Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Програма-конвертер файлів баз даних у файли формату XML»

Виконав: студент групи К20-1
Спеціальність 122 «Комп'ютерні науки»
Котельва М. А.
(прізвище та ініціали)

Керівник
к. т. н., доцент Мормуль М. Ф.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент
Національний університет імені Олеся Гончара
(місце роботи)
доцент кафедри обчислювальної математики
та математичної кібернетики
(посада)
к. ф.-м. н., доцент Трофімов О. В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Котельва М. А. Програма-конвертер файлів баз даних у файли формату XML.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 122 «Комп’ютерні науки». – Університет митної справи та фінансів, Дніпро, 2024.

Дана кваліфікаційна робота присвячена розробці програми-конвертера файлів баз даних у файли формату XML. Сучасні інформаційні системи часто потребують універсального та стандартизованого формату обміну даними, і XML (Extensible Markup Language) є одним з найбільш поширених і зручних для цього форматів. Розглянуті методи та технічні засоби, необхідні для реалізації системи, сформульовані вимоги до програмного забезпечення. Було проведено проектування та розробку системи, а також її тестування. Отримані результати мають практичне значення, оскільки сприяють автоматизації та поліпшенню швидкості перенесення даних з баз даних. Розроблена система є надійною та ефективною, та є важливим кроком у автоматизації перенесення даних, що може застосовуватися у різних галузях. Розроблена система отримує дані для отримання доступу до бази даних, де знаходиться інформація, або файл із потрібними даними від користувача та конвертує ці дані у файл формату XML. Цей додаток має простий та практичний дизайн.

Ключові слова: розробка, база даних, sql, автоматизація, XML, форматування.

ABSTRACTS

Kotelva M. A. Program-converter of database files into XML files.

Qualification work for a bachelor's degree in speciality 122 "Computer Science." - University of Customs and Finance, Dnipro, 2024.

This qualification work is devoted to the development of a database file converter program to XML files. Modern information systems often require a universal and standardised data exchange format, and XML (Extensible Markup Language) is one of the most common and convenient formats for this purpose. The methods and technical means necessary for the implementation of the system are considered, and software requirements are formulated. The system was designed and developed, as well as tested. The obtained results are of practical importance, as they contribute to automation and improvement of the speed of data transfer from databases. The developed system is reliable and efficient, and is an important step in the automation of data transfer, which can be used in various industries. The developed system receives data to access the database where the information is located or a file with the required data from the user and converts this data into an XML file. This application has a simple and practical design.

Keywords: development, database, sql, automation, XML, formatting.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасному світі інформаційні технології грають важливу роль у функціонуванні різних галузей економіки, науки, освіти та повсякденного життя. Збереження, обробка та обмін даними є невід'ємною частиною діяльності підприємств, організацій та установ. Бази даних є основним інструментом для організації та зберігання великих обсягів структурованої інформації. Однак ефективне управління даними є критично важливим для забезпечення конкурентоспроможності та розвитку організацій. Одним із ключових аспектів роботи з даними є їх зберігання, обробка та обмін між різними системами та платформами.

У багатьох організаціях використовуються різноманітні інформаційні системи та бази даних, які можуть відрізнятися за своєю структурою, форматом та способом зберігання даних. Часто виникає необхідність інтеграції цих систем для забезпечення безперервного потоку даних між ними. Формат XML є універсальним засобом для обміну даними, оскільки він забезпечує гнучкість та самодокументованість. Завдяки своїй структурі XML дозволяє зберігати дані у вигляді, що легко зчитується як машинами, так і людьми, що сприяє спрощенню інтеграційних процесів.

Конвертація даних з баз даних у формат XML дозволяє значно підвищити ефективність обробки інформації. XML-файли можуть бути легко оброблені за допомогою різноманітних інструментів та бібліотек, що підтримують цей формат. Це відкриває нові можливості для аналізу даних, їх трансформації та представлення у зручному вигляді для користувачів. Крім того, XML дозволяє зберігати метадані, що забезпечує додатковий контекст для розуміння змісту даних.

Сучасні веб-технології широко використовують XML для передачі даних між клієнтськими та серверними додатками. Формат XML є основою для таких стандартів, як SOAP (Simple Object Access Protocol) та WSDL (Web Services Description Language), які активно використовуються в веб-службах

та сервісно-орієнтованих архітектурах (SOA). Наявність даних у форматі XML значно спрощує інтеграцію з веб-сервісами та іншими інтернет-додатками, що підвищує загальну гнучкість та масштабованість систем.

У багатьох галузях, таких як банківська справа, медицина, наука та освіта, обробляються великі обсяги даних, які необхідно зберігати та обмінювати між різними установами та системами. XML дозволяє ефективно структурувати та зберігати ці дані, забезпечуючи їх цілісність та доступність. Використання XML у таких випадках дозволяє уникнути втрати даних при передачі між різними системами та забезпечує високу надійність обміну інформацією.

З розвитком технологій та збільшенням обсягів даних, формат XML продовжує залишатися актуальним і затребуваним. Його універсальність, гнучкість та сумісність з різними системами забезпечують його використання в нових галузях та додатках. Розробка ефективних інструментів для конвертації даних у формат XML є важливою задачею, яка сприятиме подальшому розвитку інформаційних систем та технологій.

Метою роботи є розробка програми-конвертера, яка забезпечить перетворення файлів, що містять інформацію з баз даних, у формат XML, що сприятиме автоматизації перенесення даних з баз даних.

Методи дослідження: метод теорії інформації, обробка та аналіз інформації, методи проектування та розробки програмного забезпечення.

У відповідності до поставленої мети в кваліфікаційній роботі ставились та вирішувались наступні завдання дослідження.

1. Провести аналіз існуючих методів та алгоритмів конвертації даних у формат XML. Це включає вивчення наукової літератури, технічної документації та огляд існуючих рішень, які використовуються для перетворення даних з баз даних у формат XML.

2. Розробити структуру та архітектуру програми-конвертера. Визначити основні компоненти програми, їх функції та взаємодію між ними.

3. Реалізувати програму-конвертер з використанням сучасних засобів програмування. Обрати мову програмування та технології, які найкраще підходять для реалізації даної задачі. Написати код програми та забезпечити його документування.

4. Описати можливі шляхи подальшого розвитку та вдосконалення програми. Розглянути можливості розширення функціональності програми, додавання нових форматів для конвертації, оптимізації продуктивності та покращення інтерфейсу користувача.

Об'єктом дослідження є процес конвертації даних з баз даних у формат XML.

Предметом дослідження є програмні засоби та алгоритми, що забезпечують перетворення даних з баз даних у формат XML. Це включає вивчення існуючих бібліотек, фреймворків та інструментів, які можуть бути використані для реалізації програми-конвертера.

Структура роботи.

Розділ 1. Дослідження предметної області. Постановка завдань дослідження. У даному розділі буде виконано пошук та аналіз публікацій стосовно тем баз даних, формату файлів XML та конвертації файлів баз даних в файли XML.

Розділ 2. Аналіз засобів реалізації конвертера файлів. В даному розділі буде проаналізовано схожі програмні рішення та обрано технології для реалізації програми-конвертера файлів баз даних у файли формату XML.

Розділ 3. Розробка програми-конвертера файлів баз даних у файли формату XML. В даному розділі буде спроектовано та розроблено програму-конвертер файлів баз даних у файли формату XML.

Робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел з 24 найменувань, 1 додатку. Обсяг роботи 91 сторінка, 29 рисунків.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАННЯ ДОСЛІДЖЕННЯ

1.1. Огляд розроблених програмних рішень

Сайт Aspose.app надає можливість конвертувати SQL-файли у формат XML за допомогою онлайн-інструменту (див. рис. 1.1). Для цього потрібно завантажити SQL-файл, після чого інструмент автоматично виконає перетворення даних у XML. Також сайт підтримує можливість введення скопійованих користувачем даних із SQL-файлу напряму та конвертування файлів із хмарних сховищ даних Dropbox та Google Drive. Інструмент також підтримує різні формати даних (docx, pptx, pdf, xlsx, csv, png, epub тощо) і пропонує безкоштовне використання для базових операцій. З мінусів варто відмітити відсутність інших форматів окрім SQL, та не можливість завантаження файлів з інших популярних хмарних сховищ, таких як, наприклад, Microsoft OneDrive або MEGA.

Рисунок 1.1 – Веб-сторінка конвертора файлів Aspose.app

Сайт tableconvert.com також пропонує конвертувати SQL-файли у формат XML за допомогою онлайн-інструменту (див. рис. 1.2). Для цього потрібно завантажити SQL-файл, або вставити дані вручну у поле для вводу, або надати URL, з якого потрібно вивести дані. Особливістями цього сервісу є можливість редагування SQL-таблиці безпосередньо перед конвертацією та наявність демонстраційного прикладу. Сервіс також підтримує різні формати файлів баз даних для вводу (SQL, MySQL, csv, Markdown, XML) та виводу даних (xlsx, csv, pdf, XML, json, html тощо).

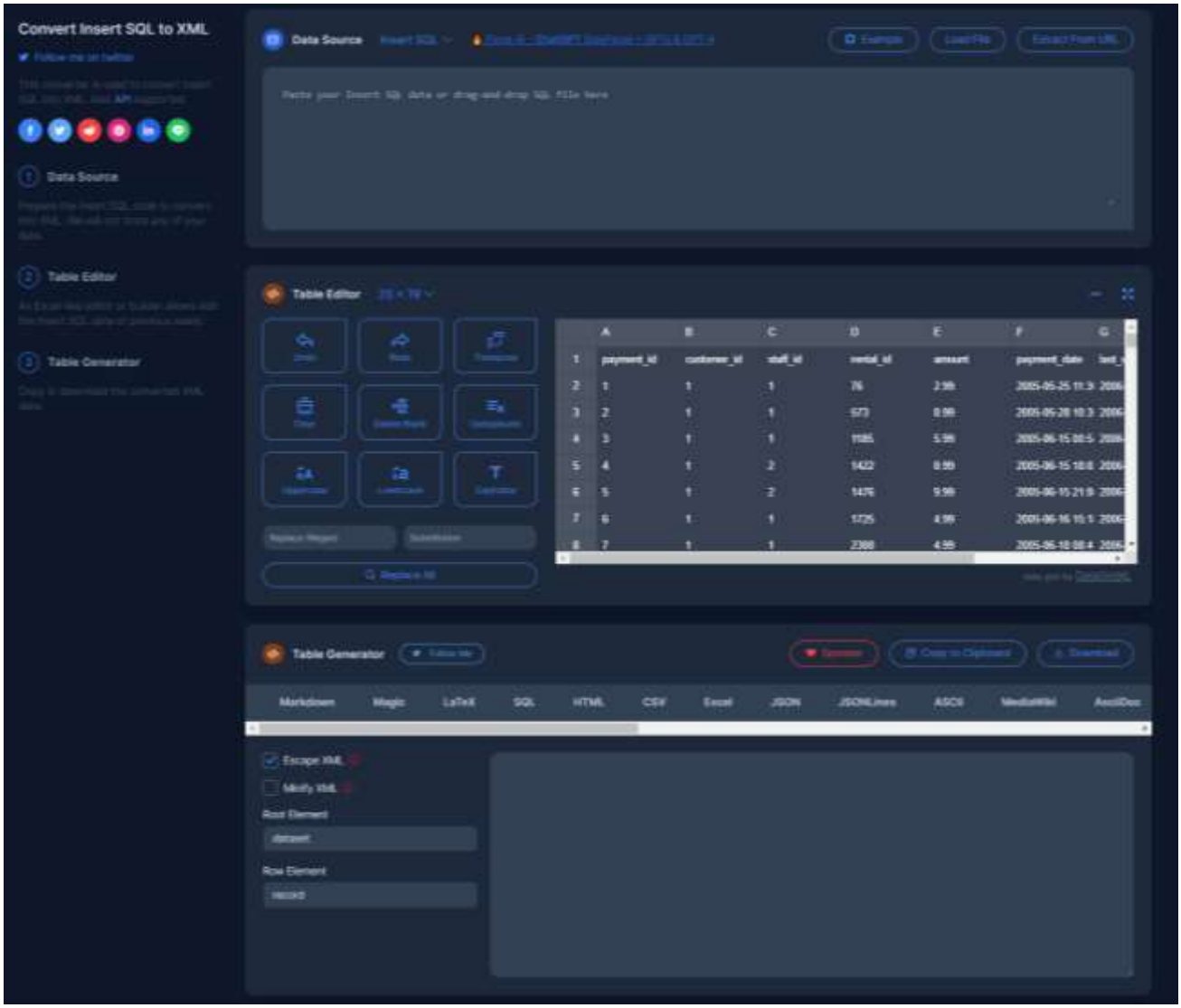

Рисунок 1.2 – Веб-сторінка конвертора файлів tableconvert.com

Веб-сайт onlinetools.com (див. рис. 1.3) пропонує різноманітні онлайн-інструменти, в тому числі можливість конвертувати CSV-файли у формат XML. Інструмент має простий інтерфейс, за допомогою якого користувачі можуть завантажувати свої CSV-файли безпосередньо зі свого комп'ютера. Користувачі можуть переглядати та маніпулювати даними CSV перед конвертацією, забезпечуючи правильне форматування та вміст. Також інструмент дозволяє налаштувати деякі опції як для вхідного файлу (наприклад обрати символ роздільник), так і для отримуваного XML-файлу (наприклад можливість згенерувати мета тег).

Рисунок 1.3 – Веб-сторінка конвертора файлів onlinetools.com

Веб-сайт AnyConv (див. рис. 1.4) пропонує широкий спектр онлайн-інструментів для конвертації файлів різних форматів. Один з таких інструментів – конвертер DBF в XML. Сайт має простий інтерфейс, який дозволяє користувачам легко завантажувати свої файли DBF безпосередньо з комп'ютера. З мінусів варто відмітити відсутність інших форматів окрім DBF, який в наш час сам є застарілим форматом.

Рисунок 1.4 – Веб-сторінка конвертора файлів anyconv.com

Такі сайти забезпечують простий і зручний інтерфейс для користувачів, дозволяючи швидко отримати необхідний результат без необхідності встановлення додаткового програмного забезпечення.

Застосунок FOSS Database Converter (див. рис. 1.5) оперує базами даних MySQL, Oracle та MSSQL за допомогою підключення до бази даних і пропонує такі функції:

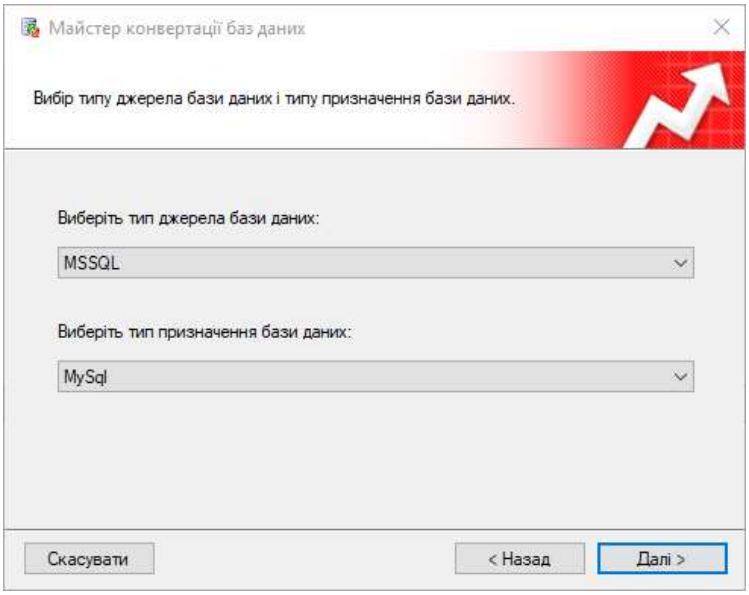

Рисунок 1.5 – Головне вікно конвертора БД FOSS Database Converter

– дозволяє конвертувати бази даних одного провайдеру до бази даних іншого, наприклад, якщо база працює на MySQL, можна перенести її на сервер БД під керуванням MS SQL (див. рис. 1.6);

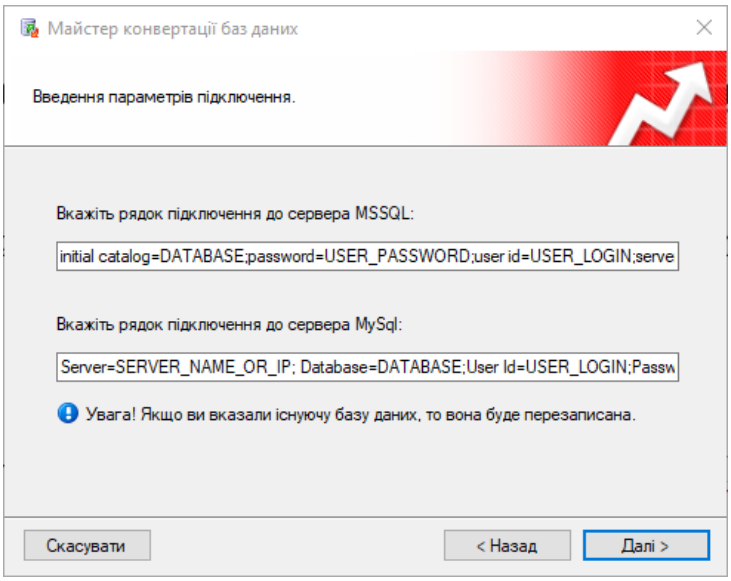

Рисунок 1.6 – Вікно конвертації з одної бази даних в іншу програми FOSS Database Converter

– дозволяє вивантажувати бази даних на диск у форматі текстових файлів (див. рис. 1.7) [1].

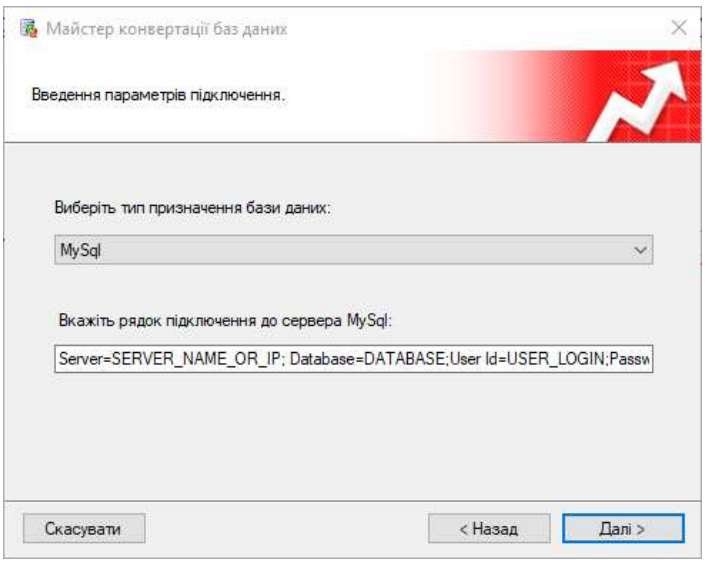

Рисунок 1.7 – Вікно вивантаження бази даних на диск програми FOSS Database Converter

DBF Viewer 2000 (див. рис. 1.8) – це програмне забезпечення, призначене для перегляду, редагування та управління DBF файлами, які часто використовуються у базах даних та різних додатках для зберігання структурованих даних. Це корисний інструмент для тих, хто працює з файлами формату DBF, наприклад, користувачів старих систем управління базами даних або розробників, які обробляють дані з цього формату. DBF Viewer 2000 дає можливість відкривати, переглядати вміст DBF файлів, експортувати цей вміст у формати CSV, Excel, HTML, XML та навпаки. Цей застосунок працює з файлами dBase, Visual dBase, Foxpro, Visual FoxPro і Clipper та іншими старими системами управління базами даних [2].

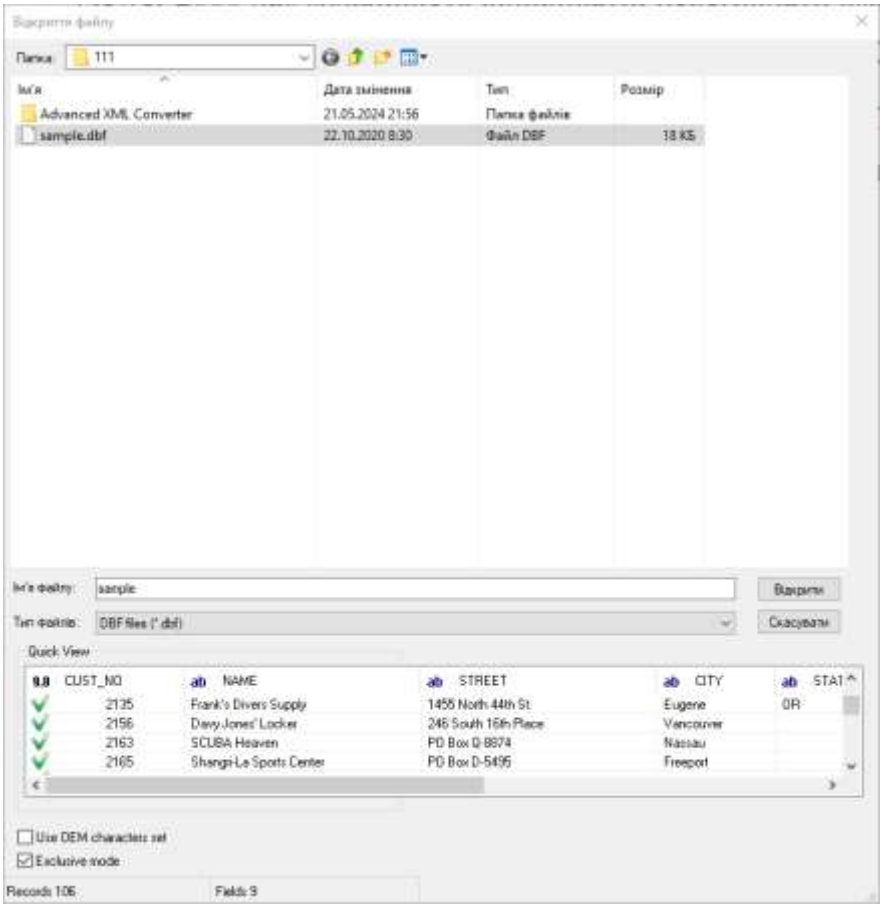

Рисунок 1.8 – Головне вікно програми DBF Viewer 2000

Altova MapForce – це потужний інструмент для графічного проектування та виконання трансформацій даних між різними форматами, включаючи XML,

JSON, бази даних, Excel та веб-сервіси (див. рис. 1.9). Він широко використовується для конвертації даних, інтеграції та автоматизації робочих процесів в різних IT середовищах. Ця програма підтримує такі системи управління базами даних як Firebird, IBM, MariaDB, Microsoft Access, MS SQL, MySQL, Oracle, MongoDB, CouchDB та багато інших. MapForce підтримує розширені перетворення XML між декількома вхідними і декількома вихідними схемами, декількома вихідними і/або цільовими файлами [3].

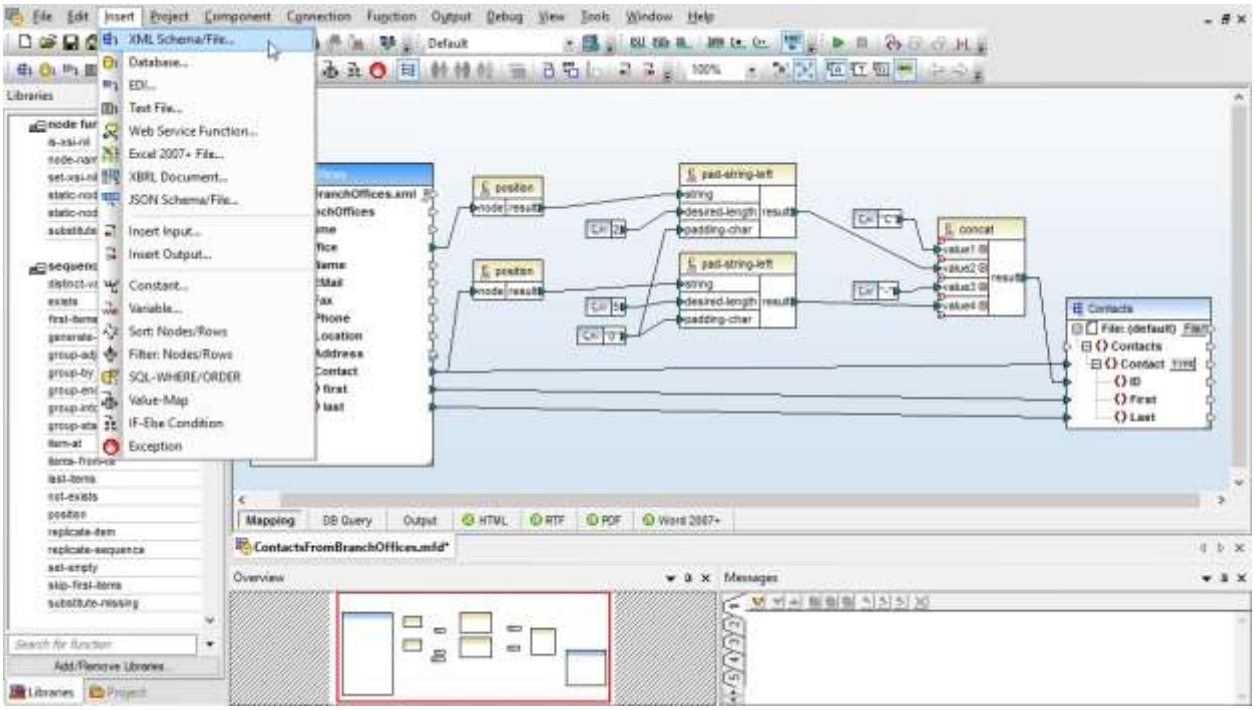

Рисунок 1.9 – Головне вікно програми Altova MapForce

1.2. Вибір програмних засобів для реалізації проекту

При розробці програми-конвертера файлів баз даних в файли XML було важливо обрати оптимальний набір програмних засобів для забезпечення ефективної та надійної роботи програми. У цьому розділі будуть розглянуті використані інструменти та їх вибір.

Visual Studio 2022 обране як інтегроване середовище розробки (Integrated Development Environment, IDE) для проекту програми-конвертера файлів баз даних в файли XML з кількох ключових причин.

Visual Studio 2022 відоме своїм інтуїтивно зрозумілим та дружнім інтерфейсом, який значно спрощує розробку. Його вбудовані інструменти дозволяють розробникам зосередитися на самому процесі розробки, а не витрачати час на розв'язання проблем з інструментами.

Підтримка різних мов програмування є ще однією важливою перевагою Visual Studio 2022. Воно підтримує широкий спектр мов, включаючи C#, що було обрано для даного проекту. Це дозволяє використовувати мову, яка найбільш підходить для вирішення конкретних завдань проекту, забезпечуючи гнучкість та швидкість розробки.

Ще однією перевагою Visual Studio 2022 є широкі можливості налагодження програмного забезпечення. Зокрема, доступність потужних інструментів для відлагодження, таких як крокування по коду, перегляд значень змінних та використання точок зупинки, значно спрощує процес виявлення та виправлення помилок у програмі.

Visual Studio 2022 також відоме своїм широким вибором бібліотек та плагінів, які можуть значно покращити продуктивність розробки. Це дає можливість розширити функціональність IDE за допомогою спеціалізованих інструментів, що відповідають конкретним потребам проекту [4].

C# – це об'єктно-орієнтована мова програмування, створена компанією Microsoft для розробки додатків на платформі .NET. Вона дозволяє розробникам використовувати широкий спектр бібліотек та інструментів .NET для створення потужних, високоякісних додатків [5]. Мову програмування C# було обрано в якості основної мови для розробки програми-конвертера файлів баз даних в файли XML з кількох ключових причин.

C# вирізняється ефективністю та продуктивністю. Вона пропонує сучасні конструкції, такі як механізми обробки винятків, збірка сміття,

лямбда-вирази та інші, що сприяють швидкій та надійній розробці програмного забезпечення.

Також, C# має широку спільноту розробників, яка створюється навколо неї. Це одна з найпопулярніших мов програмування, що привертає до себе велику кількість експертів та ентузіастів. Наявність великої кількості ресурсів, таких як форуми, блоги та бібліотеки, дозволяє розробникам отримувати необхідну підтримку та поради.

Крім того, стандарт .NET Core або .NET 6, який підтримує мову C#, надає можливість розробки програм, що є платформонезалежними. Це означає, що програми, створені з використанням C#, можуть працювати на різних операційних системах, таких як Windows, Linux та macOS, що розширює аудиторію користувачів та забезпечує більшу гнучкість в розгортанні програмного забезпечення. Бібліотека MySql.Data.MySqlClient була обрана для забезпечення зв'язку програми з базою даних MySQL з наступних причин.

1. Підтримка MySQL. MySql.Data.MySqlClient – це офіційна бібліотека для роботи з базами даних MySQL в середовищі .NET. Вона надає повну підтримку для взаємодії з MySQL сервером, включаючи виконання SQL запитів, отримання результатів та управління транзакціями.

2. Надійність та стабільність. Офіційна бібліотека MySql.Data.MySqlClient має високий рівень стабільності та надійності. Вона регулярно оновлюється та підтримується розробниками, що забезпечує виправлення помилок та додавання нових функцій для підтримки останніх версій MySQL сервера.

3. Простота використання. Бібліотека MySql.Data.MySqlClient надає простий та зрозумілий API для взаємодії з базою даних MySQL з допомогою мови програмування C#. Це дозволяє розробникам легко і швидко інтегрувати роботу з базою даних у свої програми.

4. Підтримка основних операцій. MySql.Data.MySqlClient надає можливості для виконання основних операцій з базою даних, таких як вибірка,

вставка, оновлення та видалення даних. Це дозволяє розробникам ефективно взаємодіяти з даними у базі даних MySQL [6].

Windows Forms (WinForms) було обрано для створення користувацького інтерфейсу програми-конвертера файлів баз даних в файли XML з наступних причин.

1. Простота розробки інтерфейсу. WinForms надає простий та інтуїтивно зрозумілий спосіб створення графічного інтерфейсу користувача (GUI) для програм. За допомогою візуального конструктора можна швидко створити інтерфейс, додавши до нього різноманітні елементи, такі як кнопки, тексти, таблиці тощо [7].

2. Підтримка стандартів Windows. WinForms інтегрується з операційною системою Windows та використовує стандартні елементи керування і відображення, що робить інтерфейс програми знайомим та зручним для користувачів, які вже звикли до Windows.

3. Гнучкість налаштувань. WinForms дозволяє швидко та просто налаштувати вигляд та поведінку елементів інтерфейсу за допомогою властивостей. Це дозволяє розробникам створювати інтерфейс, який відповідає конкретним потребам та вимогам програми.

4. Підтримка подій і обробників подій. WinForms надає можливість легко визначати обробники подій для різних елементів інтерфейсу, таких як кнопки або тексти. Це дозволяє реалізувати реакцію програми на дії користувача та взаємодію з програмою.

Простір імен System.IO використовується для роботи з файловою системою і був вибраний у проекті програми-конвертера файлів баз даних в файли XML з наступних причин.

1. Читання та запис файлів. System.IO надає класи та методи для зручного читання та запису файлів. Це дозволяє програмі отримувати доступ до даних у файлах баз даних та зберігати результати конвертації у нових XML-файлах.

2. Робота з шляхами файлів. System.IO дозволяє отримувати інформацію про шляхи файлів, включаючи інформацію про імена файлів, розширення, розмір, час створення та зміни. Це допомагає програмі правильно взаємодіяти з файлами та виконувати необхідні операції.

Простір імен System.Text.RegularExpressions використовується для роботи з регулярними виразами і був включений до проекту програми-конвертера файлів баз даних в файли XML з наступних причин.

1. Пошук та заміна тексту за шаблоном. System.Text.RegularExpressions надає можливості для виконання пошуку та заміни текстових фрагментів у відповідності з визначеними шаблонами регулярних виразів. Це дозволяє програмі виявляти та обробляти структурований текст з високою точністю та ефективністю.

2. Фільтрація та перетворення даних. Використання регулярних виразів дозволяє програмі ефективно фільтрувати та перетворювати дані у відповідності до визначених правил. Наприклад, це може бути корисно для виділення конкретної інформації з текстових даних бази даних перед конвертацією до формату XML.

3. Аналіз та обробка тексту. За допомогою регулярних виразів можна виконувати складний аналіз та обробку текстових даних. Це дозволяє програмі реалізувати різноманітні функції, такі як визначення структури даних у текстових файлах або вилучення певних елементів інформації.

Простір імен System.Xml використовується для роботи з XML-документами і був включений до проекту програми-конвертера файлів баз даних в файли XML з наступних причин.

1. Підтримка роботи з XML. System.Xml містить класи та методи для парсингу, створення, зчитування та запису XML-документів. Це дозволяє програмі-конвертеру ефективно взаємодіяти з XML-структурами даних та виконувати різноманітні операції з ними.

2. Можливості генерації XML. System.Xml дозволяє програмі створювати нові XML-документи або модифікувати існуючі, додаючи,

видаляючи або змінюючи елементи та їх атрибути відповідно до вимог програми.

Простір імен Microsoft.Data.SqlClient використовується для роботи з базами даних SQL Server і був включений до проекту програми-конвертера файлів баз даних у файли формату XML з наступних причин.

1. Підключення до MSSQL Server. Microsoft.Data.SqlClient надає можливості для легкого підключення до серверів SQL Server, включаючи підтримку різних типів автентифікації та шифрування з'єднань. Це дозволяє програмі безпечно та ефективно взаємодіяти з базами даних MSSQL, що є критично важливим для забезпечення надійного доступу до даних.

2. Виконання SQL-запитів. Microsoft.Data.SqlClient включає класи та методи для виконання SQL-запитів, таких як SELECT, INSERT, UPDATE і DELETE. Це дозволяє програмі-конвертеру взаємодіяти з базою даних, зчитувати необхідні дані, виконувати зміни та зберігати результати у форматі XML, що є основним завданням програми.

Простір імен Newtonsoft.Json використовується для роботи з JSON-даними і був включений до проекту програми-конвертера файлів баз даних з наступних причин.

1. Серіалізація та десеріалізація JSON. Newtonsoft.Json надає потужні можливості для серіалізації об'єктів C# у JSON та десеріалізації JSON у об'єкти C#. Це дозволяє програмі-конвертеру легко перетворювати дані з бази даних у формат JSON для зберігання або передачі, а також обробляти JSON-дані для подальшої конвертації в інші формати, такі як XML.

2. Гнучкість та налаштування. Newtonsoft.Json забезпечує високий рівень гнучкості та налаштувань для процесу серіалізації та десеріалізації, включаючи можливості для обробки складних та вкладених структур даних. Це дозволяє програмі ефективно працювати з різноманітними структурами даних, що зберігаються у базі даних, та конвертувати їх у потрібний формат з урахуванням специфічних вимог проекту.

3. Підтримка LINQ to JSON. Newtonsoft.Json включає можливості для використання LINQ-запитів до JSON-даних, що дозволяє програмі виконувати складні запити та фільтрацію даних безпосередньо у форматі JSON. Це спрощує процес обробки даних та дозволяє швидко знаходити необхідну інформацію перед конвертацією у формат XML або інший формат.

Простір імен MongoDB.Driver використовується для роботи з базами даних MongoDB і був включений до проекту програми для взаємодії з базами даних з тої причини, що MongoDB.Driver надає можливості для легкого підключення до серверів MongoDB, включаючи підтримку різних типів автентифікації та шифрування з'єднань. Це дозволяє програмі безпечно та ефективно взаємодіяти з базами даних.

1.3. Висновки до розділу 1

У цьому розділі було здійснено аналіз існуючих програмних рішень для конвертації файлів баз даних у формат XML та вибір інструментів для реалізації проекту конвертера.

Розглянуто кілька онлайн-інструментів для конвертації SQL-файлів у XML, таких як Aspose.app, tableconvert.com, onlinetools.com та anyconv.com. Aspose.app пропонує базові можливості для конвертації без додаткових функцій. Tableconvert.com дозволяє редагувати дані перед конвертацією та підтримує більше форматів. Onlinetools.com спеціалізується на конвертації CSV у XML з можливістю налаштування вхідних та вихідних файлів. AnyConv також, як і Aspose.app, пропонує лише базові можливості для конвертації, але конвертує файли формату DBF.

Проаналізовано також декілька десктопних програм, такі як FOSS Database Converter, DBF Viewer 2000 та Altova MapForce.

FOSS Database Converter відзначається можливістю конвертування баз даних одного провайдера в інший, а також вивантаженням баз даних у форматі

текстових файлів. Цей засіб спрощує процеси міграції даних та забезпечує їх зручне зберігання та обробку.

DBF Viewer 2000 є корисним інструментом для роботи з файлами формату DBF, забезпечуючи їх перегляд, редагування та експорт у різні формати. Цей застосунок особливо корисний для користувачів старих систем управління базами даних, де використовуються такі формати.

Altova MapForce представляє собою потужний інструмент для графічного проектування та виконання трансформацій даних між різними форматами. Він забезпечує розширені можливості конвертації даних та інтеграції робочих процесів у різних IT середовищах.

Кожен з цих інструментів має свою нішу в обробці та конвертації даних, забезпечуючи різні можливості для різних сценаріїв використання.

На основі проведеного аналізу було зроблено висновок, що використання сучасних технологій і бібліотек дозволяє значно спростити процес перетворення даних та забезпечити високу продуктивність розробленого програмного забезпечення. Важливим аспектом є також забезпечення користувацького досвіду за допомогою інтуїтивно зрозумілого інтерфейсу, що підвищує загальну ефективність та зручність використання програми.

Вибрано наступні інструменти для реалізації проекту.

1. Visual Studio 2022 – інтегроване середовище розробки з широкими можливостями налагодження та великою кількістю бібліотек. Воно дозволяє ефективно працювати над проектом і забезпечує зручний інтерфейс для розробки.

2. C# – ця мова програмування обрана завдяки її ефективності, продуктивності та широкій спільноті розробників. C# дозволяє швидко реалізувати потрібний функціонал і забезпечує зручний синтаксис для розробки.

3. MySql.Data.MySqlClient використовується для зв'язку з MySQL, забезпечуючи надійність та простоту використання. Він гарантує надійність та

простоту використання, що є важливим для забезпечення коректної роботи з базою даних.

4. Windows Forms (WinForms) використовується для створення користувацького інтерфейсу. WinForms надає зручні інструменти для створення інтерфейсу користувача, що сприяє зручності взаємодії з програмою.

5. System.IO – для роботи з файловою системою, надаючи можливості для читання та запису файлів. Це важливий інструмент для роботи зі вхідними та вихідними даними.

6. System.Text.RegularExpressions використовується для роботи з регулярними виразами, що дозволяє виконувати складну обробку текстових даних. Він допомагає в забезпеченні правильної обробки та аналізу текстової інформації.

7. System.Xml використовується для роботи з XML-документами, дозволяючи ефективно парсити, створювати та зчитувати XML-документи. Цей інструмент важливий для обміну даними у форматі XML та забезпечує правильну обробку структурованої інформації.

8. Microsoft.Data.SqlClient використовується для підключення та роботи з даними MSSQL.

9. Newtonsoft.Json використовується для роботи з JSON-даними, надаючи потужні можливості для серіалізації об'єктів C# у JSON та десеріалізації JSON у об'єкти C#.

10. MongoDB.Driver використовується для роботи з базами даних MongoDB, надаючи можливості для легкого підключення до серверів MongoDB, включаючи підтримку різних типів автентифікації та шифрування з'єднань.

Цей набір інструментів забезпечує реалізацію проекту конвертера файлів баз даних у XML, задовольняючи вимоги до функціональності та зручності користування.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ КОНВЕРТЕРА ФАЙЛІВ

2.1. Огляд концепції баз даних

Бази даних є невід'ємною частиною сучасних інформаційних систем і грають ключову роль у зберіганні, організації та доступі до даних. Вони забезпечують інфраструктуру для зберігання величезної кількості інформації, яка може бути швидко й ефективно оброблена та використана для різноманітних завдань, від бізнес-аналітики до управління клієнтськими відносинами.

База даних (БД) – це структурована колекція даних, яка зберігається електронним способом. Основною метою баз даних є забезпечення надійного та ефективного способу зберігання й доступу до даних, а також підтримка їхньої цілісності та безпеки. Бази даних можуть зберігати різні типи інформації, включаючи текстові дані, числові дані, зображення, відео та інші форми мультимедійного контенту.

Система управління базами даних (СУБД) – це програмне забезпечення, що забезпечує створення, управління та обслуговування баз даних. СУБД надає користувачам інтерфейс для виконання різноманітних операцій з базою даних, таких як додавання, оновлення, видалення та пошук даних. СУБД також забезпечує захист даних, контроль доступу, відновлення після збоїв та інші функції для підтримки цілісності та надійності баз даних.

Для розуміння роботи баз даних важливо ознайомитися з основними поняттями та термінами.

Таблиця – основна структура даних у реляційних базах даних, що складається з рядків і стовпців. Кожна таблиця містить інформацію про певний об'єкт або явище.

Рядок (запис) – один екземпляр об'єкта або явища, описаного в таблиці. Кожен рядок містить конкретні значення для кожного стовпця.

Стовпець (поле) – атрибут об'єкта або явища, описаного в таблиці. Кожен стовпець містить значення одного типу даних для всіх рядків таблиці.

Ключ – поле або набір полів, які унікально ідентифікують запис у таблиці. Первинний ключ забезпечує унікальність кожного запису, тоді як зовнішній ключ використовується для встановлення зв'язків між таблицями.

Запит – інструкція на мові запитів (зазвичай SQL), яка використовується для отримання, оновлення або видалення даних з бази даних.

2.2. Типи баз даних, їх структура

Існує кілька основних типів баз даних, кожен з яких має свої особливості та використовується для різних завдань. Розглянемо основні типи баз даних [8].

Ієрархічні бази даних.

Ієрархічна база даних організує дані у вигляді ієрархії, де дані розподіляються за рівнями, а кожен рівень має спільну точку зв'язку. Наприклад, університет може бути вищим рівнем, а факультети та адміністрація – нижчими рівнями (див. рис. 2.1).

Рисунок 2.1 – Приклад ієрархічної бази даних

Ця структура добре підходить для даних, які мають чітку ієрархію, наприклад, організаційні структури або каталоги файлів. Вони забезпечують швидкий доступ до даних завдяки чіткій ієрархії, що робить їх простими для реалізації та навігації. Однак вони мають труднощі у модифікації структури даних і складності при необхідності підтримки багатьох зв'язків між записами.

Мережеві бази даних.

Мережева база даних схожа на ієрархічну, але з дозволом на асоціацію дочірніх записів з багатьма батьківськими записами. Це створює мережу файлів бази даних з багатьма зв'язками. Наприклад, студент може бути пов'язаний і з факультетом і з гуртком одночасно (див. рис. 2.2).

Рисунок 2.2 – Приклад мережевої бази даних

В такому типі баз даних легше представляти двонаправлені відносини. Також концептуальна простота сприяє використанню простішої мови управління базою даних. Недолік цього типу полягає у неможливості змінити структуру через її складність та високій структурній залежності.

Об’єктно-орієнтовані бази даних.

Цей тип баз даних спирається на об'єктно-орієнтований підхід до програмування. У цих базах даних інформація зберігається у вигляді об'єктів, які є екземплярами класів в програмному коді. Кожен об'єкт може мати властивості (або поля) та методи (або функції), які визначають його поведінку та можливості (див. рис. 2.3).

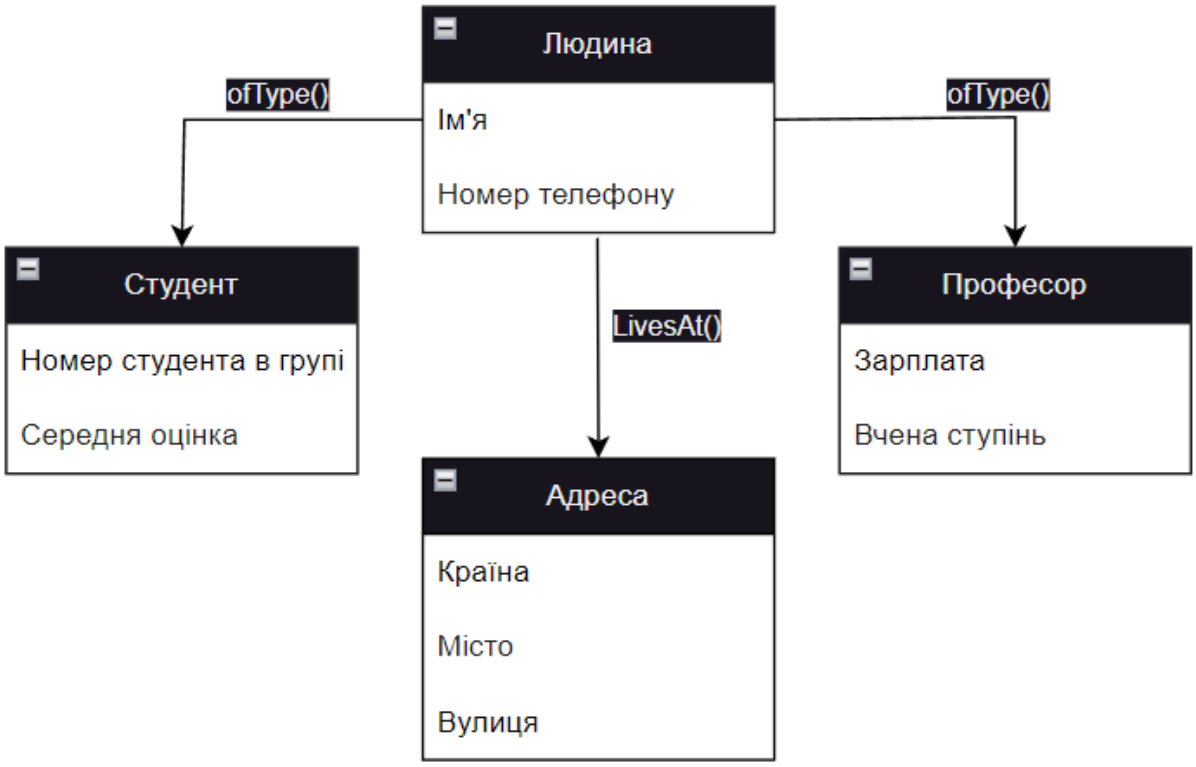

Рисунок 2.3 – Приклад об’єктно-орієнтованої бази даних

Об'єктно-орієнтовані бази даних дозволяють створювати бази даних, які відображають реальні об'єкти та взаємозв'язки між ними, що полегшує розуміння та обробку даних. Такі бази даних інтегруються з об'єктно-орієнтованими мовами програмування і дозволяють зберігати складні дані та їх взаємозв'язки. Недолік цього типу полягає у неефективному використанні ресурсів системи та сповільненні продуктивності через надмірне використання пам'яті або обробку даних для простих задач [9].

Реляційні бази даних.

Вважаються найбільш популярними з усіх баз даних. У цій базі даних кожен рядок даних пов’язаний з іншим рядком за допомогою первинного ключа. Аналогічно, кожна таблиця пов’язана з іншою таблицею за допомогою зовнішнього ключа (див. рис. 2.4).

Завдяки введенню таблиць для організації даних, ця модель стала надзвичайно популярною. Як наслідок, вони широко інтегровані у веб-інтерфейси для використання як ідеальні сховища для даних користувачів. Що робить її ще цікавішою, так це легкість у освоєнні, оскільки мова, що використовується для взаємодії з базою даних, є простою (частіше за все використовується SQL) та легкою для розуміння. Також перевагою цього типу баз даних є те, що у реляційних базах даних масштабування і проходження через дані є досить легким завданням у порівнянні з ієрархічними базами даних.

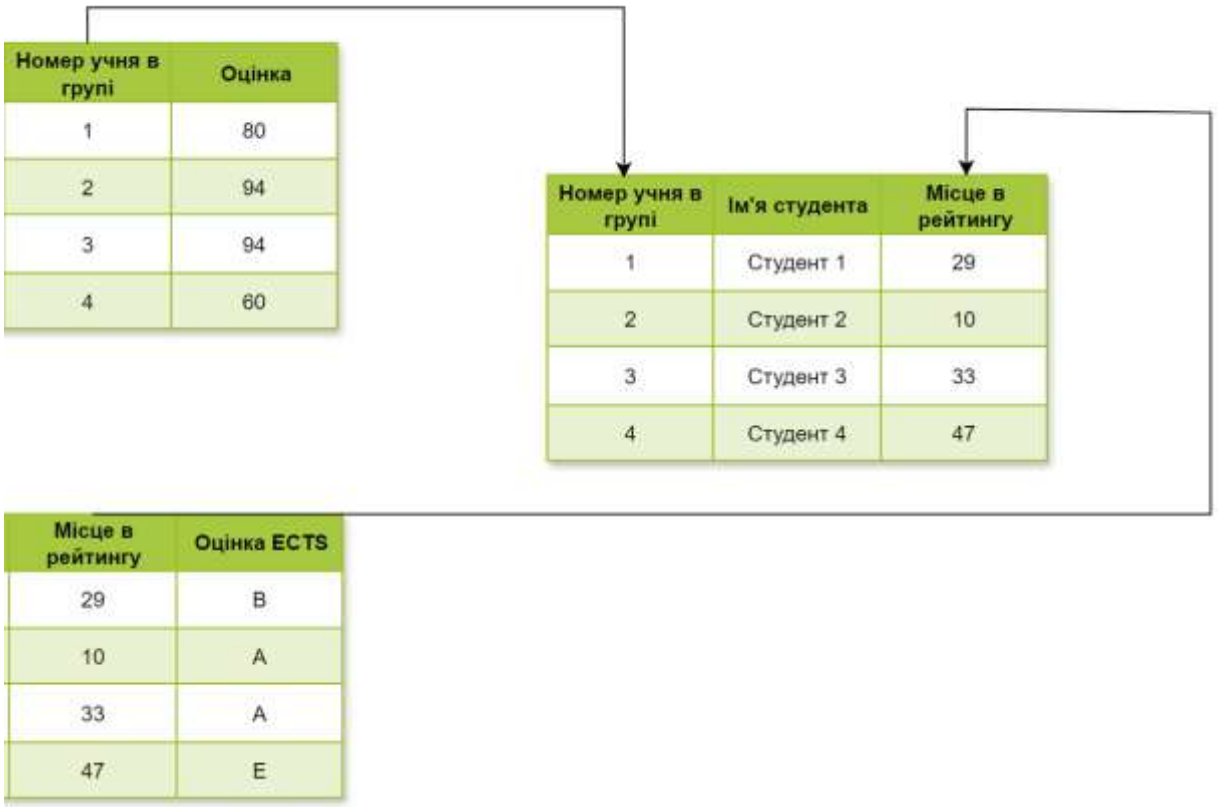

Рисунок 2.4 – Приклад реляційної бази даних

NoSQL.

NoSQL це підхід до проектування баз даних, який дозволяє зберігати та надсилати запити до даних без використання традиційних структур реляційних баз даних. NoSQL бази даних можуть зберігати ті самі типи даних, що й системи керування реляційними базами даних, але роблять це по-іншому. На відміну від типових табличних структур реляційних баз даних, NoSQL бази даних зберігають дані в одному форматі, наприклад, у вигляді документа JSON. Цей підхід не вимагає фіксованої схеми, що дозволяє швидко масштабуватися для управління великими і часто неструктурованими наборами даних. NoSQL також є типом розподілених баз даних, тобто інформація зберігається та дублюється на різних серверах, які можуть бути віддаленими або локальними. Це забезпечує доступність і надійність даних: навіть якщо один сервер виходить з ладу, інші продовжують працювати.

Переваги NoSQL.

1. Економічність. Підтримка високоякісних комерційних систем керування реляційними базами даних може бути дорогою через необхідність придбання ліцензій, навчання менеджерів баз даних та потужного обладнання для вертикального масштабування. NoSQL бази даних дозволяють швидко масштабуватися горизонтально, оптимізуючи використання ресурсів для зниження витрат.

2. Гнучкість. Горизонтальне масштабування та гнучка модель даних дозволяють NoSQL базам даних працювати з великими обсягами даних, що швидко змінюються, що робить їх ідеальними для гнучкої розробки, швидких ітерацій та частих оновлень коду.

3. Реплікація. Функція реплікації NoSQL копіює та зберігає дані на кількох серверах, забезпечуючи надійність та доступність даних навіть у разі виходу серверів з ладу.

4. Швидкість. NoSQL забезпечує швидке та гнучке зберігання й обробку даних для всіх користувачів, включаючи розробників, відділи продажів та

клієнтів. Це робить NoSQL ідеальним для сучасних складних веб-додатків, сайтів електронної комерції та мобільних додатків [10].

Структурована мова запитів (SQL) часто протиставляється до NoSQL. До появи реляційних баз даних компанії використовували ієрархічні системи баз даних із деревоподібною структурою. Однак вони мали обмеження, які призвели до розробки систем управління реляційними базами даних, що використовують SQL для взаємодії з даними. З часом вимоги до швидшої та більш гнучкої обробки великих обсягів даних зросли, особливо для нових технологій, таких як електронна комерція. Це призвело до створення NoSQL баз даних, які стали альтернативою SQL, хоча й не повністю замінили реляційні бази даних. Наприклад, у реляційних базах даних дані про клієнтів, замовлення та продукти зберігаються в окремих таблицях, що об'єднуються через спільні ключі. Цей підхід добре підходить для швидкого зберігання та пошуку даних, але потребує значних ресурсів для вертикального масштабування. У порівнянні, NoSQL бази даних усувають потребу у з'єднанні таблиць, дозволяючи легше горизонтальне масштабування.

2.3. Опис форматів файлів баз даних та їх значення

Формати файлів баз даних визначають спосіб зберігання, організації та доступу до даних. Вони можуть суттєво впливати на продуктивність, ефективність та можливості баз даних. Більшість баз даних використовують власні пропрієтарні формати файлів, але також можуть бути збережені в деяких загальновідомих форматах. Нижче наведено огляд деяких із них.

CSV або Coma separated values – це основний формат файлів для передачі двовимірних даних [11]. Завдяки своїй простоті цей формат часто використовується для передачі табличних даних між різними програмами. Microsoft Excel також посприяв його широкому використанню, дозволивши експортувати дані в цьому форматі. Сьогодні формат CSV більше не є галузевим стандартом, саме через неможливість вбудовування великих обсягів

даних або інформації про дані в один файл. Однак навіть сьогодні ми можемо зустріти цей формат у простих додатках, які не потребують тривимірного представлення даних. У форматі CSV один запис являє собою один рядок. Окремі поля із записами відокремлюються так званим роздільником, де роздільником зазвичай є кома або крапка з комою. У деяких випадках в якості роздільника використовується табуляція, тоді такий варіант позначається скорочено TSV, англійською "tab-separated value". Пробіли і символи табуляції, які можуть з'явитися до або після запису ігноруються. Наприклад, запис Максим , Котельва дасть результат Максим, Котельва. Для збереження пробілів у тексті можна використовувати подвійні лапки. Якщо потрібно вставити в текст запис, який містить кому (роздільник), необхідно взяти його в подвійні лапки. Наприклад, текст "Максим, Котельва" буде вставлено як "Максим, Котельва". Одним з недоліків формату CSV є неможливість автоматичного визначення назв стовпців. У загальному випадку перший рядок у файлі вважається заголовком, який визначає назви стовпців, але автоматично визначити, чи дійсно перший рядок містить назви стовпців, практично неможливо. Ця проблема вирішується зовнішнім втручанням, наприклад, запитом користувача. Формат CSV не позбавлений можливості вставляти нульові значення. Вони розпізнаються, коли після роздільника відсутнє значення, а наступний роздільник продовжується або рядок закінчується.

SQL або Structured Query Language – це основний формат файлів для роботи з реляційними базами даних. Завдяки своїй потужності та універсальності, цей формат використовується для створення, маніпулювання та управління даними в базах даних. SQL забезпечує взаємодію з базами даних за допомогою структурованих запитів, що робить його незамінним інструментом для розробників і адміністраторів баз даних. Сьогодні SQL є галузевим стандартом для реляційних баз даних, підтримується більшістю систем управління базами даних (СУБД), таких як MySQL, PostgreSQL, Microsoft SQL Server та Oracle. Формат SQL-файлів дозволяє зберігати не лише дані, але й структуру бази даних, включаючи визначення таблиць,

індексів, тригерів, функцій та процедур. Це дозволяє легко переносити бази даних між різними системами або створювати резервні копії [12].

У SQL-файлі зберігаються SQL-запити та команди, які можуть включати:

– команди DDL (Data Definition Language) для створення і зміни структури бази даних (CREATE TABLE, ALTER TABLE);

– команди DML (Data Manipulation Language) для вставки, оновлення і видалення даних (INSERT INTO, UPDATE, DELETE);

– команди DCL (Data Control Language) для управління доступом до даних (GRANT, REVOKE);

– команди TCL (Transaction Control Language) для управління транзакціями (COMMIT, ROLLBACK).

SQL підтримує складні запити з використанням умов, об'єднань, групувань і агрегатних функцій. Це дозволяє ефективно отримувати та аналізувати дані з великих наборів даних. Однією з основних переваг формату SQL є його здатність забезпечувати цілісність і послідовність даних. Завдяки транзакціям і обмеженням цілісності, таким як первинні та зовнішні ключі, SQL допомагає підтримувати правильність і зв'язність даних у базах даних.

Незважаючи на свої численні переваги, SQL також має деякі недоліки.

1. Складність управління великими обсягами даних: великі бази даних можуть вимагати значних ресурсів для забезпечення високої продуктивності.

2. Вертикальне масштабування: реляційні бази даних зазвичай краще масштабуються вертикально (збільшення потужності одного сервера), ніж горизонтально (додавання більше серверів), що може обмежувати їх продуктивність у масштабних системах.

3. Структурованість даних: для використання SQL потрібна попередньо визначена схема бази даних, що може бути недоліком у випадках з неструктурованими або змінними даними [12].

JSON або JavaScript Object Notation – це основний формат файлів для передачі структурованих даних. Завдяки своїй гнучкості та простоті читання,

цей формат часто використовується для обміну даними між веб-додатками та серверами. Багато сучасних мов програмування та фреймворків підтримують роботу з JSON, що робить його дуже популярним. Сьогодні формат JSON є галузевим стандартом для передачі даних у веб-додатках завдяки своїй здатності зберігати складні вкладені структури даних у легкому текстовому форматі. Це робить JSON ідеальним для передачі даних, які містять кілька рівнів вкладеності та різні типи даних, такі як рядки, числа, масиви та об'єкти. У форматі JSON дані представляються у вигляді пар "ключ-значення". Ключі є рядками, а значення можуть бути одним з кількох типів даних: рядки, числа, масиви, об'єкти, істини та хибності (boolean), або null. Дані структуруються за допомогою фігурних дужок {} для об'єктів і квадратних дужок [] для масивів. Поля відокремлюються комами, а пари ключ-значення – двокрапками (див. рис. 2.5) [13, 14].

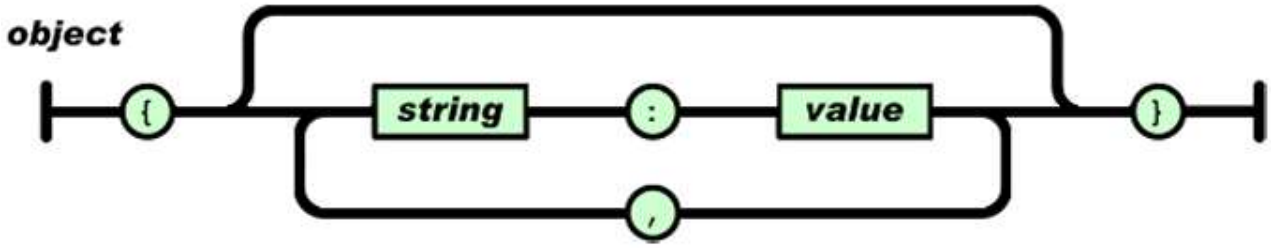

Рисунок 2.5 – Графічне представлення об'єкта JSON

Переваги JSON.

1. Легкість читання та запису. Формат JSON легко читається як людьми, так і машинами. Його синтаксис інтуїтивно зрозумілий, що полегшує його використання.

2. Широка підтримка. JSON підтримується багатьма мовами програмування та бібліотеками, що робить його універсальним для передачі даних між різними системами.

3. Незалежність від платформи. JSON є текстовим форматом, що робить його незалежним від платформи та легко переносимим між різними середовищами.

4. Легкість аналізу. Через простий синтаксис JSON легко аналізувати, що робить його популярним вибором для обміну даними між системами.

Недоліки JSON.

1. Розмір файлу. JSON файли можуть бути більшими за бінарні формати через використання текстового представлення, що може збільшити час передачі даних.

2. Відсутність підтримки простору імен. JSON не підтримує простори імен, що може ускладнити представлення складних структур даних.

3. Чутливість до помилок у синтаксисі. Невірне використання дужок або ком може призвести до помилок під час парсингу файлу.

4. Відсутність типізації. Всі дані в JSON представлені у вигляді рядків, що може призвести до необхідності додаткової обробки для забезпечення правильної типізації даних [15].

2.4. Визначення XML, його основні властивості та структура

Більшість документів в Інтернеті зараз зберігаються і передаються у форматі HTML. Однією з основних переваг HTML є його простота, що сприяє його широкому поширенню серед користувачів. Проте ця простота, можливо, має й свої недоліки, оскільки зростає попит на теги, які спрощують деякі завдання та роблять HTML-документи більш привабливими та динамічними. Щоб задовольнити цей попит, розробники впроваджують специфічні для браузерів HTML-теги. Проте це може ускладнити розробку складних веб-документів, які мають бути доступними на різних платформах. Щоб запобігти цьому розколу, W3C розробила стандарт під назвою eXtensible Markup Language (XML), який міг би зберегти загальну незалежність від додатків.

XML (eXtensible Markup Language) або "Розширювана мова розмітки" – це текстово-орієнтована мова для представлення структурованої інформації. Вона походить від мови SGML (ISO 8879). Метою було створити мову на основі SGML, щоб вона була легко переносимою, прийнятною і оброблюваною, подібно до HTML. XML була розроблена для легкої реалізації і сумісності між мовами SGML і HTML [16].

Кожен XML-документ повинен відповідати строгому синтаксису. Різні частини документа розділені тегами, які називаються елементами. Кожен тег повинен мати початковий і кінцевий тег, подібно до HTML. Мова XML дозволяє вкладати елементи один в одного, що дозволяє представляти багатовимірні дані. Наприклад, зберігання бази даних у XML-файлі може бути організоване як елемент бази даних, який містить кілька елементів таблиць. Елемент таблиці, у свою чергу, міститиме кілька елементів, які представляють окремі записи таблиці (рядки). Кожен елемент рядка міститиме додаткові елементи для кожного запису рядка (комірки). Таким чином, XML-документ каскадується від найбільшої частини, у нашому випадку бази даних, до найменшої частини – комірки таблиці [17].

Елементи позначаються в тексті за допомогою тегів. Більшість елементів мають два теги – початковий і кінцевий. Теги записуються в документі в дужках '<' і '>'. Наприклад, <database> (початковий тег) і </database> (кінцевий тег). Кожен XML-документ повинен мати кореневий елемент, який обгортає весь документ - всі інші елементи є вкладеними в нього. Документ з кількома кореневими елементами буде некоректним. Не всі елементи повинні містити значення, вони можуть бути порожніми.

Окрім значень між тегами, теги можуть містити атрибути. Атрибути використовуються для додавання уточнень або додаткової інформації до окремих елементів. Кожен атрибут складається з назви та значення. Після імені атрибута ставиться знак рівності, а значення береться в подвійні лапки. Кожен тег може мати кілька атрибутів, розділених пробілом [18].

Деякі символи (символи синтаксису розмітки XML) не можуть бути записані як значення тегу або атрибуту. Для таких випадків існують так звані символьні сутності, які дозволяють записувати ці спеціальні символи. Безпосередньо в XML вбудовано лише п'ять таких символьних сутностей (див. нижче). Якщо потрібно визначити додаткові символьні сутності, вони можуть бути визначені у визначенні типу документа [19].

< для <

> для >

& для &

" для “

' для '

Переваги використання XML у порівнянні з іншими форматами даних.

1. Простота. XML є відносно простим стандартом, що складається приблизно з 65 сторінок. Це текстова мова, яка є розбірливою і зрозумілою для людини.

2. Відкритий стандарт і незалежність від платформи/виробника. XML не залежить від платформи та виробника, а також є обмеженою формою SGML, стандарту ISO. Він також базується на ISO 10646, наборі символів Unicode, що забезпечує підтримку текстів у всіх світових алфавітах, включаючи спосіб вказати мову і кодування.

3. Розширюваність. На відміну від HTML, XML є розширюваним, що дозволяє користувачам визначати власні теги для задоволення потреб конкретних додатків.

4. Повторне використання. Розширюваність дозволяє створювати бібліотеки тегів XML один раз і використовувати їх у багатьох додатках.

5. Відокремлення вмісту та представлення. XML відокремлює вміст документа від способу його представлення (наприклад, у браузері). Це полегшує персоналізований перегляд даних, дозволяючи відображати їх у різних форматах і на різних пристроях.

6. Покращене балансування навантаження. Дані можуть бути доставлені в браузер для локальних обчислень, розвантажуючи сервер і досягаючи кращого балансування навантаження.

7. Підтримка інтеграції даних з різних джерел. XML спрощує об'єднання даних з різнорідних джерел. Програмні агенти можуть інтегрувати дані з внутрішніх баз даних та інших додатків для подальшої обробки або презентації [20].

2.5. Висновки до розділу 2

Згідно з поставленою метою роботи у цьому розділі було здійснено огляд концепцій баз даних та основних типів систем управління базами даних (СУБД). Розглянуті різні типи баз даних, включаючи реляційні, ієрархічні, мережеві, об’єктно-орієнтовані, хмарні, централізовані, персональні, операційні та NoSQL бази даних, показано різноманітність підходів до зберігання та обробки даних.

Реляційні бази даних є найбільш поширеними завдяки своїй структурованості та використанню мови SQL для управління даними. Їх основною перевагою є здатність підтримувати складні запити, що дозволяє ефективно обробляти великі обсяги даних та забезпечувати високу цілісність і безпеку.

Ієрархічні бази даних організовують дані у вигляді дерева, де кожен запис має один батьківський і може мати багато дочірніх записів. Цей тип баз даних є простим у реалізації, проте має обмеження у масштабуванні та складності модифікацій структури даних.

Мережеві бази даних подібні до ієрархічних, але дозволяють дочірнім записам мати кілька батьківських записів, що створює більш гнучку структуру даних. Однак, складність управління та залежність від структури можуть стати перешкодою для їх використання.

Об’єктно-орієнтовані бази даних дозволяють зберігати дані у вигляді об’єктів, що є екземплярами класів в програмному коді. Це спрощує відображення реальних об’єктів та їх взаємозв’язків, проте може бути неефективним при виконанні простих завдань через надмірне використання ресурсів.

NoSQL бази даних є альтернативою традиційним реляційним базам даних та використовують різні підходи до зберігання даних, наприклад, у вигляді документів JSON. Вони дозволяють ефективно працювати з великими обсягами неструктурованих даних, забезпечуючи гнучкість, горизонтальне масштабування та високу швидкість обробки запитів. NoSQL бази даних є економічно вигідними завдяки можливості використовувати дешевші ресурси та забезпечувати високу доступність даних через реплікацію на різних серверах.

Узагальнюючи, реляційні бази даних пропонують високу структурованість та надійність для виконання складних запитів, але можуть вимагати значних ресурсів для вертикального масштабування. NoSQL бази даних, з іншого боку, забезпечують гнучкість та можливість горизонтального масштабування, що робить їх ідеальними для сучасних веб-додатків та мобільних застосунків. Вибір між реляційними та NoSQL базами даних залежить від специфічних потреб та вимог проекту. Для проектів, які потребують високого рівня структурованості, цілісності та складних запитів, реляційні бази даних є оптимальним вибором. Для проектів з великими обсягами неструктурованих даних, що швидко змінюються, NoSQL бази даних забезпечать необхідну гнучкість та продуктивність.

Додатково було розглянуто формати файлів, що використовуються для зберігання та передачі даних у базах даних. Формат CSV (Comma Separated Values) є одним з найбільш поширених завдяки своїй простоті та сумісності з багатьма програмами, такими як Microsoft Excel. Однак він має обмеження у вигляді нездатності вбудовувати метадані та великий обсяг даних у один файл.

Формат JSON (JavaScript Object Notation) популярний для передачі структурованих даних між веб-додатками та серверами завдяки своїй гнучкості та читабельності. Він підтримується багатьма мовами програмування, що робить його універсальним інструментом для обміну даними. Проте, JSON файли можуть бути більшими за бінарні формати, що збільшує час передачі даних.

XML (eXtensible Markup Language) є текстово-орієнтованою мовою для представлення структурованої інформації. Його властивості та синтаксис дозволяють легко переносити дані між платформами та додатками. XML дозволяє створювати ієрархічні структури даних, використовуючи елементи та атрибути. Порівняно з іншими форматами даних, XML має такі переваги, як простота, відкритість, розширюваність, можливість повторного використання, відокремлення вмісту та представлення, покращене балансування навантаження та підтримка інтеграції даних з різних джерел.

Згідно з проаналізованою інформацією, було обрано основні формати файлів та бази даних, з яких буде реалізовано конвертація у XML. Основні формати файлів, що будуть підтримуватися програмою для конвертації у XML, включають файли форматів sql, json та дані, взяті з таких СУБД, як MySQL, MSSQL, MariaDB, MongoDB.

У відповідності до поставленої мети, в кваліфікаційній роботі ставилися та вирішувалися наступні завдання.

1. Проаналізувати вже розроблені програмні рішення за тематикою.
2. Проаналізувати та обрати інструменти для реалізації конвертації файлів.
3. Реалізувати алгоритми для конвертації файлів.
4. Спроектувати приємний інтерфейс та зрозумілий інтерфейс для користувача.

У рамках даної кваліфікаційної роботи була поставлена наступна задача дослідження: розробити програму-конвертер, яка здатна перетворювати файли баз даних різних форматів у файли формату XML.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ-КОНВЕРТЕРУ ФАЙЛІВ БАЗ ДАНИХ У ФАЙЛИ ФОРМАТУ XML

3.1. Розробка загальної архітектури конвертера файлів

Цей розділ присвячений розробці програми-конвертера для перетворення файлів баз даних у формат XML. У сучасному світі обробка та перенесення даних між різними системами є важливим аспектом інформаційних технологій. Програма, описана в цьому розділі, призначена для експорту даних з бази даних MySQL, MSSQL, MariaDB, MongoDB, SQL-файлів та JSON-файлів у формат XML, забезпечуючи користувачів ефективним інструментом для перенесення та резервного копіювання даних.

Основні вимоги до програми включають наступне.

1. Підключення до баз даних MySQL, MSSQL, MariaDB та MongoDB: програма повинна мати можливість підключатися до вказаних баз даних MySQL за допомогою параметрів підключення, введених користувачем.

2. Експорт даних із зазначених баз даних у XML: програма повинна виконувати SQL-запити для отримання даних з бази даних і конвертувати ці дані у формат XML.

3. Читання та конвертація SQL-файлів та JSON-файлів: програма повинна вміти читати SQL-файли, витягати з них дані та конвертувати ці дані у формат XML.

4. Збереження XML-файлів: програма повинна дозволяти користувачу зберігати згенеровані XML-файли на локальному диску.

У розробці програми-конвертера для перетворення даних з баз даних та файлів у формат XML використовуються кілька ключових бібліотек, кожна з яких відіграє важливу роль у забезпеченні функціональності програми (див. Додаток А). Нижче наведено детальний опис використання кожної з цих бібліотек.

Бібліотека MySQL.Data для мови програмування C# є набором інструментів, які дозволяють розробникам взаємодіяти з базами даних MySQL. Його основне завдання полягає у наданні засобів для підключення до бази даних MySQL, виконання SQL-запитів, а також отримання і обробки результатів цих запитів.

Функції MySQL.Data, використані у розробці програми.

1. Підключення до бази даних. За допомогою класу MySqlConnection програма встановлює з'єднання з базою даних MySQL використовуючи такі введені користувачем параметри: адреса сервера, ім'я користувача, пароль та назва бази даних.

2. Виконання SQL-запитів. За допомогою класу MySqlCommand програма виконує SQL-запит до бази даних з метою отримати інформацію, що запитується.

3. Обробка результатів запитів. Отримані дані з бази даних зберігаються у вигляді DataTable, що полегшує їх подальшу обробку і перетворення у формат XML [6].

System.Xml – це стандартна бібліотека .NET для роботи з XML-документами. Вона надає різноманітні засоби для створення, редагування та збереження XML-документів.

Основні функції System.Xml, використані у розробці програми.

1. Створення XML-документів. За допомогою класу XmlDocument програма створює новий XML-документ.

2. Редагування XML-документів. У створений XML-документ за допомогою методу CreateElement класу створюється спочатку кореневий елемент XML, в якому створюються елементи для кожного запису з бази даних або SQL-файлу. Метод AppendChild додає вказаний елемент до кінця списку дочірніх елементів вузла, до якого викликається цей метод. Цей метод забезпечує побудову ієрархічної структури XML-документа. Ім'я кожного елемента відповідає назві колонки або атрибуту.

3. Збереження XML-документів. Метод Save зберігає XML-документ у файл. Він використовується для збереження остаточного XML-документа у файл після заповнення його даними.

System.Text.RegularExpressions – це бібліотека для роботи з регулярними виразами в .NET. Регулярні вирази дозволяють виконувати складні операції з пошуку і обробки тексту, що робить цю бібліотеку незамінною для задач парсингу [21].

Основні функції System.Text.RegularExpressions, використані у розробці програми.

Пошук та виділення частини тексту. За допомогою методу Matches класу Regex програма виконує пошук усіх збігів у вхідному рядку, які відповідають заданому шаблону регулярного виразу. Результатом є колекція об'єктів Match, кожен з яких представляє окремий збіг [22].

Бібліотека System.IO надає можливості для роботи з файлами, потоками та введенням/виведенням даних у .NET. Бібліотека Windows Forms у .NET використовується для створення графічного інтерфейсу користувача (GUI). Вона забезпечує набір візуальних елементів, які дозволяють розробникам створювати зручні та інтуїтивно зрозумілі інтерфейси для взаємодії з користувачем. У розробленій програмі-конвертері файлів баз даних у формат XML використовується кілька ключових компонентів WinForms для реалізації інтерфейсу.

Основні функції System.IO [23] та Windows Forms [24], використані у розробці програми.

1. Зчитування вмісту. Клас StreamReader використовується для зчитування вмісту SQL-файлу, який містить дані для конвертації.

2. Діалогове вікно збереження. Клас SaveFileDialog надає користувачеві можливість вибрати ім'я та місце збереження файлу XML.

3. Діалогове вікно відкриття файлу. Клас OpenFileDialog використовується для надання користувачеві можливості вибрати SQL-файл для зчитування даних.

Бібліотека Microsoft.Data.SqlClient для мови програмування C# є набором інструментів, які дозволяють розробникам взаємодіяти з базами даних MSSQL. Його основне завдання полягає у наданні засобів для підключення до бази даних MSSQL, виконання SQL-запитів, а також отримання і обробки результатів цих запитів.

Основні функції Microsoft.Data.SqlClient, використані у розробці програми.

1. Підключення до бази даних. За допомогою класу SqlConnection програма встановлює з'єднання з базою даних MSSQL використовуючи такі введені користувачем параметри: адреса сервера, ім'я користувача, пароль та назва бази даних.

2. Виконання SQL-запитів. За допомогою класу SqlCommand програма виконує SQL-запит до бази даних з метою отримати інформацію, що запитується.

3. Обробка результатів запитів. Отримані дані з бази даних зберігаються у вигляді DataTable, що полегшує їх подальшу обробку і перетворення у формат XML.

Бібліотека MongoDB.Driver для мови програмування C# є набором інструментів, які дозволяють розробникам взаємодіяти з базами даних MongoDB. Його основне завдання полягає у наданні засобів для підключення до бази даних MongoDB, виконання запитів, а також отримання і обробки результатів цих запитів.

Функції MongoDB.Driver, використані у розробці програми.

1. Підключення до бази даних. За допомогою класу MongoClient програма встановлює з'єднання з базою даних MongoDB використовуючи такі введені користувачем параметри: адреса сервера, порт та назва бази даних.

2. Виконання запитів. За допомогою методу Find програма виконує запит до колекції бази даних з метою отримати всі документи.

3. Обробка результатів запитів. Отримані документи з бази даних перетворюються у XML-елементи для подальшого збереження у форматі XML.

Бібліотека Newtonsoft.Json для мови програмування C# є набором інструментів, які дозволяють розробникам взаємодіяти з JSON-даними. Його основне завдання полягає у наданні засобів для парсингу JSON, серіалізації об'єктів у JSON та десеріалізації JSON у об'єкти C#.

Основні функції Newtonsoft.Json, використані у розробці програми.

1. Зчитування JSON-даних з файлу. За допомогою класу File програма зчитує вміст JSON-файлу, шлях до якого вказано користувачем.

2. Парсинг JSON-даних. За допомогою класу JArray з бібліотеки Newtonsoft.Json програма парсить JSON-рядок у масив JSON-об'єктів.

3. Обробка JSON-даних та перетворення у XML. Отримані JSON-об'єкти перетворюються у XML-елементи для подальшого збереження у форматі XML.

В програмі реалізовано алгоритми конвертації даних в XML: з баз даних MySQL, MSSQL, MariaDB, MongoDB у XML та з файлів .sql та .json у XML. Розглянемо їх детальніше.

Конвертація з MySQL, MSSQL та MariaDB в XML.

1. Користувач вводить дані про сервер, користувача, базу даних, пароль та назву таблиці бази даних через відповідні текстові поля у програмному інтерфейсі.

2. Програма встановлює з'єднання з базою даних, використовуючи введені користувачем параметри підключення (адреса сервера, ім'я користувача, пароль, назва бази даних).

3. Після успішного з'єднання програма виконує SQL-запит до бази даних для отримання всієї інформації з вказаної таблиці.

4. Отримані дані перетворюються у формат XML. Кожен рядок даних із таблиці стає окремим записом у XML-структурі. Для цього програма створює XML-елементи для кожного поля (стовпця) і кожного рядка (запису) в таблиці.

5. Згенерований XML-документ зберігається на локальному диску. Програма використовує діалогове вікно збереження файлу, де користувач може обрати розташування та ім'я для збереження XML-файлу.

Конвертація з MongoDB в XML.

1. Користувач вводить дані про сервер, порт, базу даних та назву колекції MongoDB через відповідні текстові поля у програмному інтерфейсі.

2. Програма встановлює з'єднання з базою даних MongoDB, використовуючи введені користувачем параметри підключення (адреса сервера, порт, назва бази даних).

3. Після успішного з'єднання програма виконує запит до колекції бази даних для отримання всіх документів з вказаної колекції.

4. Отримані документи перетворюються у формат XML. Кожен документ з колекції стає окремим записом у XML-структурі. Для цього програма створює XML-елементи для кожного поля (ключа) і кожного документа в колекції.

5. Згенерований XML-документ зберігається на локальному диску. Програма використовує діалогове вікно збереження файлу, де користувач може обрати розташування та ім'я для збереження XML-файлу.

Конвертація з SQL-файлу в XML.

1. Користувач вказує шлях до SQL-файлу через діалогове вікно у програмному інтерфейсі.

2. Програма зчитує вміст SQL-файлу, знаходить найдовший рядок, який містить дані, і визначає його як зразок для подальшого аналізу.

3. За допомогою регулярних виразів програма аналізує цей зразок, визначає атрибути, які містяться у файлі.

4. Користувачу пропонується ввести назву для кожного атрибуту.

5. Для кожного рядка даних у файлі програма створює відповідний запис у форматі XML, де назви полів відображаються як теги, а значення – як вміст тегів.

6. Сформований XML-документ зберігається на локальному диску за допомогою діалогового вікна збереження файлу, аналогічно до першого алгоритму.

Конвертація з JSON-файлу в XML.

1. Користувач вказує шлях до JSON-файлу через діалогове вікно у програмному інтерфейсі.

2. Програма зчитує вміст JSON-файлу у рядок.

3. Програма створює кореневий елемент XML-документа, який буде містити всі записи.

4. Програма парсить JSON-рядок у масив JSON-об'єктів за допомогою бібліотеки Newtonsoft.Json.

5. Отримані JSON-об'єкти перетворюються у формат XML. Кожен JSON-об'єкт стає окремим записом у XML-структурі. Для цього програма створює XML-елементи для кожного поля (властивості) і кожного JSON-об'єкта.

6. Згенерований XML-документ зберігається на локальному диску. Програма використовує діалогове вікно збереження файлу, де користувач може обрати розташування та ім'я для збереження XML-файлу.

3.2. Опис графічного інтерфейсу програми.

Інтерфейс програми складається з одного основного вікна (див. рис. 3.1), яке змінюється під час вибору алгоритму конвертації, та додаткового вікна для введення назви атрибута, яке використовується при виборі конвертації з SQL-файлу в файл XML.

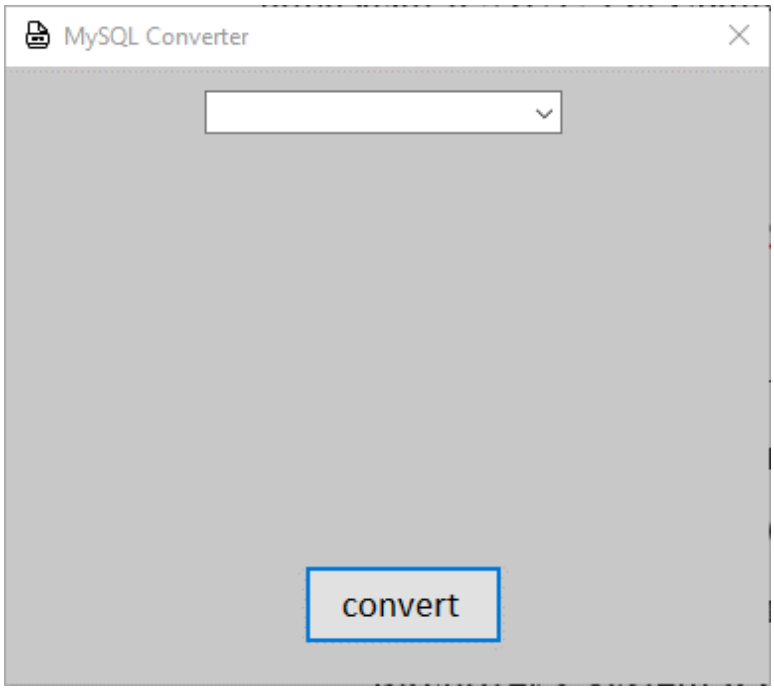

Рисунок 3.1 – Вигляд головного вікна програми

У основному вікні програми відображається випадний список, реалізований за допомогою елементу ListBox у Windows Forms, для вибору алгоритму конвертації та кнопка для власне конвертації, реалізована за допомогою елементу Button у Windows Forms. Алгоритм конвертації обирається у випадному списку, яке знаходиться у верхній частині програми (див. рис. 3.2). У списку є шість варіантів вибору.

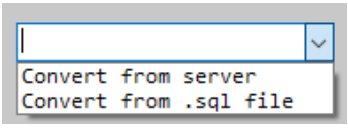

Рисунок 3.2 – Вигляд випадного списку

При виборі першого варіанту, а саме «Convert from MySQL» (див. рис. 3.3), у головному вікні програми нижче за випадний список з'являться п'ять полів для введення, реалізовані за допомогою елементу TextBox у Windows Forms та підписи поряд з ними, що зазначають який саме параметр вводити у відповідне поле, реалізовані за допомогою елементу Label у Windows Forms.

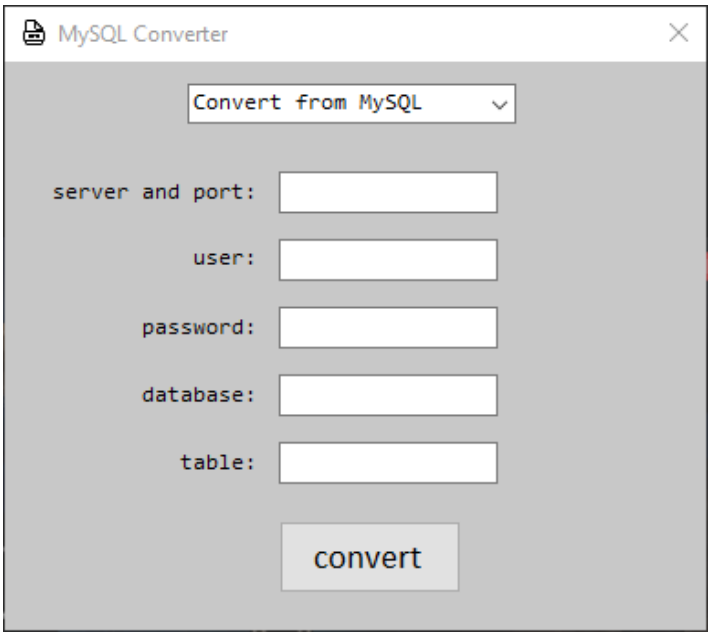

Рисунок 3.3 – Вигляд головного вікна під час вибору опції «Convert from MySQL»

При виборі другого варіанту, а саме «convert from .sql file» (див. рис. 3.4), у головному вікні програми посередині з'явиться поле для введення шляху до файлу, реалізоване за допомогою елементу TextBox у Windows Forms та кнопку, що відкриває стандартне діалогове вікно для вибору файлу.

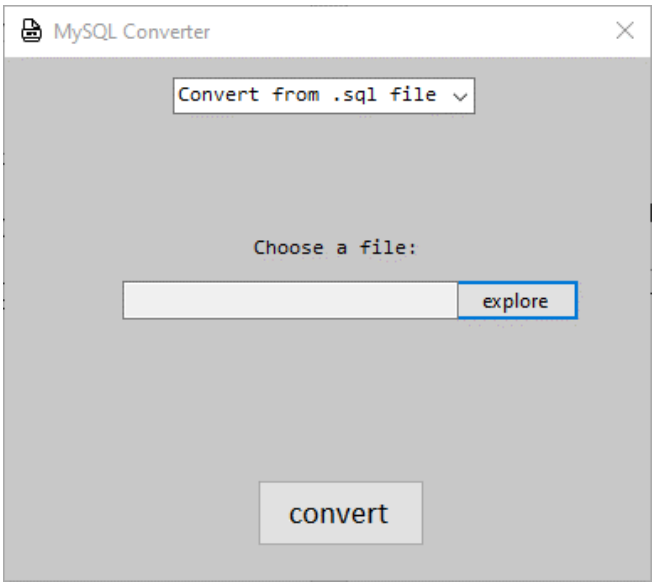

Рисунок 3.4 – Вигляд головного вікна під час вибору опції «convert from .sql file»

При виборі файлу за допомогою другого варіанту текст у полі для введення шляху заміниться на шлях до обраного в діалоговому вікні файлу.

Після обрання файлу та натиснення на кнопку «convert» відкривається додаткове вікно (див. рис. 3.5), яке пропонує ввести в TextBox назву для атрибуту.

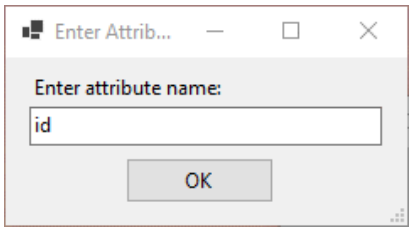

Рисунок 3.5 – Вигляд додаткового вікна

При виборі третього варіанту, а саме «Convert from MongoDB» (див. рис. 3.6), у головному вікні програми нижче за випадний список з'являться чотири поля для введення: host, port, database, table та підписи поряд з ними, що зазначають який саме параметр вводити у відповідне поле.

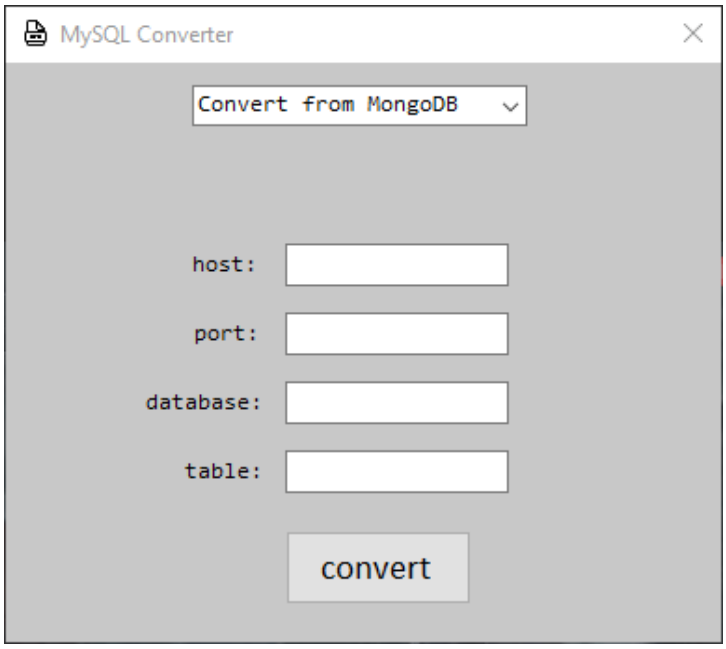

Рисунок 3.6 – Вигляд головного вікна під час вибору опції «Convert from MongoDB»

При виборі четвертого варіанту, а саме «Convert from MSSQL» (див. рис. 3.7), у головному вікні програми нижче за випадний список з'являться п'ять полів для введення та підписи поряд з ними, що зазначають який саме параметр вводити у відповідне поле: server – для введення адреси сервера, user – для введення логіну користувача, password – для введення паролю користувача, database – для введення назви бази даних, з якої потрібно обирати таблицю, table – для введення назви таблиці, дані з якої потрібно конвертувати.

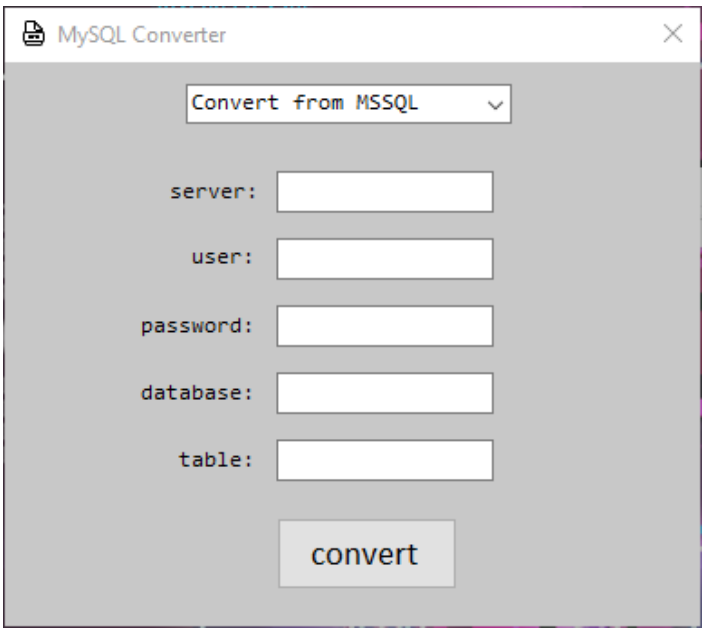

Рисунок 3.7 – Вигляд головного вікна під час вибору опції «Convert from MSSQL»

При виборі п’ятого варіанту, а саме «Convert from MariaDB» (див. рис. 3.8), у головному вікні програми нижче за випадний список з'являться п'ять полів для введення та підписи поряд з ними, що зазначають який саме параметр вводити у відповідне поле (server and port, user, password, database, table).

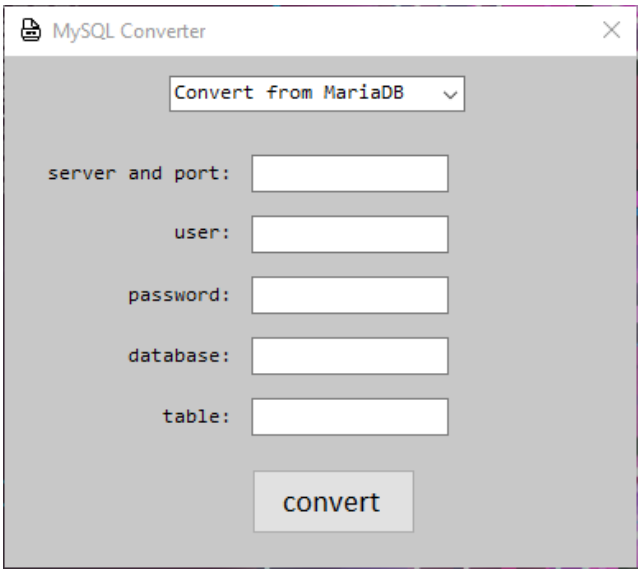

Рисунок 3.8 – Вигляд головного вікна під час вибору опції «Convert from MariaDB»

При виборі шостого варіанту, а саме «Convert from .json file» (див. рис. 3.9), у головному вікні програми посередині з'явиться поле для введення шляху до файлу, реалізоване за допомогою заблокованого до редагування елементу TextBox у Windows Forms та кнопку, що відкриває стандартне діалогове вікно для вибору файлу.

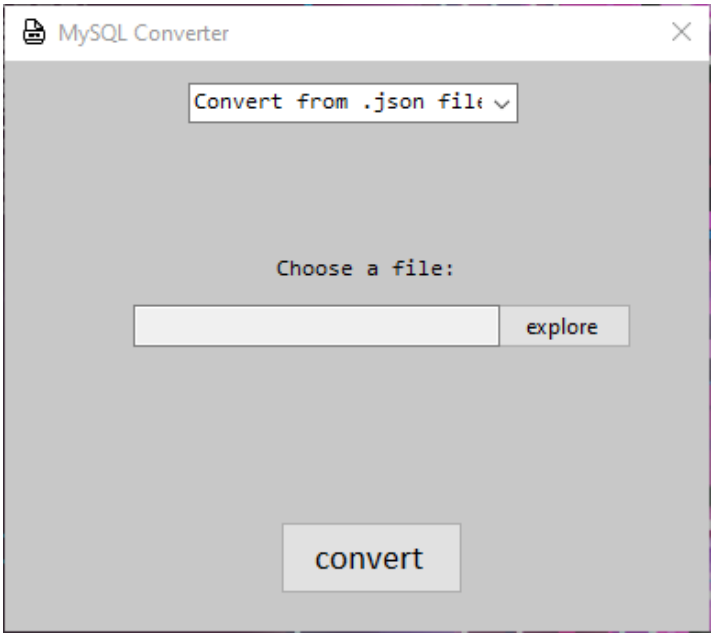

Рисунок 3.9 – Вигляд головного вікна під час вибору опції «Convert from .json file»

3.3. Сценарій роботи користувача з системою

Для коректної роботи програмного продукту персональний комп’ютер має відповідати таким мінімальним системним та апаратним вимогам:

– встановлена операційна система Windows 7 32 або 64-розрядна;

– процесор із тактовою частотою 2 ГГц або швидший – 32-розрядний (x86) або 64-розрядний (x64);

– оперативна пам’ять 2 гігабайт (ГБ) (для 32-розрядної версії) або 4 ГБ (для 64-розрядної версії).

Перед запуском, програму потрібно розпакувати у будь-яке зручне місце та запустити її через ярлик Converter або напряму через graduation_project.exe, що знаходиться у кореневій теці. Після запуску програми користувач бачить головне вікно програми (див. рис. 3.1).

Далі користувачу потрібно обрати один з варіантів конвертації у випадному списку (див. рис. 3.2): «Convert from MySQL» (конвертує дані, підключившись до сервера MySQL), «Convert from .sql file» (конвертує дані з .sql файлу), «Convert from MongoDB» (конвертує дані, підключившись до сервера MongoDB), «Convert from MSSQL» (конвертує дані, підключившись до сервера MSSQL), «Convert from MariaDB» (конвертує дані, підключившись до сервера MariaDB), «Convert from .json file» (конвертує дані з .json файлу).

Якщо користувач обирає пункт «Convert from MySQL», в головному вікні з’являться такі поля (див. рис. 3.3), як «server and port» (вводиться адреса серверу двокрапка і порт серверу), «user», «password», «database», куди потрібно вводити відповідні дані для рядка підключення до бази даних, та «table», куди потрібно ввести назву таблиці в базі даних, з якої потрібно конвертувати дані.

Після заповнення всіх полів (див. рис. 3.10) користувачу потрібно натиснути кнопку «convert», після чого відкриється діалогове вікно збереження файлу (див. рис. 3.11), де можна обрати куди зберегти файл і ввести назву файлу.

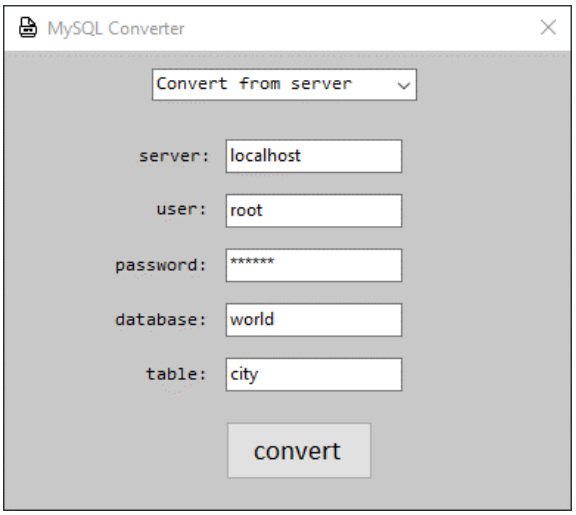

Рисунок 3.10 – Вигляд головного вікна під час вибору опції «Convert from MySQL» після введення всіх даних

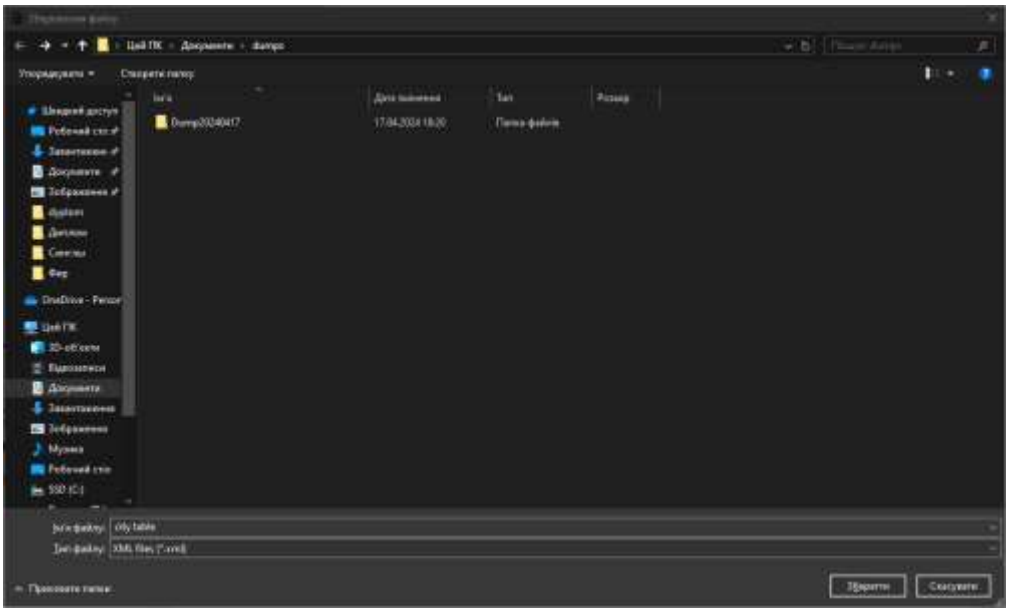

Рисунок 3.11 – Вигляд діалогового вікна збереження

Якщо користувач обирає пункт «convert from .sql file», в головному вікні з’явиться поле та кнопка (див. рис. 3.4), яка відкриває діалогове вікно для вибору .sql файлу (див. рис. 3.12).

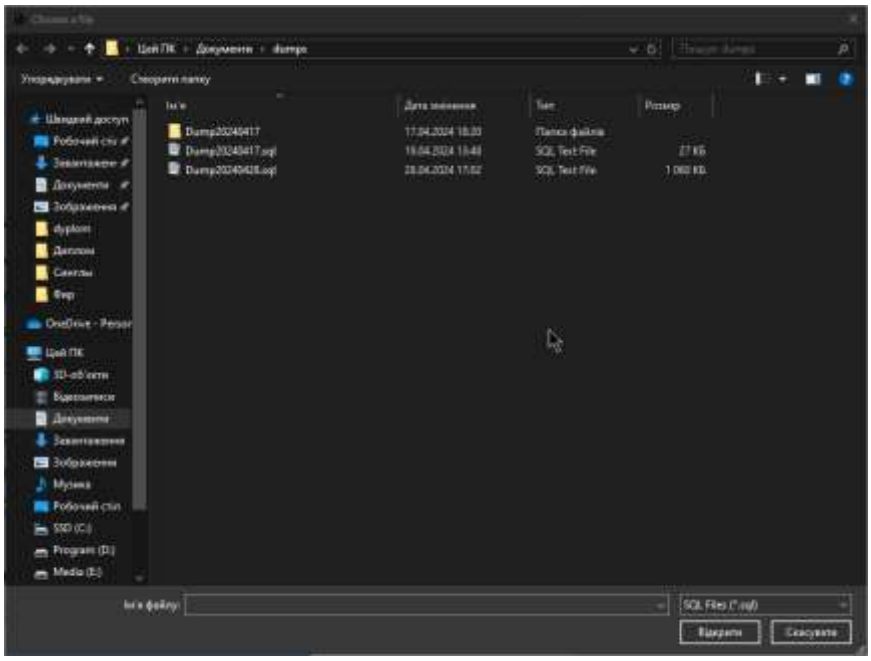

Рисунок 3.12 – Вигляд діалогового вікна вибору файлу

Після вибору файлу користувачу потрібно натиснути кнопку «convert», після чого відкриється додаткове вікно (див. рис. 3.5) із полем для вводу назви атрибуту. Після введення назви відкриється аналогічне вікно для 2 атрибуту і так далі, поки не буде введено імена для всіх атрибутів.

Після введення імен для всіх атрибутів користувачу відкриється діалогове вікно збереження файлу (див. рис. 3.7), де можна обрати куди зберегти файл і ввести назву файлу.

Якщо користувач обирає пункт «Convert from MongoDB», в головному вікні з’являться такі поля (див. рис. 3.6), як «host», «port», «database», куди потрібно вводити відповідні дані для рядка підключення до бази даних, та «table», куди потрібно ввести назву таблиці в базі даних, з якої потрібно конвертувати дані.

Після заповнення всіх полів (див. рис. 3.13) користувачу потрібно натиснути кнопку «convert», після чого відкриється діалогове вікно збереження файлу (див. рис. 3.11), де можна обрати куди зберегти файл і ввести назву файлу.

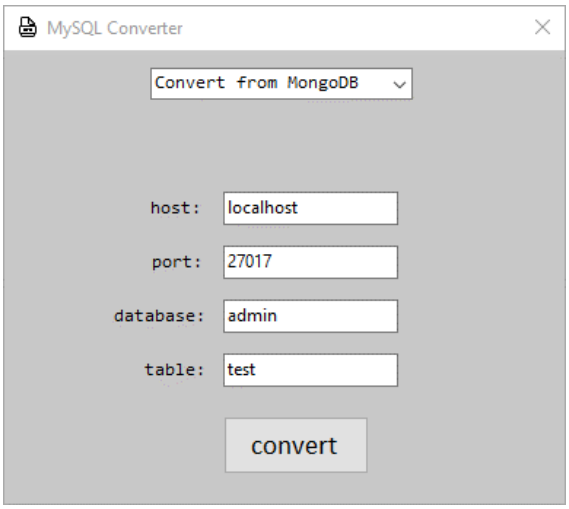

Рисунок 3.13 – Вигляд головного вікна під час вибору опції «Convert from MongoDB» після введення всіх даних

Якщо користувач обирає пункт «Convert from MSSQL», в головному вікні з’являться такі поля (див. рис. 3.7), як «server», «user», «password», «database», куди потрібно вводити відповідні дані для рядка підключення до бази даних, та «table», куди потрібно ввести назву таблиці в базі даних, з якої потрібно конвертувати дані. Після заповнення всіх полів (див. рис. 3.14) користувачу потрібно натиснути кнопку «convert», після чого відкриється діалогове вікно збереження файлу (див. рис. 3.11), де можна обрати куди зберегти файл і ввести назву файлу.

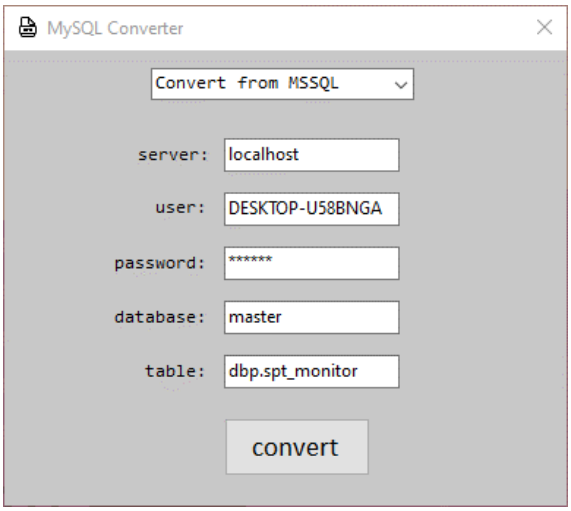

Рисунок 3.14 – Вигляд головного вікна під час вибору опції «Convert from MSSQL» після введення всіх даних

Якщо користувач обирає пункт «Convert from MariaDB», в головному вікні з’являться такі поля (див. рис. 3.8), як «server and port» (у форматі адреса:порт), «user», «password», «database», куди потрібно вводити відповідні дані для рядка підключення до бази даних, та «table», куди потрібно ввести назву таблиці в базі даних, з якої потрібно конвертувати дані.

Після заповнення всіх полів (див. рис. 3.15) користувачу потрібно натиснути кнопку «convert», після чого відкриється діалогове вікно збереження файлу (див. рис. 3.11), де можна обрати куди зберегти файл і ввести назву файлу.

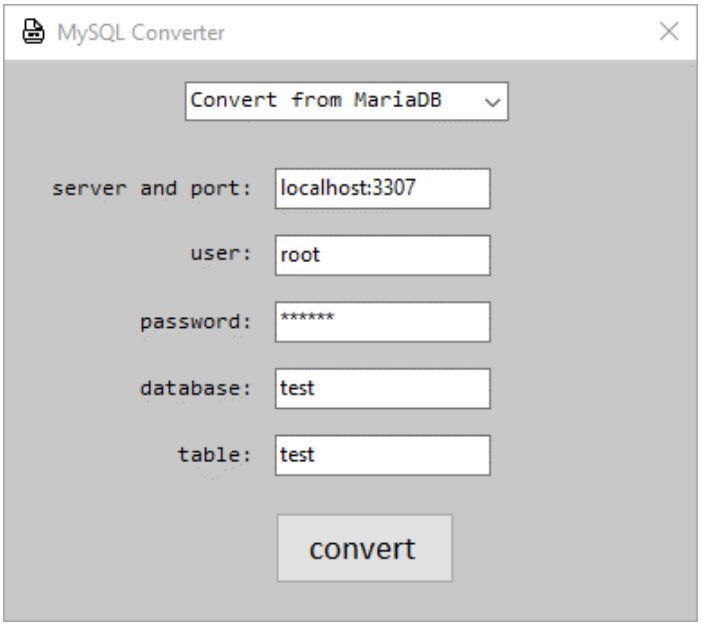

Рисунок 3.15 – Вигляд головного вікна під час вибору опції «Convert from MariaDB» після введення всіх даних

Якщо користувач обирає пункт «convert from .json file», в головному вікні з’явиться поле та кнопка (див. рис. 3.9), яка відкриває діалогове вікно для вибору .json файлу. Після вибору файлу користувач повинен натиснути кнопку «convert», щоб конвертувати обраний файл.

Для того, щоб закінчити роботу з програмою, користувачу потрібно натиснути хрестик у правому верхньому куті вікна.

3.4. Висновки до розділу 3

Програма-конвертер для перетворення файлів баз даних у формат XML, розроблена в рамках цього проекту, демонструє ефективне використання різних бібліотек .NET для досягнення своєї функціональності. За допомогою бібліотеки MySQL.Data програма встановлює підключення до бази даних MySQL та виконує SQL-запити для отримання необхідних даних. Бібліотека System.Xml забезпечує створення та збереження XML-документів, що дозволяє експортувати отримані дані у потрібному форматі. Використання бібліотеки System.Text.RegularExpressions сприяє ефективному парсингу текстових даних, тоді як бібліотека System.IO забезпечує зручне зчитування та збереження файлів. За допомогою бібліотеки Microsoft.Data.SqlClient програма встановлює підключення до бази даних MSSQL та виконує SQL-запити для отримання необхідних даних. Програма використовує бібліотеку MongoDB.Driver для взаємодії з базами даних MongoDB. Ця бібліотека надає засоби для підключення до MongoDB, виконання запитів та обробки результатів. Також, програма використовує бібліотеку Newtonsoft.Json для роботи з JSON-даними. Ця бібліотека надає інструменти для парсингу JSON, серіалізації та десеріалізації JSON-об'єктів.

Інтерфейс користувача, реалізований за допомогою Windows Forms, робить програму інтуїтивно зрозумілою для користувачів, дозволяючи легко налаштовувати параметри підключення до бази даних або вибір SQL-файлів. Застосування різних діалогових вікон, таких як SaveFileDialog та OpenFileDialog, полегшує процес вибору файлів та збереження результатів конвертації.

Програма надає користувачам зручний інструмент для перенесення та резервного копіювання даних з баз даних у формат XML. Усі компоненти та функціональні можливості реалізовані на високому рівні, що забезпечує ефективність роботи програми.

ВИСНОВКИ

З метою автоматизації перенесення даних з баз даних, в роботі було розглянуто та розроблено програму-конвертер файлів баз даних у файли формату XML. При створенні програми враховувалися сучасні технології, такі як бази даних, графічний редактор інтерфейсу Windows Forms та сучасні бібліотеки мови програмування C#. Дослідження актуальне, оскільки в сучасному світі збільшується кількість програмних додатків, багато з яких використовують уніфіковані формати файлів у своїх системах. Таким чином у результаті виконання кваліфікаційної роботи була досягнута її мета – автоматизація перенесення даних з баз даних за допомогою конвертування файлів баз даних у файли формату XML.

У даній дипломній роботі була розглянута проблема конвертації файлів баз даних у формат XML, що є актуальною задачею в сучасних інформаційних системах. Це питання важливе у контексті передачі даних між різними платформами та додатками, які використовують різні формати збереження інформації. Програма-конвертер, розроблена в рамках цього дослідження, демонструє можливості сучасних технологій та інструментів для вирішення складних задач з обробки та передачі даних.

У першому розділі дипломної роботи проведено детальне дослідження та аналіз предметної області, що включає вивчення сучасних методів і підходів до вирішення поставлених завдань. Основним завданням було розглянути існуючі технології та методики, що використовуються для перетворення даних баз даних у інші формати. Під час аналізу предметної області були розглянуті різні інструменти та бібліотеки, які надають можливість ефективно працювати з даними баз даних та здійснювати їх конвертацію у XML-формат.

На основі проведеного аналізу було зроблено висновок, що використання сучасних технологій і бібліотек дозволяє значно спростити процес перетворення даних та забезпечити високу продуктивність розробленого програмного забезпечення. Важливим аспектом є також

забезпечення користувацького досвіду за допомогою інтуїтивно зрозумілого інтерфейсу, що підвищує загальну ефективність та зручність використання програми. Було проведено аналіз існуючих програм і інструментів для конвертації даних, що допомогло визначити сильні та слабкі сторони різних підходів і врахувати їх у власній розробці.

Таким чином, розділ 1 не лише надав теоретичне обґрунтування вибору методів і технологій, але й заклав міцну основу для подальшої реалізації проекту, дозволяючи перейти до практичного впровадження розроблених рішень у рамках наступного розділу дипломної роботи.

У другому розділі дипломної роботи було проведено огляд концепцій баз даних та основних типів систем управління базами даних (СУБД). Розглянуто різні типи баз даних, включаючи реляційні, ієрархічні, мережеві, об'єктно-орієнтовані та NoSQL бази даних, що демонструє різноманітність підходів до зберігання та обробки даних. Реляційні бази даних, що використовують SQL, є найбільш поширеними завдяки підтримці складних запитів, цілісності та безпеці даних. Ієрархічні бази даних організовують дані у вигляді дерева, що обмежує масштабування. Мережеві бази даних дозволяють дочірнім записам мати кілька батьківських записів, але ускладнюють управління. Об'єктно-орієнтовані бази даних зберігають дані у вигляді об'єктів, але можуть бути неефективними для простих завдань. NoSQL бази даних, що використовують документи JSON, забезпечують гнучкість та високу швидкість обробки, що робить їх ідеальними для сучасних веб- та мобільних додатків. Вибір між реляційними та NoSQL базами даних залежить від специфічних потреб проекту. Реляційні бази даних підходять для складних запитів з високим рівнем структурованості, тоді як NoSQL – для великих обсягів неструктурованих даних.

У третьому розділі дипломної роботи було проведено ґрунтовне дослідження ключових аспектів розробки програмного забезпечення для конвертації даних з MySQL у XML. Розглядалися різні технічні підходи та рішення, які забезпечують ефективну реалізацію завдання.

Були досягнуті значні результати в області автоматизації перетворення даних із серверу MySQL та файлів .sql у формат XML. Використання бібліотек .NET, таких як System.Text.RegularExpressions та System.IO, дозволило ефективно працювати з текстовими та файловими даними, забезпечуючи високий рівень точності та швидкості обробки.

Було описано інтерфейс програми, який складається з двох основних вікон: головного вікна та додаткового вікна для введення назви атрибута. У головному вікні програми знаходяться елементи управління, такі як випадний список для вибору алгоритму конвертації та кнопка запуску процесу конвертації. Під час вибору алгоритму в головному вікні змінюється набір доступних параметрів для введення. Для кожного алгоритму передбачені відповідні елементи управління, такі як поля для введення даних або шлях до файлу, залежно від обраної опції. Використання стандартних елементів Windows Forms сприяє зручності користувача та спрощує процес взаємодії з програмою. Такий підхід до реалізації інтерфейсу дозволяє забезпечити зручну та інтуїтивно зрозумілу роботу програми для користувача.

Застосування Windows Forms у програмі зробило можливим створення зручного графічного інтерфейсу користувача, що забезпечує інтуїтивно зрозумілий процес налаштування та запуску конвертації даних. Це полегшило взаємодію користувача з програмою, дозволяючи швидко та ефективно виконувати необхідні операції.

Програма складається з двох основних компонентів: інтерфейсу користувача та алгоритму конвертації. Інтерфейс користувача реалізований за допомогою Windows Forms для вибору файлів баз даних та введення користувачем даних, потрібних для підключення до бази даних MySQL. Алгоритм конвертації забезпечує ефективне перетворення даних з формату SQL або напряму з баз даних у формат.

Отримані результати виявили практичну значущість програми, оскільки вона спрощує процес конвертації файлів баз даних у формат XML, що може бути важливим для подальшого аналізу та обробки даних. Отже, розроблена

програма-конвертер є важливим інструментом для покращення обробки даних та подальшого аналізу у різних галузях.

Розробка програмного забезпечення ніколи не зупиняється на першій версії продукту. Проаналізувавши результати дослідження, можна визначити кілька напрямків для подальшого розвитку проекту.

Перш за все, важливо розширити підтримку нових форматів баз даних та інших типів файлів. Це включає в себе підтримку інших типів баз даних та окремих СУБД, таких як PostgreSQL, Redis або Cassandra, що дозволить користувачам працювати з більш широким спектром даних. Крім того, інтеграція можливостей для роботи не тільки з форматом XML, а й з іншими форматами, як-то accdb та xlsx підвищить універсальність програми, зробивши її корисною для ще більшої кількості користувачів.

Однією з ключових задач є оптимізація продуктивності програми. Поліпшення швидкості конвертації даних дозволить ефективніше обробляти великі обсяги інформації. Це можна досягти шляхом аналізу існуючих алгоритмів та виявлення вузьких місць, що уповільнюють процес. Використання багатопоточності та паралельних обчислень для розподілу навантаження допоможе значно знизити час обробки даних.

Ще одним важливим напрямком є покращення інтерфейсу користувача. Впровадження сучасних підходів до дизайну, таких як Material Design або Flat Design зробить програму більш зручною та красивою для користувачів.

Кваліфікаційна робота виконана у відповідності до стандарту спеціальності 122 – «Комп’ютерні науки» і демонструє володіння такими компетентностями як:

– здатність обґрунтовано обирати та освоювати інструментарій з розробки та супроводження програмного забезпечення;

– здатність до алгоритмічного та логічного мислення;

– здатність проектувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування: узагальненого, об’єктно-

орієнтованого, функціонального, логічного, з відповідними моделями, методами та алгоритмами обчислень;

– здатність розробляти архітектури, модулі та компоненти програмних систем.

Серед результатів навчання, визначених стандартом кваліфікаційна робота реалізовує наступні:

– використовувати інструментальні засоби розробки програмних застосунків;

– володіти мовами системного програмування та методами розробки програм, що взаємодіють з компонентами комп’ютерних систем;

– розробляти програмні моделі предметних середовищ, обирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв’язання задач в галузі комп’ютерних наук.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Майстер конвертації баз даних. URL: https://fossdoc.com/sed-docs/db-transfer-wizard

2. DBF Viewer. URL: https://www.dbf2002.com/

3. Data Mapping Tools. URL: https://www.altova.com/mapforce#xml

4. Microsoft Visual Studio. URL: https://visualstudio.microsoft.com/

5. Introduction to C#: definition and utilities. URL: https://www.mytaskpanel.com/introduction-to-csharp/

6. MySQL Connector/NET Developer Guide. URL: https://dev.mysql.com/doc/connector-net/en/connector-net-ref-mysqlclient.html

7. Advantages and Disadvantages of Winforms? URL: https://www.software-developer-india.com/advantages-and-disadvantages-of-winforms/

8. Types of Databases. URL: https://www.geeksforgeeks.org/types-of-databases/

9. What Is an Object-Oriented Database? URL: https://www.mongodb.com/resources/basics/databases/what-is-an-object-oriented-database

10. What is a NoSQL database? URL: https://www.ibm.com/topics/nosql-databases

11. The Comma Separated Value (CSV) File Format. URL: https://creativyst.com/Doc/Articles/CSV/CSV01.shtml

12. Alan Beaulieu Learning SQL, Second Edition. Sebastopol : O’Reilly Media, 2009. 337 c.

13. Introducing JSON. URL: https://www.json.org/json-en.html

14. Dymola-JADE Co-Simulation for Agent-Based Control in Office Spaces - Scientific Figure on ResearchGate. URL: https://www.researchgate.net/figure/object-Figure-3-Object-structure-according-to-JSON-standard-ECMA-2013_fig4_318219915

15. Exploring the Evolution, Benefits, and Limitations of JSON: A Beginner's Guide. URL: https://medium.com/@adnankattekaden/exploring-the-evolution-benefits-and-limitations-of-json-a-beginners-guide-7d8ba5e1c5e4

16. Extensible Markup Language (XML) 1.0 (Fifth Edition). URL: https://www.w3.org/TR/xml/

17. XML Tree Structure. URL: https://www.techguruspeaks.com/structure-of-xml-documents/

18. XML Document Layout and Syntax. URL: https://itwebtutorials.mga.edu/xml/chp1/document-layout-syntax.aspx

19. Encoding Special Characters in XML. URL: https://www.baeldung.com/xml-encode-special-characters

20. Thomas Connolly, Carolyn Begg Database Systems: A Practical Approach to Design, Implementation, and Management, Sixth Edition. Harlow : Pearson, 2014. 1440 c.

21. System.Xml Namespace. URL: https://learn.microsoft.com/uk-ua/dotnet/api/system.xml?view=net-8.0

22. System.Text.RegularExpressions Namespace. URL: https://learn.microsoft.com/uk-ua/dotnet/api/system.text.regularexpressions?view=net-8.0

23. System.IO Namespace. URL: https://learn.microsoft.com/en-us/dotnet/api/system.io?view=net-8.0

24. System.Windows.Forms Namespace. URL: https://learn.microsoft.com/uk-ua/dotnet/api/system.windows.forms?view=windowsdesktop-8.0

ДОДАТКИ

Додаток А

Лістинг програми.

Form1.cs

```
using System.Data;
using System.Xml;
using System.Text.RegularExpressions;
using MongoDB.Bson;
using MongoDB.Driver;
using Microsoft.Data.SqlClient;
using Newtonsoft.Json.Linq;

namespace dyplomTry1
{

    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
```

```
    private void comboBox1_SelectedIndexChanged_1(object sender, EventArgs
e)
    {
        switch (comboBox1.SelectedIndex)
        {
            case 0: // Assuming the first item is selected
                lbl_server.Visible = false;
                lbl_user.Visible = true;
                lbl_pwd.Visible = true;
                lbl_db.Visible = true;
                lbl_table.Visible = true;
                host_lbl.Visible = false;
                port_lbl.Visible = false;
                lbl_serv_port.Visible = true;
                txtbox_server.Visible = true;
                txtbox_user.Visible = true;
                txtbox_db.Visible = true;
                txtbox_pwd.Visible = true;
                txtbox_pwd.PasswordChar = '*';
                txtbox_table.Visible = true;
                lbl_file.Visible = false;
                txtbox_file.Visible = false;
                file_btn.Visible = false;
                break;

            case 1: // Assuming the second item is selected
                lbl_server.Visible = false;
                lbl_user.Visible = false;
                lbl_pwd.Visible = false;
                lbl_db.Visible = false;
```

```
    lbl_table.Visible = false;
    host_lbl.Visible = false;
    port_lbl.Visible = false;
    lbl_serv_port.Visible = false;
    txtbox_server.Visible = false;
    txtbox_user.Visible = false;
    txtbox_db.Visible = false;
    txtbox_pwd.Visible = false;
    txtbox_table.Visible = false;
    lbl_file.Visible = true;
    txtbox_file.Visible = true;
    file_btn.Visible = true;
    break;

case 2: // Assuming the third item is selected
    lbl_server.Visible = false;
    lbl_user.Visible = false;
    lbl_pwd.Visible = false;
    lbl_db.Visible = true;
    lbl_table.Visible = true;
    host_lbl.Visible = true;
    port_lbl.Visible = true;
    lbl_serv_port.Visible = false;
    txtbox_server.Visible = false;
    txtbox_user.Visible = true;
    txtbox_db.Visible = true;
    txtbox_pwd.Visible = true;
    txtbox_pwd.PasswordChar = '\0';
    txtbox_table.Visible = true;
    lbl_file.Visible = false;
```

```
    txtbox_file.Visible = false;
    file_btn.Visible = false;
    break;

case 3: // Assuming the fourth item is selected
    lbl_server.Visible = true;
    lbl_user.Visible = true;
    lbl_pwd.Visible = true;
    lbl_db.Visible = true;
    lbl_table.Visible = true;
    lbl_serv_port.Visible = false;
    host_lbl.Visible = false;
    port_lbl.Visible = false;
    txtbox_server.Visible = true;
    txtbox_user.Visible = true;
    txtbox_db.Visible = true;
    txtbox_pwd.Visible = true;
    txtbox_pwd.PasswordChar = '*';
    txtbox_table.Visible = true;
    lbl_file.Visible = false;
    txtbox_file.Visible = false;
    file_btn.Visible = false;
    break;

case 4: // Assuming the 5 item is selected
    lbl_server.Visible = true;
    lbl_user.Visible = true;
    lbl_pwd.Visible = true;
    lbl_db.Visible = true;
    lbl_table.Visible = true;
```

```
        lbl_serv_port.Visible = true;
        host_lbl.Visible = false;
        port_lbl.Visible = false;
        txtbox_server.Visible = true;
        txtbox_user.Visible = true;
        txtbox_db.Visible = true;
        txtbox_pwd.Visible = true;
        txtbox_pwd.PasswordChar = '*';
        txtbox_table.Visible = true;
        lbl_file.Visible = false;
        txtbox_file.Visible = false;
        file_btn.Visible = false;
        break;

    case 5: // Assuming the 6 item is selected
        lbl_server.Visible = false;
        lbl_user.Visible = false;
        lbl_pwd.Visible = false;
        lbl_db.Visible = false;
        lbl_table.Visible = false;
        host_lbl.Visible = false;
        port_lbl.Visible = false;
        lbl_serv_port.Visible = false;
        txtbox_server.Visible = false;
        txtbox_user.Visible = false;
        txtbox_db.Visible = false;
        txtbox_pwd.Visible = false;
        txtbox_table.Visible = false;
        lbl_file.Visible = true;
        txtbox_file.Visible = true;
```

```
                file_btn.Visible = true;
                break;

            default:
                // Дії за замовчуванням або обробка непередбачених значень
                break;
        }
    }

    private void mysql_to_xml(XmlDocument xmlDocument)
    {
        // Get info about server
        string serverport = txtbox_server.Text;
        string[] parts = serverport.Split(":");
        string server = parts[0];
        string port = parts[1];
        string user = txtbox_user.Text;
        string db = txtbox_db.Text;
        string pwd = txtbox_pwd.Text;
        string table = txtbox_table.Text;
        try
        {
            // Connection to the server
            string connectionString = $"server={server};port={port};user={user};database={db};password={pwd}";
            MySql.Data.MySqlClient.MySqlConnection connection = new MySql.Data.MySqlClient.MySqlConnection(connectionString);
            connection.Open();
            // Select data from the specified table
            string query = $"SELECT * FROM {table}";
```

```
            using (MySql.Data.MySqlClient.MySqlCommand command = new
MySql.Data.MySqlClient.MySqlCommand(query, connection))
            {
                using (MySql.Data.MySqlClient.MySqlDataAdapter adapter = new
MySql.Data.MySqlClient.MySqlDataAdapter(command))
                {
                    DataTable dataTable = new DataTable();
                    adapter.Fill(dataTable);
                    XmlElement rootElement = xmlDocument.CreateElement("data");
                    xmlDocument.AppendChild(rootElement);

                    // Convert each row in DataTable to XML elements
                    foreach (DataRow row in dataTable.Rows)
                    {
                        XmlElement recordElement =
xmlDocument.CreateElement("record");

                        foreach (DataColumn column in dataTable.Columns)
                        {
                            XmlElement fieldElement =
xmlDocument.CreateElement(column.ColumnName);
                            fieldElement.InnerText = row[column].ToString();
                            recordElement.AppendChild(fieldElement);
                        }

                        rootElement.AppendChild(recordElement);
                    }
                }
            }
```

```
            // Saving .xml file
            using (SaveFileDialog saveFileDialog = new SaveFileDialog())
            {
                saveFileDialog.Filter = "XML files (*.xml)|*.xml|All files (*.*)|*.*";
                saveFileDialog.FilterIndex = 1;
                saveFileDialog.RestoreDirectory = true;

                DialogResult result = saveFileDialog.ShowDialog();
                if (result == DialogResult.OK)
                {
                    try
                    {
                        string filePath = saveFileDialog.FileName;
                        xmlDocument.Save(filePath);
                        MessageBox.Show("XML file saved successfully.", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show("Error saving XML file: " + ex.Message,
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error accessing to database: " + ex.Message, "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
```

```
    }

    private void mongodb_to_xml(XmlDocument xmlDocument)
    {
        // Get info about server
        string host = txtbox_user.Text;
        string port = txtbox_pwd.Text;
        string db = txtbox_db.Text;
        string collectionName = txtbox_table.Text;
        try
        {
            // Connection to the server
            string connectionString = "mongodb://" + host + ":" + port;
            MongoClient client = new MongoClient(connectionString);
            IMongoDatabase database = client.GetDatabase(db);
            var collection =
database.GetCollection<BsonDocument>(collectionName);
            // Select data from the specified table
            var documents = collection.Find(new BsonDocument()).ToList();
            XmlElement rootElement = xmlDocument.CreateElement("data");
            xmlDocument.AppendChild(rootElement);

            // Convert documents to XML elements
            foreach (var document in documents)
            {
                XmlElement recordElement = xmlDocument.CreateElement("record");

                foreach (var element in document.Elements)
                {
```

```
                XmlElement fieldElement =
xmlDocument.CreateElement(element.Name);
                fieldElement.InnerText = element.Value.ToString();
                recordElement.AppendChild(fieldElement);
            }

            rootElement.AppendChild(recordElement);
        }

        // Saving .xml file
        using (SaveFileDialog saveFileDialog = new SaveFileDialog())
        {
            saveFileDialog.Filter = "XML files (*.xml)|*.xml|All files (*.*)|*.*";
            saveFileDialog.FilterIndex = 1;
            saveFileDialog.RestoreDirectory = true;

            DialogResult result = saveFileDialog.ShowDialog();
            if (result == DialogResult.OK)
            {
                try
                {
                    string filePath = saveFileDialog.FileName;
                    xmlDocument.Save(filePath);
                    MessageBox.Show("XML file saved successfully.", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error saving XML file: " + ex.Message,
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
                }
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error accessing to database: " + ex.Message, "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void mssql_to_xml(XmlDocument xmlDocument)
{
    // Get info about server
    string server = txtbox_server.Text;
    string user = txtbox_user.Text;
    string db = txtbox_db.Text;
    string pwd = txtbox_pwd.Text;
    string table = txtbox_table.Text;
    try
    {
        // Connection to the MSSQL
        string connectionString = $"Server={server};Password={pwd};User
Id={user};initial catalog={db};Trusted_Connection =
True;TrustServerCertificate=True" /**/;
        using (SqlConnection connection = new
SqlConnection(connectionString))
        {
            connection.Open();
```

```
            // Selecting data from the specified table
            string query = $"SELECT * FROM {table}";
            using (SqlCommand command = new SqlCommand(query, connection))
            {
                using (SqlDataAdapter adapter = new SqlDataAdapter(command))
                {
                    DataTable dataTable = new DataTable();
                    adapter.Fill(dataTable);
                    XmlElement rootElement = xmlDocument.CreateElement("data");
                    xmlDocument.AppendChild(rootElement);

                    // Convert each row in a DataTable to XML elements
                    foreach (DataRow row in dataTable.Rows)
                    {
                        XmlElement recordElement = xmlDocument.CreateElement("record");

                        foreach (DataColumn column in dataTable.Columns)
                        {
                            XmlElement fieldElement = xmlDocument.CreateElement(column.ColumnName);
                            fieldElement.InnerText = row[column].ToString();
                            recordElement.AppendChild(fieldElement);
                        }

                        rootElement.AppendChild(recordElement);
                    }
                }
```

```
            }

            // Saving the .xml file
            using (SaveFileDialog saveFileDialog = new SaveFileDialog())
            {
                saveFileDialog.Filter = "XML files (*.xml)|*.xml|All files (*.*)|*.*";
                saveFileDialog.FilterIndex = 1;
                saveFileDialog.RestoreDirectory = true;

                DialogResult result = saveFileDialog.ShowDialog();
                if (result == DialogResult.OK)
                {
                    try
                    {
                        string filePath = saveFileDialog.FileName;
                        xmlDocument.Save(filePath);
                        MessageBox.Show("XML file saved successfully.", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show("Error saving XML file: " + ex.Message,
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
            }
        }
        catch (Exception ex)
        {
```

```
            MessageBox.Show("Error accessing database: " + ex.Message, "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void mariadb_to_xml(XmlDocument xmlDocument)
    {
        // Get info about server
        string serverport = txtbox_server.Text;
        string[] parts = serverport.Split(":");
        string server = parts[0];
        string port = parts[1];
        string user = txtbox_user.Text;
        string db = txtbox_db.Text;
        string pwd = txtbox_pwd.Text;
        string table = txtbox_table.Text;
        try
        {
            // Connection to the MariaDB
            string                    connectionString                    =
$"Server={server};Port={port};Database={db};User
ID={user};Password={pwd};";
            using (MySql.Data.MySqlClient.MySqlConnection connection = new
MySql.Data.MySqlClient.MySqlConnection(connectionString))
            {
                connection.Open();

                // Selecting data from the specified table
                string query = $"SELECT * FROM {table}";
```

```
            using (MySql.Data.MySqlClient.MySqlCommand command = new
MySql.Data.MySqlClient.MySqlCommand(query, connection))
            {
                using (MySql.Data.MySqlClient.MySqlDataAdapter adapter = new
MySql.Data.MySqlClient.MySqlDataAdapter(command))
                {
                    DataTable dataTable = new DataTable();
                    adapter.Fill(dataTable);
                    XmlElement rootElement =
xmlDocument.CreateElement("data");
                    xmlDocument.AppendChild(rootElement);

                    // Convert each row in a DataTable to XML elements
                    foreach (DataRow row in dataTable.Rows)
                    {
                        XmlElement recordElement =
xmlDocument.CreateElement("record");

                        foreach (DataColumn column in dataTable.Columns)
                        {
                            XmlElement fieldElement =
xmlDocument.CreateElement(column.ColumnName);
                            fieldElement.InnerText = row[column].ToString();
                            recordElement.AppendChild(fieldElement);
                        }

                        rootElement.AppendChild(recordElement);
                    }
                }
            }
```

```
            // Saving the .xml file
            using (SaveFileDialog saveFileDialog = new SaveFileDialog())
            {
                saveFileDialog.Filter = "XML files (*.xml)|*.xml|All files (*.*)|*.*";
                saveFileDialog.FilterIndex = 1;
                saveFileDialog.RestoreDirectory = true;

                DialogResult result = saveFileDialog.ShowDialog();
                if (result == DialogResult.OK)
                {
                    try
                    {
                        string filePath = saveFileDialog.FileName;
                        xmlDocument.Save(filePath);
                        MessageBox.Show("XML file saved successfully.", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show("Error saving XML file: " + ex.Message,
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
            }
        }
        catch (Exception ex)
        {
```

```
            MessageBox.Show("Error accessing to database: " + ex.Message, "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void convert_btn_Click(object sender, EventArgs e)
    {
        XmlDocument xmlDocument = new XmlDocument();
        switch (comboBox1.SelectedIndex)
        {
            case 0:
                mysql_to_xml(xmlDocument);
                txtbox_server.Text = "";
                txtbox_user.Text = "";
                txtbox_db.Text = "";
                txtbox_pwd.Text = "";
                txtbox_table.Text = "";
                break;

            case 1:
                file_sql_to_xml(xmlDocument);
                txtbox_file.Text = "";
                break;

            case 2:
                mongodb_to_xml(xmlDocument);
                txtbox_user.Text = "";
                txtbox_db.Text = "";
                txtbox_pwd.Text = "";
                txtbox_table.Text = "";
```

```
            break;

        case 3:
            mssql_to_xml(xmlDocument);
            txtbox_server.Text = "";
            txtbox_user.Text = "";
            txtbox_db.Text = "";
            txtbox_pwd.Text = "";
            txtbox_table.Text = "";
            break;

        case 4:
            mariadb_to_xml(xmlDocument);
            txtbox_server.Text = "";
            txtbox_user.Text = "";
            txtbox_db.Text = "";
            txtbox_pwd.Text = "";
            txtbox_table.Text = "";
            break;

        case 5:
            file_json_to_xml(xmlDocument);
            txtbox_file.Text = "";
            break;

        default:
            break;
    }

}
```

```
private void file_sql_to_xml(XmlDocument xmlDocument)
{
   try
   {
      string FilePath = txtbox_file.Text;

      // Initialize variables to store the longest line and its length
      string longestLine = "";
      int maxLength = 0;

      // Read input from the file line by line and find the longest line
      using (StreamReader sr = new StreamReader(FilePath))
      {
         string line;
         while ((line = sr.ReadLine()) != null)
         {
            if (line.Length > maxLength)
            {
               longestLine = line;
               maxLength = line.Length;
            }
         }
      }

      // Define the regular expression pattern to match the values within parentheses
      string pattern = @"\(([^)]*)\)";

      // Create a regular expression object
```

```
            Regex regex = new Regex(pattern);

            // Match the pattern in the longest line
            MatchCollection matches = regex.Matches(longestLine);

            // Prompt the user for the name of each argument using a new form
            List<string> argumentNames = new List<string>();
            foreach (string value in matches[0].Groups[1].Value.Split(','))
            {
               // Remove leading and trailing whitespaces and single quotes
               string trimmedValue = value.Trim().Trim('\'');

               // Prompt the user for the name of the argument using a new form
               using (AttributeNameForm attributeNameForm = new
AttributeNameForm(trimmedValue))
               {
                  if (attributeNameForm.ShowDialog() == DialogResult.OK)
                  {
                     string argumentName = attributeNameForm.AttributeName;
                     // Store the argument name
                     argumentNames.Add(argumentName);
                  }
                  else
                  {
                     break;
                  }
               }
            }

            // Create the root element
```

```
            XmlElement root = xmlDocument.CreateElement("records");
            xmlDocument.AppendChild(root);

            // Iterate through each match and extract the values
            foreach (Match match in matches)
            {
                // Create a record element
                XmlElement record = xmlDocument.CreateElement("record");
                root.AppendChild(record);

                // Get the values within parentheses
                string values = match.Groups[1].Value;

                // Split the values by comma
                string[] valueArray = values.Split(',');

                // Iterate through each value and add it to the record element with its corresponding argument name
                for (int i = 0; i < valueArray.Length; i++)
                {
                    // Remove leading and trailing whitespaces and single quotes
                    string trimmedValue = valueArray[i].Trim().Trim('\'');

                    // Create an element with the argument name as the tag
                    XmlElement element = xmlDocument.CreateElement(argumentNames[i]);
                    element.InnerText = trimmedValue;
                    // Add the element to the record
                    record.AppendChild(element);
                }
```

```
            }
            using (SaveFileDialog saveFileDialog = new SaveFileDialog())
            {
                saveFileDialog.Filter = "XML files (*.xml)|*.xml|All files (*.*)|*.*";
                saveFileDialog.FilterIndex = 1;
                saveFileDialog.RestoreDirectory = true;
                DialogResult result = saveFileDialog.ShowDialog();
                if (result == DialogResult.OK)
                {
                    try
                    {
                        string filePath = saveFileDialog.FileName;
                        xmlDocument.Save(filePath);
                        MessageBox.Show("XML file saved successfully.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show("Error saving XML file: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error accessing to file: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
```

```
private void file_json_to_xml(XmlDocument xmlDocument)
{
   try
   {
      // JSON input string
      string jsonPath = txtbox_file.Text;
      string jsonString = File.ReadAllText(jsonPath);
      XmlElement root = xmlDocument.CreateElement("records");
      xmlDocument.AppendChild(root);

      JArray jsonArray = JArray.Parse(jsonString);
      foreach (JObject jsonObject in jsonArray)
      {
         XmlElement recordElement = xmlDocument.CreateElement("record");
         foreach (var property in jsonObject.Properties())
         {
            XmlElement propertyElement =
xmlDocument.CreateElement(property.Name);
            propertyElement.InnerText = property.Value.ToString();
            recordElement.AppendChild(propertyElement);
         }
         root.AppendChild(recordElement);

      }

      using (SaveFileDialog saveFileDialog = new SaveFileDialog())
      {
         saveFileDialog.Filter = "XML files (*.xml)|*.xml|All files (*.*)|*.*";
         saveFileDialog.FilterIndex = 1;
```

```
            saveFileDialog.RestoreDirectory = true;
            DialogResult result = saveFileDialog.ShowDialog();
            if (result == DialogResult.OK)
            {
               try
               {
                  string filePath = saveFileDialog.FileName;
                  xmlDocument.Save(filePath);
                  MessageBox.Show("XML file saved successfully.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
               }
               catch (Exception ex)
               {
                  MessageBox.Show("Error saving XML file: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
               }
            }
         }
         catch (Exception ex)
         {
            MessageBox.Show("Error saving XML file: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
         }
      }

private void file_btn_Click(object sender, EventArgs e)
      {
         switch (comboBox1.SelectedIndex)
         {
```

```
case 1:
    // Create an instance of OpenFileDialog
    OpenFileDialog openFileDialog1 = new OpenFileDialog();

    // Set properties of the OpenFileDialog
    openFileDialog1.Title = "Choose a file";
    openFileDialog1.Filter = "SQL Files (*.sql)|*.sql";
    openFileDialog1.Multiselect = false;

    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        string selectedFilePath = openFileDialog1.FileName;

        // Set the selected file path in the TextBox
        txtbox_file.Text = selectedFilePath;
    }
    break;

case 5:
    // Create an instance of OpenFileDialog
    OpenFileDialog openFileDialog2 = new OpenFileDialog();

    // Set properties of the OpenFileDialog
    openFileDialog2.Title = "Choose a file";
    openFileDialog2.Filter = "Json files (*.json)|*.json";
    openFileDialog2.Multiselect = false;

    if (openFileDialog2.ShowDialog() == DialogResult.OK)
    {
        string selectedFilePath = openFileDialog2.FileName;
```

```
                // Set the selected file path in the TextBox
                txtbox_file.Text = selectedFilePath;
            }
            break;
        }
    }
}

// Define a form to prompt the user for attribute names
public class AttributeNameForm : Form
{
    private Label label1;
    private TextBox textBox1;
    private Button button1;

    public string AttributeName { get { return textBox1.Text; } }

    public AttributeNameForm(string attributeName)
    {
        InitializeComponent();
        textBox1.Text = attributeName;
    }

    private void InitializeComponent()
    {
        label1 = new Label();
        textBox1 = new TextBox();
        button1 = new Button();
        SuspendLayout();
```

```
//
// label1
//
label1.AutoSize = true;
label1.Location = new System.Drawing.Point(12, 9);
label1.Name = "label1";
label1.Size = new System.Drawing.Size(94, 13);
label1.TabIndex = 0;
label1.Text = "Enter attribute name:";
//
// textBox1
//
textBox1.Location = new System.Drawing.Point(12, 25);
textBox1.Name = "textBox1";
textBox1.Size = new System.Drawing.Size(180, 20);
textBox1.TabIndex = 1;
//
// button1
//
button1.DialogResult = DialogResult.OK;
button1.Location = new System.Drawing.Point(61, 51);
button1.Name = "button1";
button1.Size = new System.Drawing.Size(75, 23);
button1.TabIndex = 2;
button1.Text = "OK";
button1.UseVisualStyleBackColor = true;
//
// AttributeNameForm
//
AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
```

```
            AutoScaleMode = AutoScaleMode.Font;
            ClientSize = new System.Drawing.Size(204, 86);
            Controls.Add(button1);
            Controls.Add(textBox1);
            Controls.Add(label1);
            Name = "AttributeNameForm";
            Text = "Enter Attribute Name";
            ResumeLayout(false);
            PerformLayout();

        }
    }
}
```