболее популярных систем управления базами данных (СУБД) в мире. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов.

**SQL Server** характеризуется такими особенностями как:

Производительность. **SQL Server** работает очень быстро.

Надежность и безопасность. **SQL Server** предоставляет шифрование данных.

Простота. С данной СУБД относительно легко работать и вести администрирование.

Центральным аспектом в **MS SQL Server**, как и в любой СУБД, является база данных. **База данных** представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз данных применяются системы управления базами данных (**database management system**) или СУБД (DBMS). И как раз **MS SQL Server** является одной из такой СУБД.

Для организации баз данных **MS SQL Server** использует реляционную модель. Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

Для идентификации каждой строки в рамках таблицы применяется первичный ключ (**primary key**). В качестве первичного ключа может выступать один или несколько столбцов. Используя первичный ключ, мы можем ссылаться на определенную строку в таблице. Соответственно две строки не могут иметь один и тот же первичный ключ.

Через ключи одна таблица может быть связана с другой, то есть между двумя таблицами могут быть организованы связи. А сама таблица может быть представлена в виде отношения ("**relation**").

Для взаимодействия с базой данных применяется язык **SQL (Structured Query Language)**. Клиент (например, внешняя программа) отправляет запрос на языке **SQL** посредством специального API. СУБД должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения.

Выделяются две разновидности языка **SQL**: **PL-SQL** и **T-SQL**. **PL-SQL** используется в таких СУБД как **Oracle** и **MySQL**. **T-SQL** (**Transact-SQL**) применяется в **SQL Server**.

В зависимости от задачи, которую выполняет команда **T-SQL**, он может принадлежать к одному из следующих типов:

**DDL** (Data Definition Language / Язык определения данных). К этому типу относятся различные команды, которые создают базу данных, таблицы, индексы, хранимые процедуры и т.д. В общем определяют данные.

В частности, к этому типу мы можем отнести следующие команды:

**CREATE**: создает объекты базы данных (саму базу данных, таблицы, индексы и т.д.)

**ALTER**: изменяет объекты базы данных

**DROP**: удаляет объекты базы данных

**TRUNCATE**: удаляет все данные из таблиц

**DML** (Data Manipulation Language / Язык манипуляции данными). К этому типу относят команды на выборку данных, их обновление, добавление, удаление - в общем все те команды, с помощью которыми мы можем управлять данными.

К этому типу относятся следующие команды:

**SELECT**: извлекает данные из БД

**UPDATE**: обновляет данные

**INSERT**: добавляет новые данные

**DELETE**: удаляет данные

**DCL** (Data Control Language / Язык управления доступа к данным). К этому типу относят команды, которые управляют правами по доступу к данным. В частности, это следующие команды:

**GRANT**: предоставляет права для доступа к данным

**REVOKE**: отзывает права на доступ к данным

* 1. Microsoft SQL Server Management Studio

**SQL Server Management Studio (SSMS)** — это интегрированная среда для управления любой инфраструктурой **SQL**. Используйте **SSMS** для доступа, настройки, управления, администрирования и разработки всех компонентов **SQL** Server, Базы данных **SQL Azure**, Управляемого экземпляра **SQL Azure**, **SQL Server** на виртуальной машине **Azure** и **Azure Synapse Analytics**. **SSMS** предоставляет единую комплексную служебную программу, которая сочетает в себе широкую группу графических инструментов с множеством многофункциональных редакторов сценариев для предоставления доступа к **SQL** **Server** разработчикам и администраторам баз данных любого уровня квалификации.

Среда **SQL Server Management Studio (SSMS)** позволяет управлять объектами служб **Analysis Services**, например выполнять резервное копирование и обработку объектов.

**SSMS** предоставляет проект скриптов служб **Analysis Services**, в котором можно разрабатывать и сохранять скрипты, написанные в **многомерных выражениях (MDX),** **выражениях анализа данных (DAX),** **расширениях интеллектуального анализа данных (DMX)** и **XML для анализа (XMLA).**

Эти сценарии используются для выполнения задач управления или воссоздания объектов, таких как базы данных и экземпляры, в кубах служб **Analysis Services**. Например, можно разработать сценарий **XMLA** в проекте сценария служб **Analysis Services** для создания новых объектов непосредственно в существующем экземпляре. Эти проекты могут быть сохранены как часть решения и интегрированы с системой управления исходным кодом.

**SQL Server Management Studio (SSMS)** можно использовать для управления и мониторинга выполняемых пакетов служб **SSIS**. Вы можете упорядочивать пакеты по папкам, запускать, импортировать, экспортировать и обновлять пакеты служб **Integration Services**. Однако, начиная с **SSIS 2012**, хранение пакетов изменилось. Они больше не хранятся в базе данных сервера экземпляра по умолчанию, а теперь управляются через базу данных каталога служб **SSIS** (). Это означает, что вы больше не можете управлять пакетами так же, как это было в предыдущих версиях служб **SSIS**. Вы по-прежнему можете использовать **SSMS** для управления базой данных каталога служб **SSIS**, но необходимо использовать узел каталогов служб **Integration Services** в обозревателе объектов.msdbSSISDB

Последняя версия **SSMS** предоставляет интегрированную среду для управления любой инфраструктурой **SQL**. Он также позволяет пользователям запускать пакеты служб **SSIS**, хранящиеся в каталоге служб **SSIS**, из обозревателя объектов в системе **SSMS**.

Мастер импорта и экспорта в **SSMS** можно использовать для создания пакетов служб **SSIS**, что является хорошей отправной точкой для изучения служб **SSIS**. Однако для создания пакетов для более сложных пакетов и управления ими необходимо использовать **SQL Server Data Tools (SSDT).**

Среда **SQL** **Server Management Studio** (**SSMS**) включает функции служб Reporting Services, администрирует сервер и базы данных, а также управляет ролями и заданиями. Вы можете управлять общими расписаниями с помощью папки **Shared Schedules** и управлять базами данных сервера отчетов (, ). При перемещении базы данных сервера отчетов в новый экземпляр **SQL Server** необходимо создать роль **RSExecRole** в системной базе данных.ReportServerReportServerTempDBmaster

* 1. Entity Framework

**Entity Framework** представляет **ORM**-технологию (**object-relational mapping** - отображения данных на реальные объекты) от компании Microsoft для доступа к данным. **Entity Framework Core** позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными как с объектами классом независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает **Entity Framework**, мы уже работаем с объектами.

Поскольку **Entity Framework Core** работает на основе платформы .NET, то он развивается вместе с данной платформой. Текущая версия **EF Core - 9.0** была выпущена в ноябре 2024 года вместе с .NET 8., и технология продолжает развиваться.

**Entity Framework Core** поддерживает множество различных систем баз данных. Таким образом, мы можем через **EF Core** работать с любой СУБД, если для нее имеется нужный провайдер. По умолчанию на данный момент Microsoft предоставляет ряд встроенных провайдеров: для работы с **MS SQL Server**, для **SQLite**, для **PostgreSQL**. Также имеются провайдеры от сторонних поставщиков, например, для **MySQL**.

Центральной концепцией **Entity Framework** является понятие сущности или **entity**. Сущность определяет набор данных, которые связаны с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их коллекциями.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст. Свойства необязательно представляют простые данные типа int или string, но могут также представлять и более комплексные типы данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой **Entity Framework Core**, как технологии **ORM**, является использование запросов **LINQ** для выборки данных из БД. С помощью **LINQ** мы можем создавать различные запросы на выборку объектов, в том числе связанных различными ассоциативными связями. А **Entity Framework** при выполнении запроса транслирует выражения **LINQ** в выражения, понятные для конкретной СУБД (как правило, в выражения SQL).

Основная функциональность **Entity Framework Core** сосредоточена в

следующих пакетах:

Microsoft.EntityFrameworkCore: основной пакет EF Core

Microsoft.EntityFrameworkCore.SqlServer: представляет функциональность

провайдера для Microsoft SQL Server и SQL Azure

Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite: предоставляет

поддержку географических типов (spatial types) для SQL Server

Microsoft.EntityFrameworkCore.Sqlite: представляет функциональность

провайдера для SQLite и включает нативные бинарные файлы для движка базы данных

Microsoft.EntityFrameworkCore.Sqlite.Core: представляет

функциональность провайдера для SQLite, но в отличие от предыдущего пакета

не содержит нативные бинарные файлы для движка базы данных

Microsoft.EntityFrameworkCore.Sqlite.NetTopologySuite: предоставляет

поддержку географических типов (spatial types) для SQLite

Microsoft.EntityFrameworkCore.Cosmos: представляет функциональность

провайдера для Azure Cosmos DB

Microsoft.EntityFrameworkCore.InMemory: представляет

функциональность провайдера базы данных в памяти

Microsoft.EntityFrameworkCore.Tools: содержит команды EF Core

PowerShell для Visual Studio Package Manager Console; применяется в Visual

Studio для миграций и генерации классов по готовой бд

Microsoft.EntityFrameworkCore.Design: содержит вспомогательные

компоненты EF Core, применяемые в процессе разработки

Microsoft.EntityFrameworkCore.Proxies: хранит функциональность для так

называемой "ленивой загрузки" (lazy-loading) и прокси остлеживания изменений

Microsoft.EntityFrameworkCore.Abstractions: содержит набор абстракций

EF Core, которые не зависят от конкретной СУБД

Microsoft.EntityFrameworkCore.Relational: хранит компоненты EF Core для

провайдеров реляционных СУБД

Microsoft.EntityFrameworkCore.Analyzers: содержит функционал

анализаторов C# для EF Core

# ГЛАВА 2. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

1. 1. Разработка диаграммы ERD

Схема «сущность-связь» (также **ERD** или **ER-диаграмма**) — это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы. **ER-диаграммы** чаще всего применяются для проектирования и отладки реляционных баз данных в сфере образования, исследования и разработки программного обеспечения и информационных систем для бизнеса. **ER-диаграммы** (или **ER-модели**) полагаются на стандартный набор символов, включая прямоугольники, ромбы, овалы и соединительные линии, для отображения сущностей, их атрибутов и связей. Эти диаграммы устроены по тому же принципу, что и грамматические структуры: сущности выполняют роль существительных, а связи — глаголов.

В ER-моделях и моделях данных обычно выделяют до трех уровней детализации:

**Концептуальная модель данных**

Схема наивысшего уровня с минимальным количеством подробностей. Достоинство этого подхода заключается в возможности отобразить общую структуру модели и всю архитектуру системы. Менее масштабные системы могут обойтись и без этой модели. В этом случае можно сразу переходить к логической модели.

**Логическая модель данных**

Содержит более подробную информацию, нежели концептуальная модель. На этом уровне определяются более подробные операционные и транзакционные сущности. Логическая модель не зависит от технологии, в которой она будет применяться.

**Физическая модель данных**

На основе каждой логической модели данных можно составить одну или две физических модели. В последних должно присутствовать достаточно технических подробностей для составления и внедрения самой базы данных.

Диаграммы «сущность-связь» (или **ERD**) — неотъемлемая составляющая процесса моделирования любых систем, включая простые и сложные базы данных, однако применяемые в них фигуры и способы нотации могут запросто ввести в заблуждение любого.

Для того чтобы построить ER-диаграмму, можно использовать разные нотации. Три самые известные из них:

1. **Нотация IDEF1X.** Её относят к фундаментальным, но на практике давно не используют, потому что есть более удобные варианты.

2. **Нотация Чена.** Классическая нотация, которая состоит из простых символов — прямоугольников, овалов и линий. Из-за этого нотацию часто используют для концептуальных моделей, которые презентуют заказчику. Человеку, который далёк от аналитики данных, проще разобраться в понятных диаграммах со знакомыми символами.

3. **Нотация Мартина.** Её ещё называют «воронья лапка» (от англ. **Crow's Foot**). Она компактнее нотации Чена, поэтому её используют для построения ER-моделей логического уровня, когда нужно описать в модели все атрибуты сущностей.

В нотациях Чена и Мартина есть одинаковые элементы: сущности, атрибуты и связи. Но эти элементы диаграмм обозначают разными символами.

Элементы ER-диаграммы в нотации Чена соединяют линиями. Если линия соединяет две сущности, сверху обозначают тип связи:

* 1:1 — «один-к-одному»;
* 1:N — «один-ко-многим»;
* M:N — «многие-ко-многим».

В нотации Мартина сущность также вписывают в прямоугольник, а атрибуты и связи обозначают по-другому:

* Атрибуты перечисляют прямо под сущностью;
* Связи рисуют разными соединительными линиями.

**Концептуальные модели** данных дают общее представление о том, что должно входить в состав модели. Концептуальные **ER-диаграммы** можно брать за основу логических моделей данных. Их также можно использовать для создания отношений общности между разными **ER-моделями**, положив их в основу интеграции.

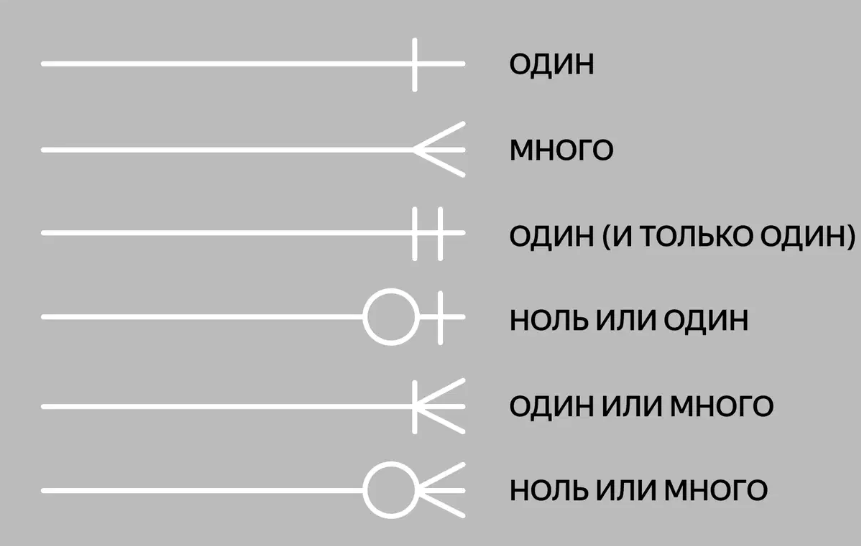


Рисунок 2.1.1 Виды связей в ER диаграмме.

* 1. Разработка базы данных

# ГЛАВА 3. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ

1. 1. Разработка прототипа информационной системы
   2. Программирование информационной системы

3.2.1 Разработка модуля «Авторизация»

3.2.2 Разработка модуля …..

3.2.3 Разработка модуля …..

3.2.4 Разработка модуля …..

3.2.5 Разработка модуля «Отчет»

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Полное руководство по языку программирования С# 13 и платформе .NET 9. [Электронный ресурс] — режим доступа: <https://metanit.com/sharp/tutorial/1.1.php>
2. Документация по C# [Электронный ресурс] — режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/overview>
3. Руководство по WPF [Электронный ресурс] — режим доступа: <https://metanit.com/sharp/wpf/1.php>
4. Введение в MS SQL Server и T-SQL [Электронный ресурс] — режим доступа: <https://metanit.com/sql/sqlserver/1.1.php>
5. Документация по Microsoft SQL [Электронный ресурс] — режим доступа: <https://learn.microsoft.com/ru-ru/ssms/sql-server-management-studio-ssms>
6. Введение в Entity Framework Core [Электронный ресурс] — режим доступа: <https://metanit.com/sharp/efcore/1.1.php>