МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.
"_____"_____ 2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: Програмна система конвертації даних при інформаційному моделюванні споруд_____
_____

Виконавець: _____Сім'я Я.В.

Керівник: _____ Халімон Н.Ф.

Нормоконтролер: _____Тупота Є.В.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут комп'ютерних інформаційних технологій

Кафедра комп'ютеризованих систем управління

Напрям 123 «Комп'ютерна інженерія»

ЗАТВЕРЖУЮ
Завідувач кафедри

Литвиненко О. Є.

« ____ »_____2020 р.

З А В Д А Н Н Я
на виконання дипломної роботи

Сім'ї Ярослава Володимировича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломного проекту: Програмна система конвертації даних при інформаційному моделюванні споруд
затверджена наказом ректора від *" 27 " серпня 2020 р. № 1203/ст.*

2. Термін виконання роботи: з 5.10.2020 до 31.12.2020

3. Вихідні дані до роботи: постановка задачі, програмні платформи, мови програмування, бібліотеки, СУБД.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

1) Аналіз програмних засобів конвертації даних;

2) Технології конвертації даних при інформаційному моделюванні споруд;

3) Проектування програмної системи конвертації даних при інформаційному моделюванні споруд;

4) Розробка програмної системи конвертації даних при інформаційному моделюванні споруд.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Структурна схема програмної системи конвертації даних;

2) Схема алгоритму конвертації даних;

3) Вікно форми програмної компоненти створення правил конвертації;

4) Графіки залежності часу конвертації від кількості строк таблиці;

5) Діаграма зміни часу конвертації при зростанні кількості строк даних.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Примітка
1	Проаналізувати існуючі аналоги програмної системи конвертації даних при інформаційному моделюванні споруд	05.10.2020 – 20.10.2020	
2	Проаналізувати технології конвертації даних	21.10.2020 – 28.10.2020	
3	Розробити проект програмної системи конвертації даних при інформаційному моделюванні споруд	29.10.2020 – 05.11.2020	
4	Розробити алгоритмічну структуру програмної системи конвертації даних при інформаційному моделюванні споруд	06.11.2020 – 12.11.2020	
5	Розробити програмну систему конвертації даних при інформаційному моделюванні споруд	13.11.2020 – 20.11.2020	
6	Оформити пояснювальну записку	21.11.2020 – 04.12.2020	
7	Оформити графічний та ілюстративний матеріал	05.12.2020 – 13.12.2020	

7. Дата видачі завдання: «22» жовтня 2020 р.

Керівник дипломної роботи _____ Халімон Н.Ф.
(підпис керівника)

Завдання прийняв до виконання _____ Сім’я Я.В.
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмна система конвертації даних при інформаційному моделюванні споруд»: 104 сторінки, 16 рисунків, 23 використане джерело.

КОНВЕРТАЦІЯ ДАНИХ, ПРОГРАМНА СИСТЕМА, МОДЕЛЮВАННЯ СПОРУД, ІНФОРМАЙНЕ МОДЕЛЮВАННЯ, *SQL*, *XML*, *SQL*-ЗАПИТИ, *C#*, *TEMPLATES*.

Об’єктом дослідження даної дипломної роботи є системи конвертації даних при інформаційному моделюванні споруд.

Предметом дослідження є проектування та створення програмної системи конвертації даних при інформаційному моделюванні споруд.

Метою даної дипломної роботи є розробка програмної системи автоматичної конвертації даних, яка може бути використана як складова частина в програмах інформаційного моделювання споруд.

Методи дослідження – технології створення програмної системи конвертації даних при інформаційному моделюванні споруд, засоби конвертації даних.

Матеріали дипломної роботи рекомендується використовувати при проведенні наукових досліджень, у навчальному процесі для фахівців з системного програмування, а також у всіх сферах, де є необхідність створення систем конвертації даних.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

BIM (*building information modeling*) – це технологія оптимізації процесів проектування та будівництва, основа якої полягає в використанні єдиної моделі споруди.

БД – база даних.

СУБД (система управління базами даних) – програмна система, що реалізує набір візуальних та функціональних інструментів для адміністрування баз даних підтримуваних форматів.

SSMS (*SQL Server Management Studio*) – утиліта, призначена для керування та адміністрування сервера баз даних *SQL Server*.

SQL (*structured query language*) – декларативна мова програмування, яка застосовується для створення, модифікації та керування.

XML (*extensible markup language*) – розширювана мова розмітки. Запропонований консорціумом *World Wide Web Consortium* стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосуваннями, зокрема через інтернет.

IDA (*Interactive DisAssembler*) – інтерактивних дизасемблер, який широко використовується для реверсної розробки. Підхід відрізняється гнучкістю, наявністю вбудованої мови програмування, підтримкою багатьох форматів виконуваних файлів для великої кількості процесорів та операційних систем.

ASCII (*American standard code for information interchange*) – назва таблиці, в якій деяким розповсюдженим друкарським та недрукарським символам виділені числові коди.

PC (*portative computer*) – персональный або портативний комп’ютер.

CPU (*central processing unit*) – електронний блок або інтегральна схема, що виконує машинні інструкції. Основна частина апаратного забезпечення.

ВСТУП

Більшість користувачів розглядає інформаційне моделювання споруд як новий підхід в проектуванні. Технологія принципово полегшує всі рутинні операції, пов’язані як із започаткуванням самої ідеї проектування нової споруди, так і з автоматизацією створення великої кількості супроводжуючої будь-який проект технічної документації. Інформаційне моделювання споруд поліпшує виконання інтелектуальної роботи проектувальника. Також, ця технологія дозволяє проводити всі етапи проектування споруд без використання роздрукованих схем.

Інформаційне моделювання споруд (*building information modeling, BIM*) – технологія автоматизації процесів проектування та будівництва, основа якої полягає в використанні єдиної моделі споруди та обміні інформацією про будь-який об’єкт між усіма учасниками протягом усього життєвого циклу будівництва. У залежності від етапів проектування споруди, дані можуть зберігатися в декількох екземплярах інформаційної моделі різного формату. У результаті виникає необхідність структуризації даних та створення якісного та ресурсо-економного програмного забезпечення, тому тема дипломної роботи є актуальною.

Об’єктом дослідження даної дипломної роботи є системи конвертації даних при інформаційному моделюванні споруд.

Предметом дослідження є проектування та створення програмної системи конвертації даних при інформаційному моделюванні споруд.

Метою даної дипломної роботи є розробка програмної системи автоматичної конвертації даних, яка може бути використана як складова частина в програмах інформаційного моделювання споруд.

Методами дослідження є технології створення програмних систем конвертації даних при інформаційному моделюванні споруд, засоби конвертації даних.

В ході виконання дипломної роботи були отримані порівняльні характеристики середнього часу виконання процесів конвертації розробленим програмним забезпеченням. На основі порівняння отриманих характеристик із характеристиками уже існуючих програмних рішень даного типу були зроблені висновки щодо доцільності впровадження програмної системи у галузь конвертації даних при інформаційному моделюванні споруд. Також, отримані результати можуть бути використані для проектування більш ефективного програмного забезпечення аналогічного типу.

Використання *BIM* технології покращує роботу із спорудами чи конструкціями, які проектуються. Вона дозволяє у віртуальному режимі зібрати разом, підібрати по значенню, розрахувати та зістикувати компоненти та системи майбутньої споруди, що створюються спеціалістами різного профіля. Провірити їх особливості та життєздатність, функціональну придатність та експлуатаційні якості як окремих частин, так і всієї споруди разом було вкрай важкою задачею. Також технологія *BIM* дозволяє уникнути самої незручної з точки зору проектування проблеми – появу внутрішніх відмінностей (колізій), виникаючих при сполученні в одному проекті його складових частин та суміжних розділів.

При інформаційному моделюванні навіть невеликих за масштабом споруд виникають великі об’єми даних. В залежності від етапів проектування споруди дані можуть зберігатися в декількох екземплярах інформаційної моделі. В результаті виникає необхідність структуризації даних таким чином, щоб максимально зменшити використання системних ресурсів, та спростити створення програмних рішень конвертації даних.

Конвертація даних досить складний процес, що проявляє всі свої можливості лише при застосуванні різностороннього підходу. Процес конвертації має бути зробленим таким чином, щоб забезпечувати максимальну зручність, швидкість, та мінімальну ресурсозатратність. Від програміста вимагаються доскональні знання при роботі з базами даних, в тому числі і з реляційними, роботі з файлами та роботі з форматами розмітки файлів .

Конвертація складається з наступних етапів: створення файлу моделі вихідного формату, читання всіх необхідних даних з файлу моделі, створення параметрів конвертації, експорт усіх баз знань, експорт налаштувань моделі, експорт усіх об’єктів моделі та експорт даних про існуючі конструкції чи зборки.

Конвертація даних – перетворення даних з одного формату в інший. Зазвичай із збереженням основного логічно-структурного змісту інформації. Потреба в конвертації даних виникає через необхідність програмних засобам працювати з декількома форматами даних. Перетворення даних пов’язане з різницею логічних структур даних і, як результат, має великий список проблем.

У залежності від вимог та ступеня підготовки користувача існують програми як з попередньо встановленими режимами конвертації, так і з можливостями налаштування процесів конвертації. Усі програмні рішення в галузі конвертації поділяються на вбудовані модулі конвертації, які виконують конвертацію форматів, що підтримуються головною програмою, та відокремлені програми або утиліти, що спеціалізуються на конвертації декількох форматів даних.

Програмна система – різновид програмного забезпечення, що використовується для розв’язання всього спектру задач, що можуть виникнути при вирішенні поставленої проблеми. Розрізняють прикладні програмні системи, статистичні програмні системи та інструментальні програмні системи (комп’ютерні програми, призначені для проектування, розробки та адміністрування).

До основних функцій програмної системи конвертації даних при інформаційному моделюванні споруд можна віднести наступні основні операції: підключення до баз даних формату **.mdf* засобами системи керування базами даних *Microsoft SQL Server*, читання та відображення вибраних даних у вигляді таблиць даних, створення правил імпорту та експорту даних, читання та збереження створених правил з бази знань правил конвертації формату **.xml*, виконання конвертації даних та їх відображення в інтерфейсі програмної системи.

Під підключенням до баз даних формату *.*mdf* (*main data file*) мається на увазі підключення через стандартну аутентифікацію *Windows* до існуючої на машині версії *SQL Server* та відкриття вибраної бази даних. Після чого, програмна система може створювати запити до відкритої бази даних на мові *Transact-SQL*.

Для читання та відображення вибраної інформації у вигляді таблиці даних необхідно відображати список з назвами всіх наявних в базі даних об’єктів. Після вибору одного з об’єктів, його дані мають відобразитись у вигляді таблиці, що дозволить користувачеві програми ефективно орієнтуватися в виборі необхідних даних для їх подальшої конвертації.

Під процесом створення правил імпорту та експорту даних мається на увазі визначення користувачем програмної системи колонок, між якими буде виконано транслювання даних, та формули, за якими буде виконуватися транслювання.

Створення правил імпорту – це процес визначення правил транслювання інформації з вихідного набору даних у вхідний. У вигляді правила імпорту записується формула, за якою дані при виконанні процесу конвертації будуть зчитані з однієї або декількох таблиць вихідного формату та записані в таблицю вхідного формату.

Під збереженням створених правил в базі знань правил конвертації мається на увазі надання користувачеві можливості створювати нові, відкривати, редагувати та зберігати правила в уже існуючий, або новий файл формату *.*xml*. Для збереження правила достатньо зазначити назву колонки, в яку дані будуть записані, та правило (формулу, по якій значення будуть обчислені).

Програмна система має виконувати конвертацію даних по створеним та збереженим в файл бази знань конвертації правилам. Створений пустий об’єкт таблиці з структурною схемою вихідного формату даних буде заповнюватися записами з вхідної таблиці та відображатися користувачеві для порівняння результатів.

Для порівняння показників роботи старого та розробленого модулів конвертації даних проведено серію з десяти тестів. Кожен тест полягав у виконанні конвертації формату даних з різною кількістю строк даних. Метою тестів є визначення середнього часу виконання процесу конвертації.

Для визначення часу виконання конвертації даних було використано інструменти *DiagnosticTools* середовища розробки *Visual Studio* 2019. Обов’язковою умовою використання інструментів діагностики є наявність початкового коду розробленої програмної системи конвертації даних та початкового коду модуля синхронізації баз даних програми *LIRA* 10.12.

РОЗДІЛ 1
ТЕХНОЛОГІЇ ІНФОРМАЦІЙНОГО МОДЕЛЮВАННЯ СПОРУД

1.1. *BIM* технології

Більшість користувачів розглядає інформаційне моделювання споруд як новий підхід в проектуванні. Технологія принципово полегшує всі рутинні операції, пов'язані як із започаткуванням самої ідеї проектування нової споруди, так і з автоматизацією створення великої кількості супроводжуючої будь-який проект технічної документації. Інформаційне моделювання споруд поліпшує виконання інтелектуальної роботи проектувальника. Також, ця технологія дозволяє проводити всі етапи проектування споруд без використання роздрукованих схем [1].

Інформаційне моделювання споруд (*building information modeling, BIM*) – це технологія автоматизації процесів проектування та будівництва, основа якої полягає в використанні єдиної моделі споруди та обміні інформацією про будь-який об'єкт між усіма учасниками протягом усього життєвого циклу – від задумки власника та перших начерків архітектора до технічного обслуговування готової споруди. Одна з переваг *BIM* технології перед системою автоматичного проектування *CAD* (*computer-aided design*) полягає в підтримці роздільного користування, що дозволяє використовувати дану технологію в цілях реалізації *IDA*. Інструментарій *BIM* повинен виключити надлишковість, повторне введення та втрату даних, помилки при їх передачі та конвертації.

Спільна робота керуючої команди здійснюється протягом усього життєвого циклу проекту будівництва від концепції до експлуатації. До завдань керуючої команди на етапах життєвого циклу проекту входять:

– розробка концепції дизайну – формування загального бачення проекту та його цілей. Оцінка економічного оточення, клімату, соціального оточення та стану території будівництва;

– схематичний дизайн – уточнення бачення проекту поряд з пошуком додаткових ідей, технологій і методів, які дозволять ефективніше досягти цілей проекту. Колективна оцінка проекту. Розробка завдання на проектування;

– розробка проектної документації – координація подальшої оптимізації проекту для відповідності поставленим цілям. Остаточне затвердження проекту власником об'єкта;

– розробка робочої документації – встановлення регламенту проведення будівництва. Контроль і координація підготовки документації та вибору підрядників;

– будівництво будівлі – контроль і координація ходу будівництва в певних критичних точках. Кінцевий контроль, тестування і підтвердження якості виконаних робіт;

– експлуатація будівлі – контроль і координація передачі об'єкта користувачам і експлуатаційному персоналу. Проведення оцінки ефективності функціонування будівлі і відповідності поставленим цілям.

Використання *BIM* технології покращує роботу із спорудами чи конструкціями, які проектуються. Вона дозволяє у віртуальному режимі зібрати разом, підібрати по значенню, розрахувати та зістикувати компоненти та системи майбутньої споруди, що створюються спеціалістами різного профіля. Провірити їх особливості та життєздатність, функціональну придатність та експлуатаційні якості як окремих частин, так і всієї споруди разом було вкрай важкою задачею. Також технологія *BIM* дозволяє уникнути самої незручної з точки зору проектування проблеми – появу внутрішніх відмінностей (колізій), виникаючих при сполученні в одному проекті його складових частин та суміжних розділів.

Технологія *BIM* додатково надає нові можливості, які раніше не розглядалися. До таких можливостей варто віднести:

– новий, цифровий рівень керування експлуатацією споруд, підтримку та корегування його функцій впродовж усіх етапів проектування;

– дослідження та експериментування в області розробки та проектування споруд;

– рівень точності будівництва.

Інформаційне моделювання споруд надає можливість проводити наукові дослідження та експерименти практично по всім питанням, пов'язаним із плануванням, моделюванням, внутрішнім конструюванням, оснащенням, енергоспоживанням, особливостями проектування, побудовою та іншими аспектами проектно-будівничої галузі. Для будь-яких з перерахованих цілей створюється не конкретна модель реальної споруди, а абстрактна комп'ютерна конструкція, що імітує досліджувану ситуацію. У подальшому на таку модель створюють також віртуальні навантаження та аналізують отримані результати. Така модель називається дослідницькою інформаційною моделлю.

Відмінність дослідницької моделі від звичайної *BIM* технології полягає в тому, що перша з самого початку призначена для дослідження будь-яких загальних аспектів проектування, оснащення або функціонування споруд, та може не відповідати жодній реально існуючій споруді [1].

На відміну від традиційних систем комп'ютерного проектування, які створюють геометричні образи, результатом інформаційного моделювання споруди досить часто є об'єктно-орієнтована цифрова модель як усієї споруди, так і процесів організації його будівництва [2]. На основі інформаційної моделі простіше організовувати процеси будівництва споруд. Відмінностями *BIM* технології від традиційних комп'ютерних моделей споруд є точна геометрія та властивості об'єктів. При точній геометрія усі об'єкти задаються достовірно (в повній відповідності з реальною, в тому числі й внутрішньою конструкцією), геометрично правильно та в точних розмірах. Також, усі об'єкти в моделі мають деякі раніше задані властивості (характеристики матеріалів, коди виробників,

ціни, дати обслуговування і т. д.), які можливо виміряти та використати як в самій моделі, так і через спеціальні формати файлів.

Використання параметрів споруди при проектуванні інформаційної моделі зв'язків приводить до того, що вся модель правильно обробляється при вимірюванні окремих елементів, оскільки всі вони сполучені необхідними залежностями. Міняючи будь-які існуючі параметри, користувач надає об’єкту нові геометричні форми та якісно-кількісні характеристики.

Зручний інтерфейс сучасних *BIM* програм дозволяє проектувальнику легко коректувати параметри для вже вставлених в модель об’єктів та принципово полегшити таким чином внесення великої кількості змін в проект. Параметричне моделювання також дозволяє швидко та точно міняти взаєморозташування об’єктів, оскільки одним із параметрів управління геометрією може бути відстань між ними. Якщо об’єкти об’єднані в які-небудь групи, то переміщення буде виконано для всіх об’єктів у вибраній групі [3]. В попередньому непараметричному проектуванні переміщення таких конструкцій (воно відносилось до найбільш ймовірних та важливих задач) потребувало велику кількість часу.

Тепер, завдяки *BIM* технології, внутрішнє перепланування у вже готовий проект споруди займає мало часу. Не менш ефективно параметричне моделювання проявило себе при зміні зовнішньої форми складних споруд, комплексне проектування яких на той момент вже встигло сильно розвинутися від начальних стадій.

Використання *BIM* дозволяє вести роботу з моделлю споруди безпосередньо з будь-якого вигляду чи ракурсу цієї моделі. Такими ракурсами можуть бути: етажні плани, фасади, просторові види чи креслярські схеми [4]. Видами моделі можуть бути різноманітні таблиці та специфікації з їх полями.

1.2. Засоби зберігання інформації при моделюванні споруд

При інформаційному моделюванні навіть невеликих за масштабом споруд виникають великі об’єми даних. У залежності від етапів проектування споруди дані можуть зберігатися в декількох екземплярах інформаційної моделі. У результаті виникає необхідність структуризації даних таким чином, щоб максимально зменшити використання системних ресурсів та спростити процеси конвертації даних в інші формати [5]. Дані, що виникають при інформаційному моделюванні споруд поділяються на: параметри проекту (мови, формати чи вигляди), інформаційні моделі, об’єкти моделі, аналоги реальних об’єктів споруди (стіни, підлога, колони чи балки), об’єкти моделі, які не мають фізичних аналогів (поверхи, кімнати), конструкції, параметри, бази знань (матеріалів, профілів, правил, параметрів) та журнали подій.

Конструкції – збірки, сполуки декількох реальних об’єктів споруди. Дозволяють спростити створення, редагування та видалення об’єктів, оскільки при роботі вони поводять себе як одне ціле.

Здебільшого, під файлом з даними моделі розуміють файл певного формату, в якому записані дані параметрів проекту, інформаційної моделі, всіх видів об’єктів, їх конструкцій та параметрів. У об’єкт моделі записуються всі параметри, незалежно від їх цільового призначення, будь то геометрія об’єкта (загальні ширина, висота, товщина і т. д.), чи аналітичні параметри (параметри навантажень). Далі, в залежності від етапу проектування споруди, з файлу моделі зчитуються лише необхідні параметри. Це зберігає всі необхідні дані лише в одному екземплярі моделі.

Винесення баз знань за межі файлу моделі дозволяє зменшити об’єми необхідної пам’яті. В базу записуються всі необхідні види об’єктів (наприклад матеріалів). Далі, в інтерфейсі програми при додаванні до об’єкту споруди матеріалу в файл моделі записується лише індекс запису з основною

інформацією про об’єкт в базі знань матеріалів. Таким чином інформації матеріалу не дублюється при його додаванні на декілька об’єктів моделі.

Дані *BIM* моделі здебільшого зберігаються в одному або декількох файлах конкретного формату, деякі дані можуть зберігатися у вигляді баз даних. До таких даних відносяться інформація про автоматизовану побудову об’єктів, матеріали, форми профілів, кольори та інше [5]. В такому випадку в базу даних записується вся основна інформація про об’єкт, в той час як в файлі моделі зазначається індекс з відповідним записом з бази даних [6]. До баз даних, що частіше за інші використовуються, відносяться: *MySQL*, *Microsoft SQL Server*, *PostgreSQL*, *MongoDB* та *Microsoft Access*.

MySQL працює на операційних системах *Linux, Windows, OSX, FreeBSD* і *Solaris*. Можна почати працювати з безкоштовним сервером, а потім перейти на комерційну версію. Ліцензія *GPL* з відкритим вихідним кодом дозволяє модифікувати програмне забезпечення *MySQL*.

Ця система управління базами даних використовує стандартну форму *SQL*. Програми для проектування таблиць мають інтуїтивно зрозумілий інтерфейс. *MySQL* підтримує до п’ятидесяти мільйонів рядків в таблиці. Граничний розмір файлу для таблиці складає за замовчуванням чотири гігабайти, але його можна збільшити. Підтримує секціонування і реплікацію, *Xpath*, процедури, тригери та подання [7]. До особливостей системи відносяться: масштабування, легкість використання, безпечність, швидкість та підтримка багатьох операційних систем.

Microsoft SQL Server – найпопулярніша комерційна СУБД (система керування базами даних). Розроблена корпорацією *Microsoft*. Основна мова запитів – *Transact-SQL*. Графічний інтерфейс та програмне забезпечення основані на командах. Особливостями системи є: висока продуктивність, залежність від системи, можливість встановити різні версії на одному комп’ютері та генерація скриптів для переміщення даних [6].

Microsoft SQL Server позиціонується як реляційна СУБД з підтримкою мови *SQL* та можливістю роботи по локальним мережам. Система підтримує

спільні роботи *SQL* та *Server dBase* або будь-яким іншим програмних забезпеченням для робочої станції. Великий акцент робиться на клієнт-серверну архітектуру продукту, завдяки чому реалізуються роздільні функції клієнтського додатку, в якому користувачі можуть бачити необхідні їм дані, та серверна частина, в якій ці дані зберігаються.

Система підтримує зберігання процедур, компілюємий *SQL Server* та значно пришвидшену вибірку даних, а також підтримку цілісності даних при роботі в середовищі, призначеному для декількох користувачів. Підтримує постійну доступність ядра для адміністративних задач та технологію, що виконує роль мосту між системами обробки онлайн-транзакцій з базами даних на машині. З виходом останньої версії *SQL Server* з'явилися наступні функції:

– компонент *SQL Server AlwaysOn* для створення резервних копій БЗ;

– можливість встановити *SQL Server* в середовищі *Windows Server Core*;

– організацію зберігання даних для пришвидшення виконання запитів;

– вдосконалення мови *T-SQL* (введення об'єктів *Sequence* та віконних функцій);

– можливість відслідковування даних *CDC* для СУБД *Oracle*;

– можливість користувачеві визначати ролі сервера;

– служби керування якістю даних *Data Quality Services* (бази даних, що визначають якість даних);

– *Crescent* – новий інструмент візуалізації даних;

– підтримка автономних баз даних (для переміщення між локальними екземплярами *SQL Server* та *SQL Azure*);

– *Juneau* – нове середовище розробки *SQL Server Developer Tools*.

PostgreSQL – об'єктно-реляційна база даних, яка працює на *Linux, Windows, OSX* і деяких інших системах. У *PostgreSQL* 10 є такі функції, як логічна реплікація, декларативне розбиття таблиць, поліпшені паралельні запити, більш безпечна аутентифікація по паролю на основі *SCRAM-SHA*-256. До особливостей системи можна віднести: підтримку табличних просторів,

процедур, об’єднань та тригерів, відновлення на момент часу (*PITR*) та асинхронну реплікацію [8].

Microsoft Access – система управління базами даних від *Microsoft*, яка поєднує в собі реляційне ядро бази даних *Microsoft Jet* з графічним інтерфейсом користувача та інструментами розробки програмного забезпечення. Ідеально підходить для початку роботи з даними, але продуктивність не розрахована на великі проекти. У *MS Access* можна використовувати наступні мови програмування *C, C #, C ++, Java, VBA* і *Visual Rudimental.NET*. *Access* зберігає всі таблиці бази даних, запити, форми, звіти, макроси і модулі в базі даних *Access Jet* у вигляді одного файлу. До особливостей системи можна віднести: можливість використовувати *VBA* для створення багатофункціональних рішень з розширеними можливостями управління даними і призначеним для користувача контролем та експорт в формати *Excel, Outlook, ASCII, dBase, Paradox, FoxPro, SQL Server* і *Oracle* [9].

MongoDB – це документо-орієнтована система керування базами даних (СКБД) з відкритим програмним кодом, яка не потребує опису схеми таблиці. *MongoDB* займає нішу між швидкими та масштабованими системами, що оперують даними у форматі ключ-значення, і реляційними СУБД, функціональними і зручними у формуванні запитів.

MongoDB підтримує зберігання документів в форматі, подібному до *JSON*, та має досить гнучку мову для формування запитів. Дана СУБД може створювати індекси для збереження атрибутів та ефективно забезпечує зберігання великих бінарних об’єктів даних. Підтримується відслідковування операцій зі зміни та додавання даних в БД, може працювати відповідно до парадигми паралельних та розподілених обчислень *Map/Reduce*, підтримує реплікацію і побудову відмовостійких конфігурацій [10]. У *MongoDB* є вбудовані засоби для забезпечення розподілу набору даних по серверах на основі певного ключа. Комбінуючи це, можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови, а це означає, що збій будь-якого вузла не позначається на роботі всієї БД.

Підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу.

Основні можливості *MongoDB*:

– документо-орієнтоване сховище (проста та потужна *JSON*-подібна схема збереження даних);

– гнучка мова для формування запитів до бази даних;

– динамічні запити;

– повна підтримка індексів;

– профілювання запитів;

– ефективне зберігання даних великих обсягів в бінарному форматі, наприклад, фото та відео;

– журналювання операцій, що модифікують дані в БД;

– підтримка відмовостійкості і масштабованості: асинхронна реплікація та набір реплік;

– може працювати відповідно до парадигми *Map/Reduce.*

СУБД керує наборами *JSON*-подібних документів, які зберігаються в бінарному форматі в форматі *BSON*. Зберігання та пошук файлів в *MongoDB* відбувається завдяки викликам протоколу *GridFS*. Ця функція, яка називається *grid file system*, входить до складу драйверів *MongoDB. MongoDB* надає розробникам функції для маніпулювання файлами та їх вмісту. Доступ до *GridFS* можна отримати за допомогою утиліти *mongofiles* або розширення для *Nginx* і *lighttpd*. *GridFS* ділить файл на частини і зберігає кожен з цих фрагментів як окремий документ. Так як і багато інших документо-орієнтованих СКБД, таких як *Couchbase*, *MarkLogic*, *CouchDB* та багато інших, *MongoDB* являється не реляційною СУБД [7].

Більшість усіх існуючих СУБД розроблені на мовах програмування *C* та *C++*, оскільки ці мови програмування надають максимальну швидкість процесів зчитування, обробки та запису даних в пам’ять машини та можливість прямого керування пам’яттю.

Мова програмування *C++* – статична, типізована мова програмування загального призначення. Підтримує парадигми процедурного програмування, об'єктно-орієнтованого програмування та загальне програмування. Мова має багату стандартну бібліотеку, яка включає в себе розповсюджені контейнери та алгоритми, ввід та вивід, регулярні вирази та ін. *C++* широко використовується для розробки програмного забезпечення, є одною з самих популярних мов програмування. Область застосування включає створення операційних систем, різноманітних прикладних програм, драйверів пристроїв, програмних додатків для вбудованих систем, серверів з високим рівнем оптимізації, а також ігор.

Transact-SQL (*T-SQL*) – процедурне розширення мови *SQL*, створене компанією *Microsoft* для *SQL Server* та *Sybase* для *Sybase ASE*. Мова була розширена такими можливостями: керуючі оператори, локальні та глобальні змінні, різноманітні додаткові функції для обробки строк, даних та математичних виразів, підтримка аутентифікації *Microsoft Windows*.

Мова є ключом до використання *MS SQL Server*. Усі додатки, незалежно від їх реалізації та користувацького інтерфейсу, відправляють серверу інструкції *Transact-SQL*.

1.3. Технології конвертації даних при моделюванні споруд

Конвертація даних при інформаційному моделюванні споруд досить складний процес, що проявляє всі свої можливості лише при застосуванні різностороннього підходу. Процес конвертації має бути зробленим таким чином, щоб забезпечувати максимальну зручність, швидкість, та мінімальну ресурсозатратність. Від програміста вимагаються доскональні знання в таких сферах роботи з даними як: робота з базами даних, в тому числі і з реляційними, робота з файлами та робота з форматами розмітки файлів [11].

Конвертація складається з наступних етапів: створення файлу моделі вихідного формату, читання всіх необхідних даних з файлу моделі, створення

параметрів конвертації, експорт усіх баз знань, експорт налаштувань моделі, експорт усіх об’єктів моделі та експорт даних про існуючі конструкції чи зборки.

Якщо всі етапи конвертації пройшли успішно, то користувач зможе відкрити вихідний файл моделі в відповідній програмі та перевірити якість конвертації. В деяких випадках відкорегувати зсуви чи відмінності в геометрії об’єктів, або власноруч імпортувати дані, конвертація яких не була вибрана при налаштуванні параметрів конвертації [12].

1.4. Висновки до розділу

У даному розділі детально розглянуто поняття інформаційного моделювання споруд, переваги та недоліки використання *BIM*-технології, етапів життєвого циклу проекту будівництва та переваги використання технології порівняно з традиційними системами комп’ютерного проектування.

Таким чином, інформаційне моделювання споруд (*building information modeling*, *BIM*) – це технологія оптимізації процесів проектування та будівництва, основа якої полягає в використанні єдиної моделі споруди та обміні інформацією будь-якого об’єкта між усіма учасниками протягом усього життєвого циклу – від задумки власника та перших начерків архітектора до технічного обслуговування готової споруди.

Розглянуто існуючі засоби зберігання інформації при інформаційному моделюванні споруд. При інформаційному моделюванні споруд оперують такими даними: параметри проекту, інформаційні моделі, об’єкти моделі, конструкції, параметри, бази знань та журнали подій.

Розглянуто поняття систем керування базами даних (СУБД) – це програмна система, що реалізує набір інструментів для створення, адміністрування та видалення баз даних підтримуваного формату. Детально порівняно переваги та недоліки таких баз даних: *MySQL*, *Microsoft SQL Server*, *PostgreSQL*, *MongoDB* та *Microsoft Access*. До переваг перерахованих бази даних

відносять легкість використання, безпечність, швидкість виконання запитів, підтримку багатьох операційних систем, інструменти візуалізації даних, динамічні запити та гнучкі мови виконання запитів до бази даних.

Розглянуто існуючу технологію конвертації даних, зазначена мета та цілі виконання процесу конвертації. Також перераховані наступні етапи конвертації даних: створення файлу вихідного формату, зчитування усіх необхідних даних з файлу, створення параметрів конвертації, експорт усіх баз знань, експорт налаштувань та всіх об’єктів.

РОЗДІЛ 2
АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ КОНВЕРТАЦІЇ ДАНИХ ПРИ ІНФОРМАЦІЙНОМУ МОДЕЛЮВАННІ СПОРУД

2.1. Огляд існуючих рішень в галузі конвертації даних

Значну частину програм займають архітектурні, конструкторські та обчислювальні програмні системи, основані на використанні *BIM* технології. Серед програмних систем даного типу слід виділити як архітектурні *Revit* та *ArchiCAD,* які дозволяють працювати над проектом одночасно декільком спеціалістам різного напряму, так і конструкторські *PC LIRA* 10.10 та *Advance Steel,* для вирішення усього спектру завдань, пов'язаних з усіма етапами проектування та будівництва споруди. Вони замінили собою звичне представлення схем об’єктів у вигляді креслень. Такі програмні системи дозволяють швидко побудувати та обчислити основні характеристики об’єктів різноманітного призначення та форми за допомогою зручного та компактного пакету функцій, засобів та пакету інтерфейсу користувача [5]. Користувачем програмних засобів в основному є архітектори та інженери.

Зазвичай програмні системи цього типу складаються з багатьох компонент до яких можуть входити: компоненти інтерфейсу користувача, компоненти представлення моделі об’єкта, компоненти відображення моделі, база даних, компоненти роботи з базами даних та ін., компоненти конвертації даних моделі. Окремі компоненти програми можуть бути написані як на одній мові, так і на декількох мовах програмування, до яких можна віднести: *C++, C#, Java, Objective C* та інші. Модульність програми забезпечує логічну послідовність процесів у її роботі, а також дозволяє більш ефективно розділяти робочі навантаження при її розробці.

Revit – програмний продукт компанії *Autodesk* призначений для створення спільних, узгоджених і повних проектів на основі моделей. Програма забезпечує

проектування, планування, розробкою дизайну, конструювання та керування будуванням конструкцій та інфраструктури. Також підтримує одночасне застосування декількох дизайнерських дисциплін при моделювання однієї будівлі, застосування інтелектуальних моделей для планування, проектування, будівництва та експлуатації будівель і об'єктів інфраструктури.

Програма дозволяє швидко створювати варіанти проектів з урахуванням цілей та заданих обмежень, створювати, використовувати параметричні компоненти та працювати над проектом одночасно декільком спеціалістам різного напряму.

ArchiCAD – програмний пакет для архітекторів, заснований на технології інформаційного моделювання, створений фірмою *Graphisoft*. Призначений для проектування архітектурно-будівельних конструкцій і рішень, а також елементів ландшафту, меблів і т. п.

При роботі в пакеті використовується концепція віртуального будування. Її суть полягає в тому, що проект *ArchiCAD* представляє віртуальну модель реальної будівлі, що існує в пам'яті комп'ютера. Для її виконання проектувальник на початкових етапах роботи з проектом фактично будує будинок, використовуючи при цьому інструменти, які мають свої повні аналоги в реальності: стіни, перекриття, вікна, сходи та інші різноманітні об'єкти. Завершивши етап моделювання, користувач може отримати з віртуальної будівлі всі необхідні дані для створення проектної документації: плани поверхів, фасади, розрізи, експлікації, специфікації та візуалізації. *ArchiCAD* є одним з перших додатків в *AIC*-індустрії, що реалізував підтримку підходу *OPEN BIM* на основі міжплатформенного формату взаємодії *IFC*.

Програмна система *PC LIRA* 10.10 дозволяє вирішувати весь спектр завдань, пов'язаних з проектуванням і будівництвом споруд будь-якої складності та рівня відповідальності

– підбір або перевірку перетинів сталевих і армування залізобетонних конструкцій;

– розрахунок на статичні та динамічні навантаження;

– геометрично та фізично нелінійні розрахунки;
– розрахунки з урахуванням поетапності зведення конструкцій;
– та багато інших типів завдань.

Програмні комплекси сімейства *LIRA* мають більш ніж сорокарічну історію створення, розвитку та застосування в наукових дослідженнях і практиці проектування конструкцій. Програмна система безперервно удосконалюється і пристосовується до нових операційних систем і графічних середовищ. Актуальною версією розрахункового комплексу *PC LIRA* є версія 10.10.

Усі галузі, які працюють з великими об'ємами даних, потребують програмні рішення конвертації даних, призначені для конвертації одного з форматів файлів в інший. В залежності від вимог та ступеня підготовки користувача існують програми як з попередньо встановленими режимами конвертації, так і з великими можливостями налаштування процесів конвертації [3]. Усі програмні рішення в галузі конвертації поділяються на вбудовані модулі конвертації даних в програмних системах керування баз даних та відокремлені програмні системи або утиліти, призначені для конвертації файлів різних форматів.

Серед вбудованих компонент можна виділити *EMS SQL Manager for SQL Server* та *Access2PostgreSQL PRO*, оскільки разом вони охоплюють дві бази даних, що часто використовуються в сімействі операційних систем *Windows*. Натомість, до програмних утиліт можна віднести *Access2MySQL SYNC* та *MyXMLData* через наявність у вільному доступі, швидкість проведення конвертації та спрямованість виконання процесів конвертації для вузького списку форматів.

EMS SQL Manager for SQL Server – потужний інструмент для розробки та адміністрування *Microsoft SQL Server и MSDE*. Працює з усіма версіями *SQL Server* починаючи з сьомої та закінчуючи версією 2005 року включно. Програма підтримує всі нові функції *SQL Server* включаючи збірки, *DDL,* тригери, колонки таблиць *XML* та систему розширень. Пакет включає в себе об'ємний інструментарій для досвідчених користувачів, до якого входять просте та

зрозуміле керування всіма об’єктами баз даних, ефективне управління даними та можливості конвертації даних в такі формати як *DOCX, XML, XLS, SQL, MDB, ACCDB, CSV* та інші [4].

До переваг програмної системи можна віднести: зручний інтерфейс, швидку навігацію при керуванні баз даних, об’ємний інструментарій для процесів конвертації даних. Серед недоліків можна виділити: вбудованість інструментів конвертації в код програми, надлишковість засобів при конвертації простих форматів даних, неможливість створювати та виконувати конвертацію з використанням налаштувань.

Access2PostgreSQL PRO – ефективний конвертор, здатний конвертувати *MS Access* (*mdb*) бази даних в *PostgreSQL* та навпаки – в бази даних *MS Access* шляхом простого визначення налаштувань в побудованому по принципу *Wizard* додатку. Програма дає можливість скопіювати всю базу даних повністю або конвертувати тільки окремі її об’єкти. У випадку якщо доступ до *PostgreSQL* бази даних закритий, дані можуть бути збережені у вигляді *PHP*-скрипта. Даний файл буде скопійований на сервер, на якому ведеться робота з конкретною *PostgreSQL* та надалі може бути відкритим з будь-якої машини, яка має доступ до інтернету.

До переваг програмного засобу можна віднести: зручний інтерфейс користувача, програмування комп’ютера за допомогою вбудованого планувальника задач *ostgreSQL* та високу швидкість конвертації даних. До недоліків програмного засобу можна віднести: малий перелік підтримуваних форматів та операційних систем.

Access2MySQL SYNC – програмна система, що надає можливість синхронізувати та конвертувати *MS Access* та *MySQL* бази даних. Програма не тільки дозволяє конвертувати дані, виконуючи синхронізацію, але й дозволяє вибирати окремі таблиці для конвертації, задавати параметри роботи програми через командну строку та вибирати між можливістю копіювати ідентичні записи з вхідної бази даних в вихідну, або ж пропускати однакові записи. Користувач також може вибрати тип таблиці *MyISAM, HEAP, BDB* або *ISAM*.

Серед переваг програмного засобу *Access2MySQL* можна виділити: можливість вибирати тип таблиці, копіювати ідентичні записи та мінімальні вимоги до системних характеристик машини. До недоліків можна віднести: малу кількість підтримуваних форматів та підтримку лише сімейства операційних систем *Windows*.

MyXMLData – програма розроблена на мові програмування *Java* для зберігання даних в форматі *XML,* додатково можливо індексувати зображення та текстові описи до них. Підтримує формати зображень *jpeg*, *jpg*, *gif* та *png*. При конвертації даних всі добавлені коментарі, описи та інші великі текстові дані індексуються та відображаються.

До переваг програмного засобу *MyXMLData* можна віднести швидкість конвертації та додавання даних, декілька демонстраційних баз даних та відновлення пошкоджених даних. Серед недоліків можна виділити низьку швидкість конвертації даних та порівняно складний інтерфейс користувача.

2.2. Підходи для вирішення проблем конвертації даних

Серед усіх підходів до вирішення проблем конвертації даних часто використовуються наступні: використання бази знань правил конвертації; використання перехідного формату; використання вбудованих інструментів конвертації; використання резервного копіювання. Усі перераховані підходи дозволяють спростити процес створення програмних рішень у галузі конвертації даних або зменшити використання системних ресурсів під час виконання процесів конвертації.

Підхід, що базується на використанні резервного копіювання як адміністрування даних – процес створення копії даних перед початком конвертації в інший формат, призначений для відновлення пошкоджених, пропущених чи видалених даних. При конвертації великих обсягів даних з одного формату в інший завжди існує можливість втрати деякої частини цих

даних. Деяка кількість втрат даних може не мати наслідків, особливо якщо ці дані несуттєві. Подібним чином деякі втрачені дані можна легко відновити за допомогою резервних файлів. Але деякі типи втрат даних набагато серйозніші. Навіть якщо відкинути потенційну катастрофу втрати конфіденційної або приватної інформації, яку потрібно захистити, втрата даних може створити небажаний ефект, який припиняє частини процесу конвертації. Якщо втрата даних залишається поза увагою *IT*-персоналу, ніхто не може усвідомити, що важливі дані відсутні, поки програма не вийде з ладу через відсутні дані.

Підхід використання баз знань форматів та правил конвертації даних дозволяє автоматизувати процес транслювання даних при повторній конвертації файлів зазначених форматів. Підхід часто використовується у випадку, коли необхідно конвертувати дані із заздалегідь невизначеними вхідним або вихідним форматами. Базується на транслюванні даних за заздалегідь встановленими правилами перетворень, що частіше за все завантажуються в пам'ять процесу перед початком конвертації. Правила зберігаються в окремому файлі, здебільшого формату *xml* або вставляються і зберігаються разом з об'єктами вхідного чи вихідного форматів. В разі якщо правила перетворень для конкретної комбінації форматів не визначені, то користувачу надається можливість власноруч провести відповідність між об’єктами, дані яких повинні бути трансльовані. В окремих випадках правила можуть мати вигляд формул з використанням арифметичних операцій та операторів умови, де в лівій частині формули визначається колонка з вихідної таблиці, в яку будуть записані дані, а в правій колонці з таблиці вхідного формату – правила, які будуть зчитані.

Підхід, що базується на використанні третього, перехідного формату даних зустрічається рідше чим з двома форматами даних. Потреба у використанні підходу виникає у випадку, коли неможливо створити швидкі або універсальні алгоритми перетворень між вхідним та вихідним форматами. У такому випадку дані спершу транслюються з початкового формату у перехідний формат, після чого з перехідного формату транслюються у кінцевий формат. Цей підхід дозволяє зменшити час та ресурси, затрачені на створення правил

перетворень, оскільки без нього кількість правил, які необхідно створити, збільшується експоненціально. Схематичне зображення відмінності підходів зображено на рисунку 2.1.

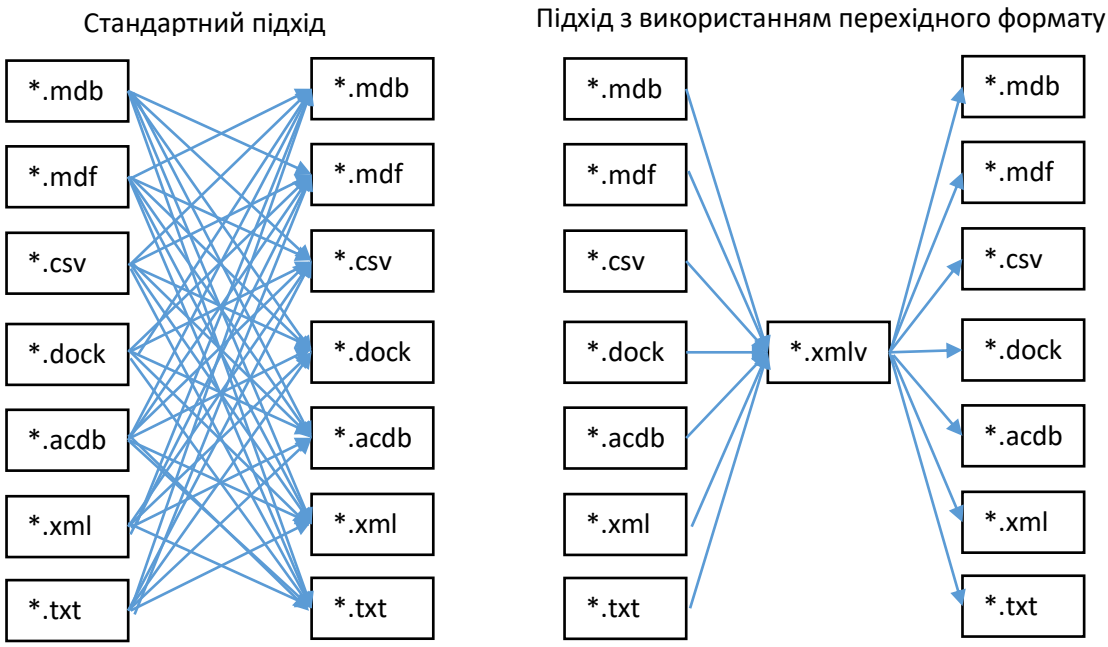

Рис. 2.1. Схематичне зображення відмінності підходу з використанням перехідного формату

На схемі відображено відмінність стандартного підходу побудови системи конвертації в середовищі, що використовує сім видів форматів даних, від підходу з використанням додаткового перехідного формату. Кожен блок представляє тип формату даних, що підтримується середовищем (наприклад операційною системою), в той час як стрілки позначають правила конвертації даних між цими форматами.

При стандартному підході з безпосередньою конвертацією кількість правил буде експоненціально залежною від кількості форматів в системі. Кількість необхідних для визначення правил визначається за формулою 2.1.

$$N_p = N_f * (N_f - 1) \quad (2.1)$$

де N_p – кількість правил, які необхідно визначити;

N_f – кількість підтримуваних програмною системою форматів.

При підході з використання перехідного формату кількість правил буде лінійно пропорційною кількості форматів в системі, та розраховуватися за формулою 2.2.

$$N_p = N_f \tag{2.2}$$

де N_p – кількість правил, які необхідно визначити;

N_f – кількість підтримуваних програмною системою форматів.

Підхід зменшує кількість необхідної роботи та системних ресурсів та має наступні недоліки: необхідність знаходження потрібного перехідного формату, який повинен забезпечувати максимальну простоту реалізації правил обміну даними з іншими форматами, збільшення часу виконання процесу конвертації через необхідність виконання двох наборів правил та необхідність користувачеві визначати одразу два набори правил у випадку, якщо в базі знань їх не буде.

Підхід використання вбудованих інструментів конвертації в програмних системах, призначених для роботи з відповідними форматами, зустрічається досить рідко, та базується на використанні компонент конвертації даних тих програм, які спеціалізуються на роботі з форматами, що транслюються. На перший погляд підхід схожий на підхід з використанням перехідного формату, але замість використання перехідного формату підхід формує абстрактні об’єкти, аналоги реальних. Алгоритм роботи програмної системи, що використовує даний підхід, можна поділити на наступні етапи: завантаження даних з вхідного формату за допомогою спеціальних компонент програмної системи, які підтримує перший формат, аналіз даних та формування абстрактних об’єктів які використовуються в програмному засобі, що підтримує перший формат даних (вихідний), приведення абстрактних об’єктів до вигляду, що використовується в програмному засобі, який підтримує другий формат даних (вхідний) та етап збереження даних в файл вихідного формату за допомогою

спеціальних компонент програмної системи, який підтримує вихідний формат файлів.

Перевагами такого підходу можна назвати відсутність потреби створювати правила перетворень даних та швидкість процесу конвертації. До недоліків належать: використання зовнішніх *DLL*, потреба постійного контролю версій програмного засобу та відсутність, як такого, інтерфейсу користувача.

2.3. Проблеми конвертації даних при моделюванні споруд

Першим кроком до перетворення даних є розуміння різних типів даних, які можуть бути конвертовані. Усі програмні мови використовують типи даних, які повідомляють компілятору або інтерпретатору, як використовувати дані. Тип даних визначає операції, які можна виконувати над даними, і визначає структуру, в якій дані повинні зберігатися. Більшість типів даних (як примітивних, так і складених) можна перетворити. Точне, повне перетворення даних має важливе значення для програм, що використовуються професіоналами, які залежать від наявних даних.

Конвертація даних – перетворення даних з одного формату в інший. Зазвичай із збереженням основного логічно-структурного змісту інформації. Потреба в конвертації даних виникає через необхідність програмних засобам працювати з декількома форматами даних [5]. Перетворення даних пов'язане з різницею логічних структур даних, а також з такими проблемами:

– багатомодельність представлення даних (ієрархічні, мережні, реляційні) в різних БД чи форматах;

– різниця в логічних структурах даних, в довідниках, класифікаторах і в системах кодування інформації;

– використання різних мов для представлення текстової інформації;

– різні типи форматів і постійний розвиток даних БД в процесі експлуатації;

– різні формати даних;

– складності створення універсального алгоритму конвертації.

Пряма конвертація – читання даних з першого файлу (вхідного формату) та збереження їх в файлі іншого (вихідного формату) з урахуванням усіх лінгвістично-структурних відмінностей. При прямій конвертації даних програмна система працює напряму з файлами даних виконуючи зчитування та наступний запис даних в файл вихідного формату.

Непряма конвертація – вид конвертації даних, при якому робота зчитування та запису даних виконуються засобами програмних систем (утилітами), які спеціалізуються на редагуванні та відображенні файлів даного формату, або їх окремі модулі чи бібліотеки. При непрямій конвертації даних використовуються засоби роботи з файлами форматів. Перетворення даних відбувається не між файлами, а між об’єктами моделі. Дані зчитуються з файлу вхідного формату та перетворюються в образи об’єктів. Після цього, дані перетворюються у вигляд об’єктів вихідного формату та додаються безпосередньо в модель іншої програми [5].

У залежності від способу зберігання правила конвертація поділяється на: конвертацію зі заздалегідь визначеними форматами даних, конвертацію з обумовленими правилами перетворень даних, де правила визначаються безпосередньо під час конвертації, та конвертацію з використанням бази знань форматів.

Конвертація з заздалегідь визначеними форматами даних зчитує та записує дані за заздалегідь визначеними правилами, здебільшого вшитими в код програмного засобу, що виконує перетворення даних. Конвертація з обумовленими правилами розділена на два етапи. Спершу користувачу надається можливість визначити правила, після чого відбудеться безпосереднє зчитування та запис даних. Конвертація даних з використанням бази знань форматів дозволяє зберігати та завантажувати (при необхідності змінювати) правила перед обміном даних.

Перетворення даних може бути складним процесом, хоча це не обов’язково. Інструменти для автоматизації процесу можуть покращити як точність, так і повноту перетворених даних, одночасно скорочуючи час розробки. До ключових задач при підготовці правил обміну входять: визначення конфігурації вхідного та вихідного форматів, загальне налаштування опцій конвертації, створення відповідностей між об’єктами метаданих вхідного та вихідного джерела, налаштування відповідностей між властивостями об’єктів в рамках правил конвертації, принципів вибірки об’єктів для вигрузки, операцій виконуваних перед завантаженням даних та додаткове налаштування оброблювачів подій.

Поняття конвертації даних часто путають з міграцією даних, трансформацією даних чи чисткою даних. Конвертація даних транслює окремі комп’ютерні об’єкти та типи даних з одного формату до іншого, а процес міграції переводить цілі бази даних або програми з одного місця збереження в інше. Міграція даних часто спричиняє конвертацію та трансформацію даних. Процес трансформації даних на відміну від конвертації даних не транслює дані з одного формату в інший. Трансформація змінює самі данні. Чистка даних знаходить та виправляє неправильні, неповні, чи дані, що повторяються.

2.4. Висновки до розділу

У даному розділі детально розглянуто архітектурні, конструкторські та обчислювальні програмні системи, які використовуються в *BIM*-технологіях. Усі програмні рішення в галузі конвертації даних були поділені на вбудовані модулі конвертації даних в програмних системах керування баз даних та відокремлені програмні системи або утиліти, призначені для конвертації файлів різних форматів.

Були детально описані вбудовані модулі конвертації даних програмних систем *EMS SQL Manager for SQL Server* та *Access2PostgreSQL PRO*, оскільки

разом вони охоплюють дві бази даних, що часто використовуються в сімействі операційних систем *Windows*. Були описані відокремлені програмні засоби *Access2MySQL SYNC* та *MyXMLData* через наявність у вільному доступі, швидкість проведення конвертації та спрямованість виконання процесів конвертації для вузького списку форматів.

Розглянуті та детально описані підходи для вирішення проблем конвертації, що основані на використанні баз знань правил конвертації, перехідних форматів, вбудованих інструментів конвертації та резервного копіювання як адміністрування. Усі перераховані підходи дозволяють спростити процес створення програмних рішень у галузі конвертації даних або зменшити використання системних ресурсів під час виконання процесів конвертації.

Детально розглянуто поняття конвертації даних, прямої конвертації даних та непрямої конвертації даних. Перераховані та детально пояснені основні проблеми конвертації даних, що включають: багатомодельність представлених даних, різницю в логічних структурах даних, використання різних мов для представлення текстової інформації та наявність різних типів форматів і постійний розвиток даних в процесі їх експлуатації.

Таким чином конвертація даних (*data conversion*) – це перетворення даних з одного формату в інший. Зазвичай із збереженням основного логічно-структурного змісту інформації. Потреба в конвертації даних виникає через необхідність програмним засобам працювати з декількома форматами даних.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ КОНВЕРТАЦІЇ ДАНИХ ПРИ ІНФОРМАЦІЙНОМУ МОДЕЛЮВАННІ СПОРУД

3.1. Проектування програмної системи конвертації даних

До основних функцій програмної системи конвертації даних при інформаційному моделюванні споруд можна віднести наступні основні операції: підключення до баз даних формату **.mdf* засобами системи керування базами даних *Microsoft SQL Server*, читання та відображення вибраних даних у вигляді таблиць даних, створення правил імпорту та експорту даних, читання та збереження створених правил з бази знань правил конвертації формату **.xml*, виконання конвертації даних та їх відображення в інтерфейсі програмної системи.

Програмна система – різновид програмного забезпечення, що використовується для розв'язання всього спектру задач, що можуть виникнути при вирішенні поставленої проблеми. Розрізняють прикладні програмні системи, статистичні програмні системи та інструментальні програмні системи (комп'ютерні програми, призначені для проектування, розробки та адміністрування) [16].

Під підключенням до баз даних формату **.mdf* (*main data file*) мається на увазі підключення через стандартну аутентифікацію *Windows* до існуючої на машині версії *SQL Server* та відкриття вибраної бази даних. Після чого, програмна система може створювати запити до відкритої бази даних на мові *Transact-SQL*.

Для читання та відображення вибраної інформації у вигляді таблиці даних необхідно відображати список з назвами всіх наявних в базі даних об'єктів. Після вибору одного з об'єктів, його дані мають відобразитись у

вигляді таблиці, що дозволить користувачеві програми ефективно орієнтуватися в виборі необхідних даних для їх подальшої конвертації.

Під процесом створення правил імпорту та експорту даних мається на увазі визначення користувачем програмної системи колонок, між якими буде виконано транслювання даних, та формули, за якими буде виконуватися транслювання.

Створенням правил імпорту – це процес визначення правил транслювання інформації з вихідного набору даних у вхідний. У вигляді правила імпорту записується формула, за якою дані при виконанні процесу конвертації будуть зчитані з однієї або декількох таблиць вихідного формату та записані в таблицю вхідного формату.

Створенням правил експорту – це процес визначення правил транслювання інформації з вхідного набору даних у вихідний. Правило експорту визначає формулу, за якою дані при виконанні процесу конвертації будуть зчитані з однієї або декількох колонок таблиці одного формату та записані в колонку таблиці іншого формату.

Під збереженням створених правил в базі знань правил конвертації мається на увазі надання користувачеві можливості створювати нові, відкривати, редагувати та зберігати правила в уже існуючий, або новий файл формату **.xml*. Для збереження правила достатньо зазначити назву колонки, в яку дані будуть записані, та правило (формулу, по якій значення будуть обчислені).

Програмна система має виконувати конвертацію даних по створеним та збереженим в файл бази знань конвертації правилам. Створений пустий об’єкт таблиці з структурною схемою вихідного формату даних буде заповнюватися записами з вхідної таблиці та відображатися користувачеві для порівняння результатів [17].

Для проектування програмної системи було використано програмне середовище розробки *Microsoft Visual Studio* 2019. Це інтегроване середовище розробки та проектування програмного забезпечення та низки інших інструментальних засобів. Продукт дозволяє розробляти як консольні програми,

так і програми з графічним інтерфейсом, в тому числі з підтримкою технології *Windows Forms*, а також веб-сайтів, веб-застосунків, веб-служб як в рідному, так і в керованому кодах для всіх платформ. Середовище розробки підтримує наступні операційні системи: *Microsoft Windows*, *Windows Mobile*, *Windows Phone*, *Windows CE*, *.NET Framework*, *.NET Compact Framework* та *Microsoft Silverlight*.

При проектуванні функцій програмної системи було спроектовано п’ять компонент: компонента відображення даних, компонента реалізації *SQL* запитів, компонента роботи з *XML* форматом та компонента створення правил конвертації. Структурна схема програмної системи конвертації даних зображена на рисунку 3.1.

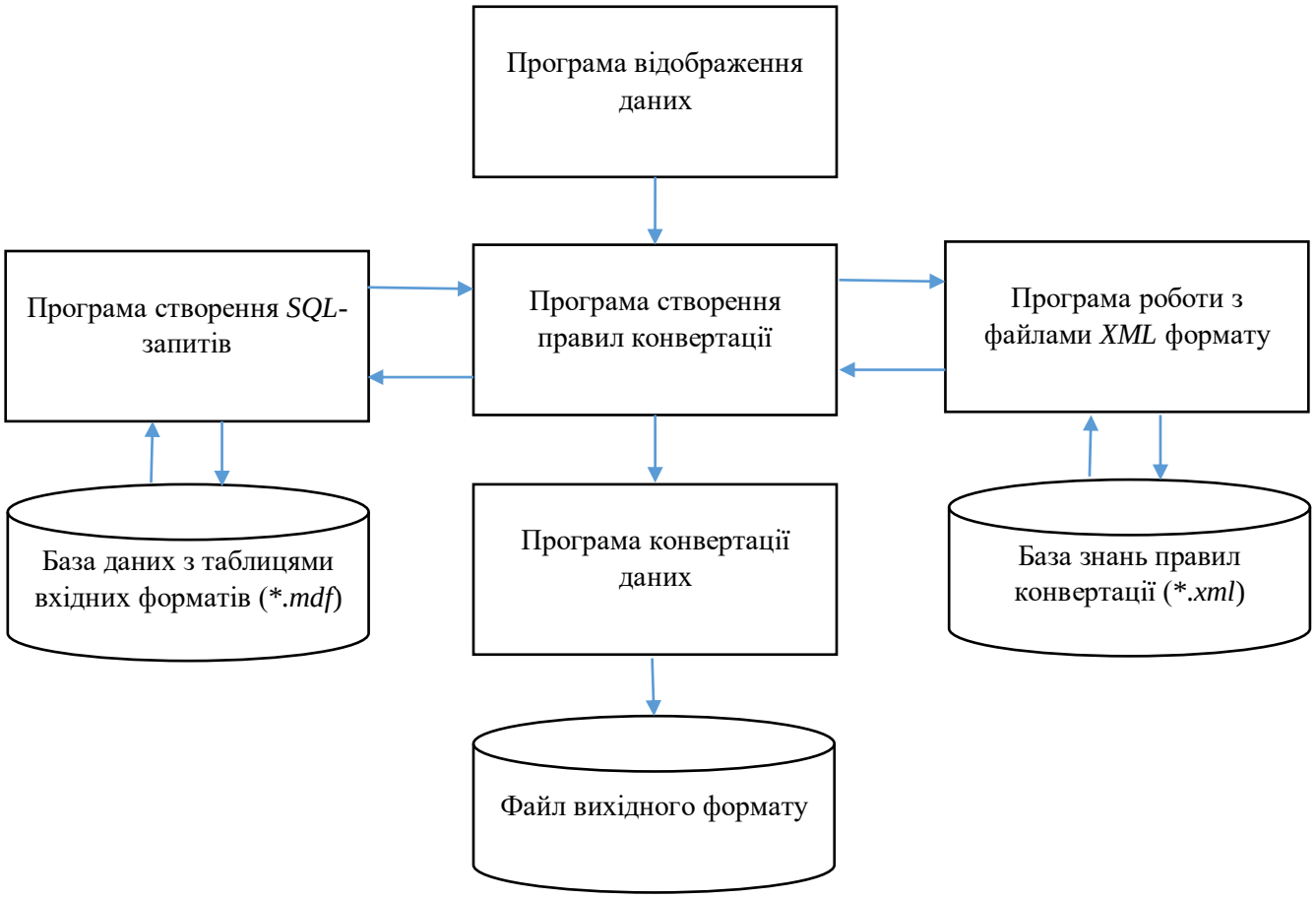

Рис. 3.1. Структурна схема програмної системи конвертації даних

3.2. Проектування програми відображення даних

Програмна компонента відображення даних складається з форм, таблиць та компонент, призначених для виведення на екран інформації. До таких даних можуть входити дані вибраних з бази даних об’єктів таблиць, їх колонки, строки та структурні схеми. До спроектованих функцій входять: відображення списку таблиць з вибраної бази даних, відображення даних вибраної таблиці, відображення структурних схем таблиць вхідного та вихідного форматів, відображення даних таблиці, створеної в результаті конвертації даних, відображення інформації щодо усіх виведених даних та відображення послідовностей дій.

До структурної схеми таблиці входять перелік усіх колонок таблиці та перелік відповідних типів даних, що зберігаються в колонках. Такий набір даних дає повне представлення про структури даних, що збережені в таблиці.

В якості прикладу структурної схеми таблиці бази даних слід навести структурні схеми таблиць типів профілів, що використовуються при моделюванні *BIM* моделей. До таких типі відносять таблиці двутаврів, таврів, каналів, уголків, труб, колон та балок.

Структурна схема таблиці двутаврів буде складатися з таких колонок (полів даних):

– *Section name* (тип даних *char*(255)) – поле назви форми профілю;

– *height full* (тип даних *nvarchar*(30) або *double*) – поле містить величину повної висоти форми профілю;

– *width* (тип даних *nvarchar*(30) або *double*) – поле, що містить значення повної ширини форми профілю;

– *ts* (тип даних *nvarchar*(30) або *double*) – поле, що містить значення товщини центральної стінки форми профілю;

– *tf* (тип даних *nvarchar*(30) або *double*) – поле, що містить значення товщини стінок полок форми профілю;

– *R* (тип даних *nvarchar*(8) або *double*) – поле, що містить значення радіусу згинання біля центральної стінки форми профілю;

– *r* (тип даних *nvarchar*(8) або *double*) – поле, що містить значення радіусу згинання по краям полок форми профілю;

– *k flange* (тип даних *nvarchar*(8) або *double*) – поле, що містить кут нахилу полок по відношенню до центральної стінки форми профілю.

Разом поля даних дають достатню кількість інформації для обчислення розрахункових параметрів, таких як: площа поперечного перерізу, вага на метр, момент інерції по осі *Y*, момент інерції по осі *Z*, статичний момент, секторіальна площа та секторіальний момент інерції. Також даний перелік параметрів дозволяє побудувати двовимірний контур перерізу профілю.

При початку процесу конвертації користувач має вибрати відповідні поля колонок у вихідній таблиці та прописати формули, за якими дані будуть трансльовані. Схема алгоритму конвертації даних зображена на рис. 3.2.

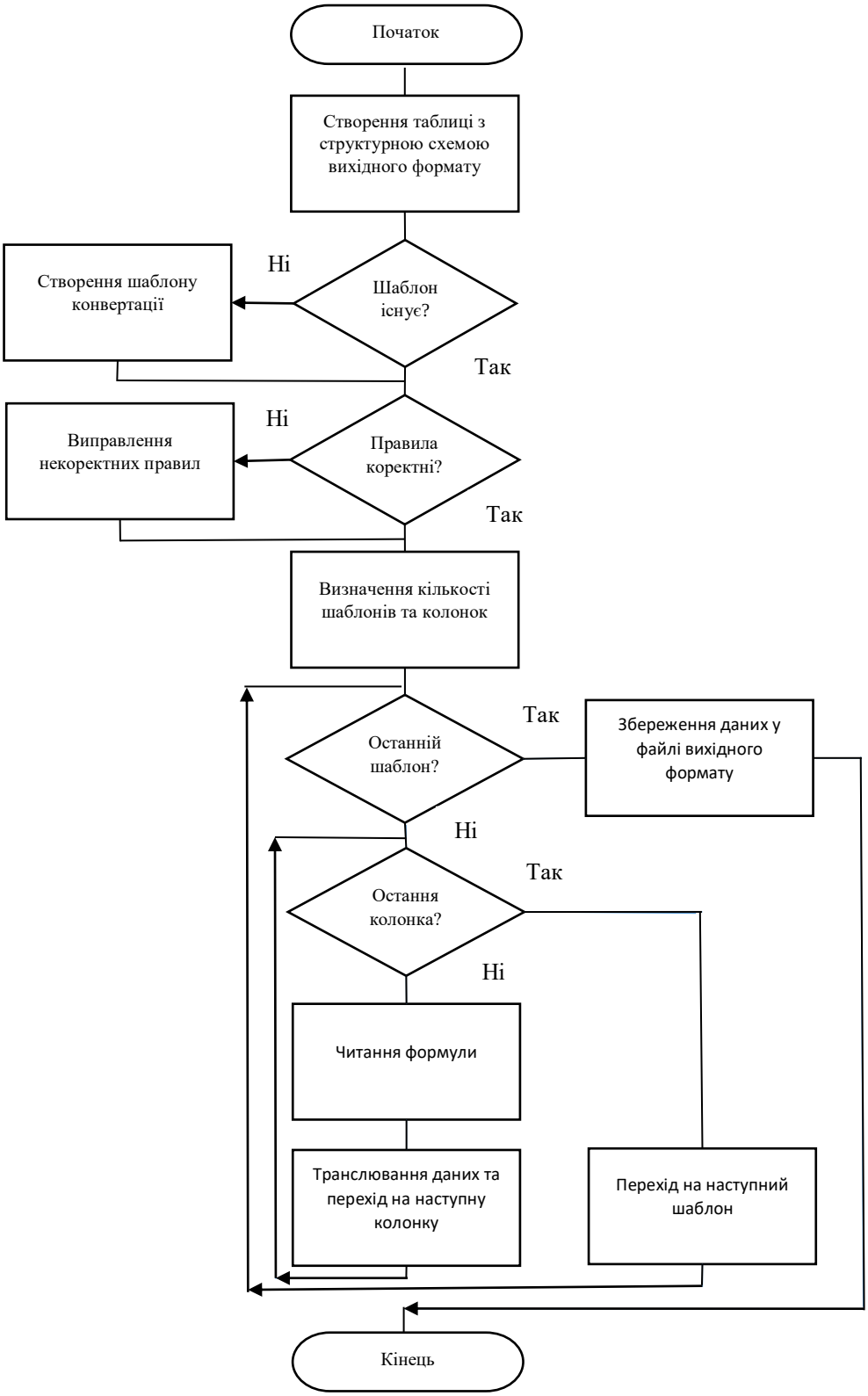

Рис. 3.2. Схема алгоритму конвертації даних

3.3. Проектування програми реалізації *SQL* запитів

Програмна компонента реалізації *SQL* запитів має використовуватися при зчитуванні чи збереженні у вибраній базі даних таблиць. Реалізація компоненти потребує створення об’єкта підключення до бази даних та виконання команд, написаних на мові запитів *Transact-SQL*. Було спроектовано наступні функції:

- вибір файлу бази даних формату **.mdf*;
- створення підключення до файлу бази даних;
- отримання списку усіх таблиць, збережених в базі даних;
- отримання структурної схеми вибраної користувачем таблиці;
- отримання збережених в таблиці даних включно з назвами стовбців;
- збереження нових та відредагованих таблиць.

Компонента має виконувати всі вище зазначені функції враховуючи показники оптимізації та швидкості виконання процесів. До команд та операторів, що можуть забезпечити виконання вимог при мінімальній складності запитів належать: *ALTER DATABASE*, *ALTER TABLE*, *SELECT*, *IDENTITY*, *PRIMARY KEY*, *CREATE TABLE*, *CREATE VIEW*, *DELETE*, *INSERT*, *UPDATE* та інші [3].

ALTER DATABASE – інструкція, що змінює базу даних або файли та файлові групи, пов’язані з базою даних. Додає або видаляє файли та файлові групи з бази даних, змінює атрибути бази даних та її файлів, змінює параметри сортування та створює параметри бази даних. При створенні бази даних з параметрами сортування, відмінними від параметрів сортування за замовчуванням, дані в базі даних завжди враховують параметри сортування. Для *SQL Server* при створенні автономних баз даних зведення внутрішніх каталогів підтримуються із використанням параметрів сортування *SQL Server*. Стан файлів в межах файлових груп визначає доступність файлових груп в цілому. Щоб файлова група була доступна, необхідно, щоб усі файли знаходилися в режимі системи. При створенні планів запитів для інструкцій *SELECT* оптимізатор

запитів уникає некластирезованних індексів та представлень, які знаходяться в файлових групах за межами доступу.

ALTER TABLE – дозволяє виконувати різноманітні операції з таблицями після їх створення (створення ключів, додавання зовнішніх ключів, констант, колонок та перевірок). При створенні або зміні таблиці за допомогою інструкції C*REATE TABLE* або *ALTER TABLE*, доступність значень *null* для типу даних, вказаного у визначенні колонки, залежить від параметрів бази даних та сеансу або навіть визначається ними. Якщо виконується перепризначення таблиці в одну із уже існуючих секцій, або переключення секції із одної таблиці на іншу, то цільова секція має існувати та бути пустою.

SELECT – команда вибору даних із зазначеною структурою. Також можна використовувати разом з оператором *UNION* або *EXEPT* для порівняння або об'єднання в одне ціле результуючого набору. Команду дозволено використовувати в визначених користувачем функціях тільки в тому випадку, якщо списки вибору цих інструкцій мають вирази, які привласнюють значення змінних, локальних для функцій. Порядок прив'язки інструкцій *SELECT* визначає ситуацію, коли об'єкти, визначені в одному кроці, стають доступними для виразів в наступних кроках. Наприклад, якщо обробних запитів може бути прив'язаним до таблиць та представлень, визначених в операторі *FROM*.

IDENTITY – команда, що використовується для вибору колонки з таким параметром, де сервером автоматично генерується зростаюча послідовність. Відлік починається із початкового значення, яке збільшується на величину інкремента. Якщо який-небудь параметр пропущений, то за замовчуванням зберігається одиниця.

PRIMARY KEY – обмеження, що може бути також застосоване для декількох колонок таблиці, що складають унікальну комбінацію значень.

CREATE TABLE – команда, що призначена для створення таблиць. Після створення загальної структури бази даних створюються таблиці, які являються відношеннями, що входять у склад проекту бази даних. Таблиці створюються пустими, тобто без строк. Команда визначає назву таблиці та множину

найменованих колонок у вказаному порядку. Для кожної колонки повинні бути визначеними тип та розмір.

CREATE VIEW – команда, використання якої дозволяє створювати представлення або подання. Припустимо, якийсь конкретний запит виконувався довгий час, то в такому випадку можливо тимчасову таблицю визначити цим конкретним запитом та створити одне нове представлення даних із отриманими даними тимчасової таблиці. Тепер безпосередній виклик подання, крім виконання цього запиту, дозволить отримати результат набагато швидше, ніж зазвичай. Команда *CREATE VIEW* допомагає створити те саме.

DELETE – команда, що використовується для видалення таблиці, конкретного стовпця таблиці, конкретного рядка таблиці або даних всередині таблиці. Зазвичай у командах *T-SQL* доступні три види видалення: видалення за допомогою деяких команд курсору, при якому видаляються деякі конкретні дані ключових стовпців. Застосування команди з використанням динамічних команд дозволяє користувачу видалити ключові дані з певною динамічною умовою, та видаляти кілька рядків на основі наданої умови.

INSERT – команда, що дозоляє вставити один або кілька рядків у таблицю. У цьому випадку доступні два типи вставок. Один вставляє рядки в таблицю або подання за допомогою певних динамічних умов. Таким чином користувач може вставити дані в таблицю на основі деяких динамічних умов, згаданих у запиті. При використанні іншого типу дані вставляються в таблицю або подання за допомогою написаної інструкції. У цьому випадку вставка може бути виконана на основі деяких умов, згаданих у команді.

UPDATE – команда оновлення в основному використовується для оновлення деяких існуючих рядків у таблиці. Три види оновлень, доступні в *T-SQL*: позиційне оновлення з використанням курсору, оновлення за допомогою динамічної команди одного виду, на основі згаданого запиту, за допомогою інструкції, що також є динамічним оновленням, але на основі конкретної інструкції.

BEGIN TRANSACTION – команда, що розпочинає транзакцію. Призначена для керування або збереження початкової точки управління транзакціями. Якщо користувач бажає виконати декілька запитів на обробку даних та здійснити один і той же самий запит один раз, то в такому випадку на початку транзакції необхідно вказати ідентифікатор запуску цих операторів *DML*.

COMMIT TRANSACTION – це в основному фіксація всієї транзакції в базі даних. При необхідності фіксації усього *DML* без будь-якої окремої команди слід використовувати *COMMIT TRANSACTION*. У тих місцях, де користувачі будуть вказувати фіксацію транзакції, програмою буде виконуватися фіксація усього виконаного *DML*, від початку транзакції до фіксації.

SET – команда, що виконується під час виконання або запуску. Якщо команда виконується в збереженій інструкції або тригері, значення параметра *SET* відновлюється після того, як збережена процедура або тригер вернуть управління. Також, якщо інструкція вказана в динамічному рядку *SQL*, який виконується за допомогою процедури *sp_executesql* або інструкції *EXECUTE*. Значення параметра відновиться після того, як керування вернеться з пакету, вказаного в динамічному рядку *SQL*.

CASE – команда, що перевіряє вираз, та виконує код в залежності від результату перевірки. Оператор *CASE* в залежності від зазначених умов повертає одне з безлічі можливих значень. У загальному випадку умовою є перевірка на *NULL*. Якщо ця умова виконується, то повертається текст “Нема в наявності”, в іншому випадку повертається необхідне значення. Оскільки результатом оператора *SELECT* завжди є таблиця, то всі значення будь-якої колонки повинні мати один і той же тип даних (з урахуванням неявного приведення типів). Тому неможливо поряд з числовим типом виводити символьну константу. Ось чому необхідно застосовувати перетворення типів – щоб привести значення до символьного поданням.

CREATE DEFAULT – команда, що створює об’єкт, який може бути прикріпленим до колонки або визначеного користувачем типу даних, при додаванні нового типу даних значення буде використано за замовчуванням.

CHECKCATALOG – команда, що перевіряє цілісність вибраних таблиць і відношень між ними.

DECLARE – оператор створення локальних змінних. Оголошує змінні за назвою (ім'я має мати одиничний символ "@" в якості першого символу), в якості підтримуваних системою типів даних. Для числових змінних також підтримується точність та масштаб та змінні типу *XML*, як узагальнена колекція схем.

EXECUTE – команда для запуску збережених процедур та інструкцій у вигляді текстових строк. Використання динамічного коду з командою *EXECUTE* є небезпечним. Для того, щоб виконати динамічну інструкцію необхідно використати змінні для динамічних змінюваних значень. Якщо ці значення будуть отримані від користувацького додатку, то існує імовірність передачі та вставлення в інструкцію шкідливого коду у вигляді тексту. Цей код буде виконаний в базі даних.

Транзакція *SQL* – це атомарна одиниця роботи, яка або повністю завершується успішно, або повністю завершується помилкою. Це набір інструкцій, що складаються із ідентифікаторів, параметрів, змінних, назв, типів даних та зареєстрованих у *SQL* словників. Інколи команда не задає початок транзакції. Аналітичні сервіси завжди фіксують неявну транзакцію, та відкатують її при збої команди.

Особливо важливими є інструкції з використанням оператора *SELECT*, який повертає строки бази даних та дозволяє робити вибір одного або декількох записів чи полів із декількох таблиць в базі даних *SQL Server*.

3.4. Проектування програми роботи з файлами *XML* формату

Програмна компонента роботи з файлами *XML* формату призначений для реалізації функцій читання, редагування та створення файлів баз знань правил конвертації. Компонента має задовольняти наступні вимоги: бути зручним у

використання при розробці програмної системи конвертації даних, реалізовувати максимально простий та зрозумілий формат *XML* розмітки для зберігання правил та виконувати процеси зчитування та запису даних в файл формату швидко та безперебійно [14].

Було спроектовано такі функції: вибір або створення файлу бази знань формату **.xml,* зчитування правил з файлу формату **.xml* та запис в оперативну пам'ять процесу програми та запис в файл формату **.xml* як нових правил, так і відредагованих.

Для забезпечення усіх вище перерахованих вимог прийнято рішення використати формат *XML* розмітки, що включає такі елементи розмітки: елемент *XML*-декларації, *Database*, *Template*, *Schema*1, *Schema*2, *Column*, *Name*, *Type*, *Transfer Rules*1, *Transfer Rules*2, *Conversation Rule*, *Column Name* та *Rule*.

Елемент *XML*-декларації – елемент розмітки в який записується інформація про вид формату розмітки та його версію. Зазвичай записується на початку документа.

Database – елемент розмітки, що представляє собою базу знань. В ньому зберігаються усі шаблони, структурні схеми та правила конвертації даних. Підмножиною елементу є список шаблонів.

Template – елемент розмітки, що представляє собою шаблон правил конвертації між двома конкретними форматами структурних схем таблиць. У файлі не можуть зберігатися два шаблони, що визначають правила конвертації між однаковими структурними схемами таблиць.

*Schema*1 – елемент розмітки, що містить дані структурної схеми таблиці вхідного формату. Містить список елементів *Column*.

*Schema*2 – елемент розмітки, що містить дані структурної схеми таблиці вихідного формату. Містить список елементів *Column*.

Column – елемент розмітки, що представляє собою колонку (поле даних) таблиці. Містить параметри назви колонки та її типу даних.

Name – параметр елементу розмітки *Column*, в якому зберігається назва колонки.

Type – параметр елементу розмітки *Column,* в якому зберігається тип даних колонки.

*Transfer Rules*1 – елемент розмітки, в якому зберігаються дані усіх правил імпорту. Підмножиною елементу є список елементів *Conversation Rule*.

*Transfer Rules*2 – елемент розмітки, в якому зберігаються дані усіх правил експорту. Підмножиною елементу є список елементів *Conversation Rule*.

Conversation Rule – елемент розмітки, що представляє собою правило конвертації даних. Містить параметри вихідної колонки та формули, за якою буде виконуватися процес транслювання даних.

Column Name – параметр елементу розмітки *Conversation Rule*, в якому зберігається назва вихідної колонки даних.

Rule – параметр елементу розмітки *Conversation Rule*, в якому зберігається правило, за яким буде виконано трансляцію даних між двома колонками таблиць вхідного та вихідного формату.

Алгоритм запису даних в файл *XML* формату починається з вибору користувачем файлу *XML* формату, що в подальшому буде використовуватися як база знань правил конвертації [15]. У випадку, коли зазначеного файл не існує, програма створить файл з указаною назвою у відповідній папці операційної системи та відкриє його в режимі запису файлів. Основним циклом алгоритму є цикл проходження по шаблонам правил конвертації. Для кожного шаблону в файл записуються структури таблиць вхідного та вихідного формату та всі правила імпорту й експорту. Після завершення запису даних усіх шаблонів конвертації, виводиться повідомлення з інформацією, що усі дані були збережені. Схема алгоритму запису даних в файл *XML* формату зображена на рисунку 3.3.

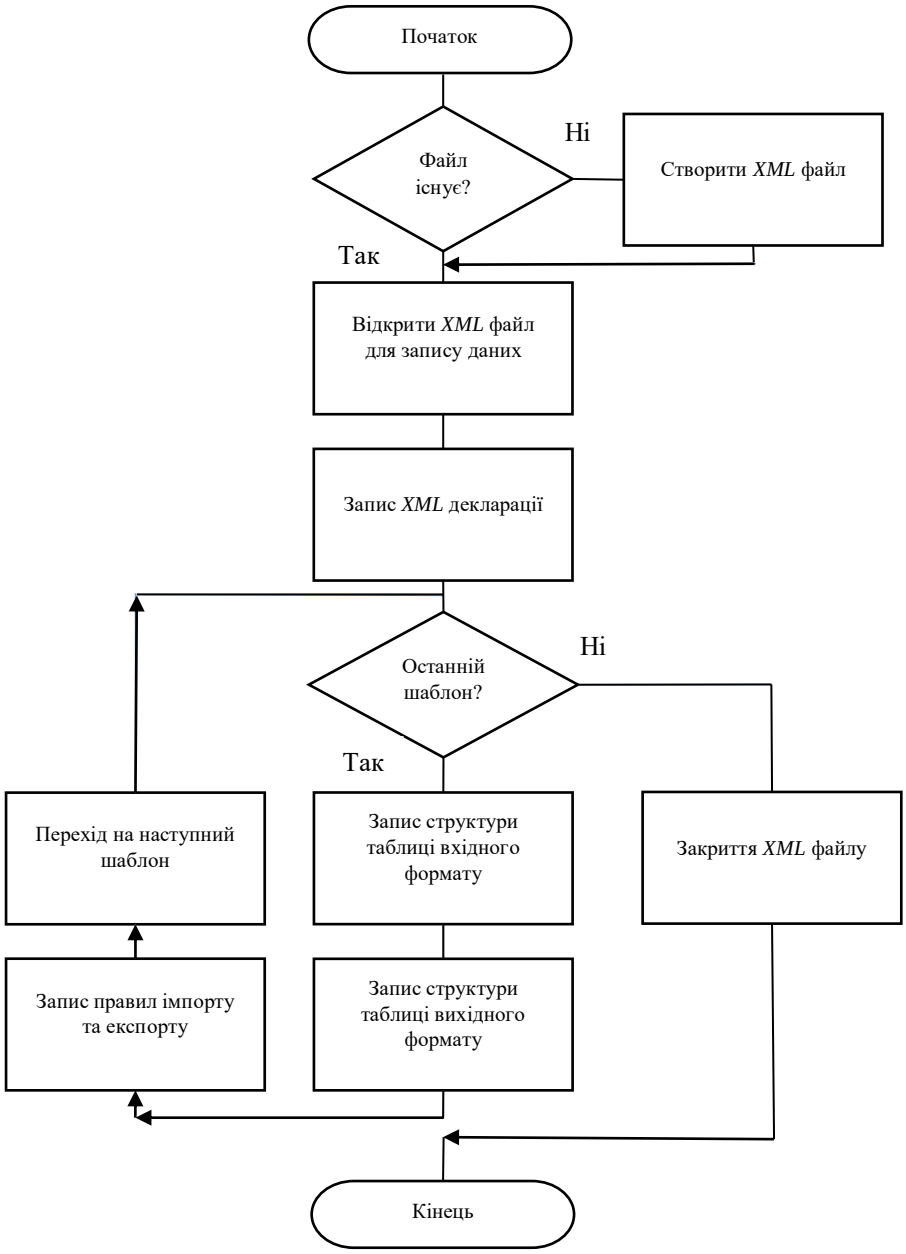

Рис. 3.3. Схема алгоритму запису даних в файл *XML* формату

Алгоритм читання даних з файлу *XML* формату починається з вибору користувачем файлу *XML* формату зі збереженими в ньому шаблонами правил конвертації. У випадку, коли зазначений файл не існує, файл пустий або у файлі немає шаблонів, що визначають правила конвертації даних між необхідними структурами таблиць, зчитування даних не відбувається. Основним циклом алгоритму є цикл проходження по шаблонам правил конвертації. Для кожного шаблону з файлу зчитуються структури таблиць вхідного та вихідного формату

та всі правила імпорту й експорту. Після завершення зчитування даних усіх шаблонів конвертації, програма перевіряє шаблони на правильність форматів. Схема алгоритму читання даних з файлу *XML* формату зображена на рисунку 3.4.

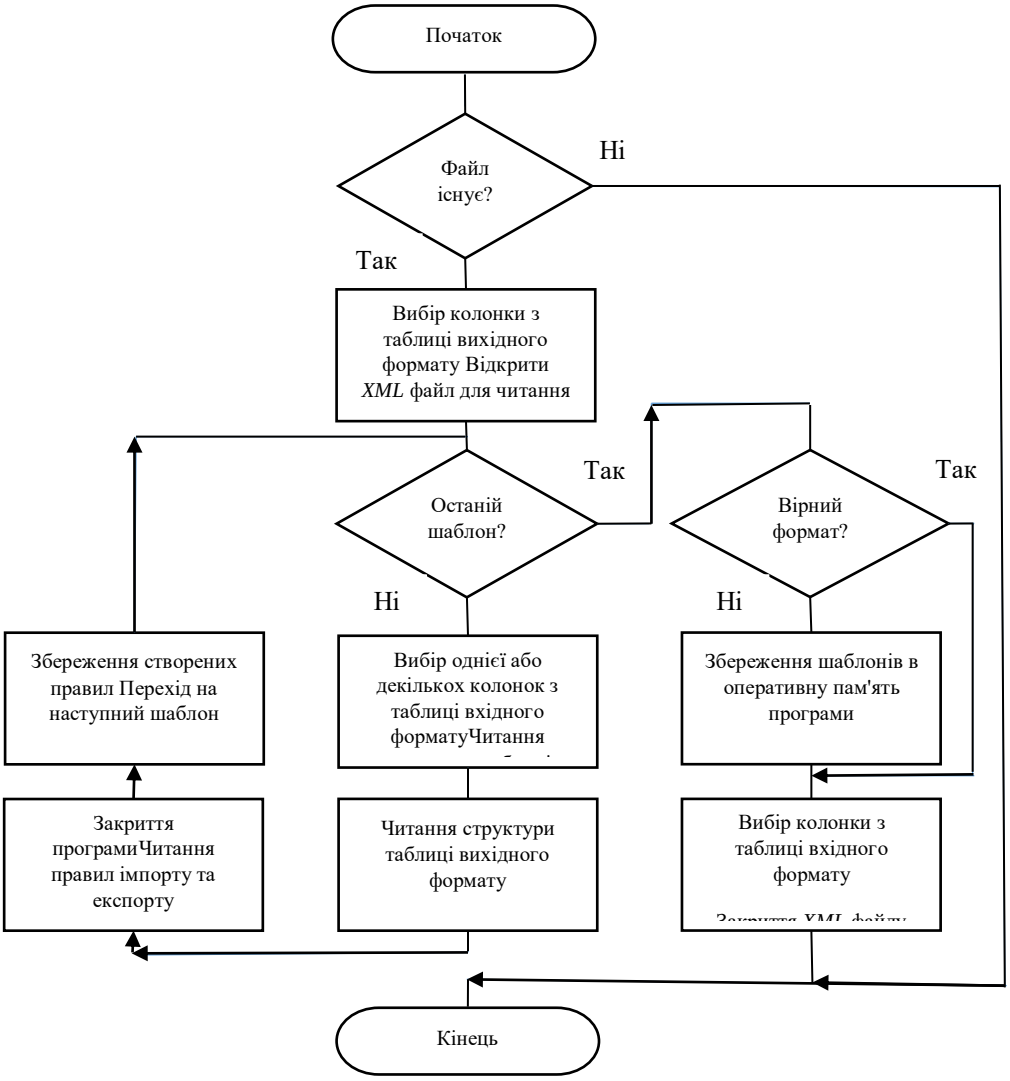

Рис. 3.4. Схема алгоритму читання даних з файлу *XML* формату

3.5. Проектування програми створення правил конвертації

Програмна компонента створення правил конвертації має надавати користувачеві можливість створювати нові, завантажувати, редагувати та

видаляти правила як імпорту, так і експорту даних. Компонента має запускатися після вибору користувачем об’єкту таблиці та вихідного формату даних.

До функцій компоненти можна віднести:

– створення правила імпорту через визначення користувачем колонки з таблиці вхідного формату та однієї або декількох колонок таблиці вихідного формату;

– створення правила експорту через визначення користувачем колонки з таблиці вихідного формату та однієї або декількох колонок таблиці вхідного формату;

– надання користувачеві можливості визначати правило конвертації як формулу;

– відображення створених або завантажених з бази знань правил конвертації;

– відображення попереджень у випадку невірно визначених правил чи виходу з програми без збереження відредагованих правил.

До формули правила конвертації даних між двома або більше колонками таблиць можуть входити оператори, назви колонок, цілі та дробові числа. Формули записуються в оперативну пам’ять програми в якості строкового типу даних.

Інтерфейс програмної компоненти має бути простим та інтуїтивно зрозумілим користувачеві. Послідовність дій, за якою користувач повинен створювати правила конвертації даних, може бути наступною:

– розглянути структури таблиць вхідного та вихідного форматів;

– розпочати створення правила імпорту або експорту;

– для правила імпорту вибрати колонку із структурної схеми таблиці вхідного формату та одну або декілька колонок із структурної схеми таблиці вихідного формату;

– для правила експорту вибрати колонку із структурної схеми таблиці вихідного формату та одну або декілька колонок із структурної схеми таблиці вхідного формату;

– по потребі доповнити автоматично створену формулу;

– створити необхідну кількість правил та зберегти їх в базу знань;

– розпочати конвертацію даних.

Усі правила мають зберігатися в оперативній пам’яті програми в окремому об’єкті даних. Цей об’єкт даних має надавати достатню кількість функціональних можливостей для зручного власного збереження та редагування. Також об’єкт даних має виводити попередження в разі спроби збереження правил з невірним форматом формул [18].

Правило конвертації має визначати послідовність дій, яка повинна бути виконана для вірного транслювання інформації з вхідної колонки у вихідну. Інформації двох або більше колонок (полів даних) повинна зберігатися в одному типі даних. До сукупностей типів даних, між якими не може відбуватися транслювання даних належать: між строковим та чисельними типами даних, із дробового в цілочисельний тип даних, із дробового в булевий тип даних, між датою та усіма іншими типами даних та із цілочисельного в булевий тип даних.

При спробі транслювати інформацію між однією з наведених сукупностей типів даних може виникнути як деформація структурного змісту, так і повна втрата цих даних. В наведеному прикладі структурної схеми таблиць серед усіх типів даних лише поля з назвами форм є строковими типами даних. Оскільки усі інші поля мають дробовий тип даних, то зміна структурного змісту є малоймовірною. Враховуючи це можливо виділити два формати формул конвертації: між текстовими типами даних та між чисельними.

output_text = imput_text1 + input_text2 + … + input_textN; – формат формули правила транслювання інформації між двома строковими типами даних, де права частина формули відображає сполучення даних колонок таблиці вхідного формату, що будуть записані в назву колонки з лівої частини формули.

output_number = imput_value1 + input_value2 + … + input_valueN + input_number1 + imput_number2 + … + imput_numberN; – формат формули правила транслювання інформації між двома строковими типами даних, де змінні *input_value1, input_value2* та *input_valueN* є значеннями полів таблиці вхідного формату, змінні *input_number1, input_ number 2* та *input_ number N* є числами, дописаними в формулу користувачем, а *output_number* є полем, в яке отримане значення буде записане.

3.6. Висновки до розділу

В даному розділі детально розглянуто поняття програмної системи конвертації даних при інформаційному моделюванні споруд, перераховані та детально описані основні операції, які має виконувати розроблена система конвертації даних, та перераховані основні інструменти, що були використані під час проектування та розробки програми. Основним інструментом є середовище розробки *Microsoft Visual Studio* 2019.

Таким чином програмна система – різновид програмного забезпечення, що використовується для розв’язання всього спектру задач, що можуть виникнути при вирішенні поставленої проблеми. Розрізняють прикладні програмні системи та інструментальні програмні системи (комп’ютерні програми, призначені для проектування, розробки та адміністрування).

До спроектованих операцій програмної системи конвертації даних належать: підключення до баз даних формату **.mdf* засобами системи керування базами даних *Microsoft SQL Server*, зчитування та відображення вибраних даних у вигляді таблиць даних, створення правил імпорту та експорту даних, зчитування та збереження створених правил з бази знань правил конвертації формату **.xml*, виконання конвертації даних та їх відображення в інтерфейсі програмної системи.

Були спроектовані та описані п’ять програмних компонент: компонента відображення даних, компонента реалізації *SQL* запитів, компонента роботи з *XML* форматом та компонента створення правил конвертації.

Компонента відображення даних, що призначена для відображення вибраних з бази даних таблиць, їх колонок, строк та структурних схем. Були перераховані спроектовані функції компоненти, наведений приклад структурної схеми таблиці двутаврів та описаний узагальнений алгоритм конвертації даних.

Компонента реалізації *SQL* запитів призначена для створення підключення до бази даних та виконання запитів, написаних на мові *Transact-SQL*. Були перераховані спроектовані функції компоненти, детально описане поняття транзакції та детально описані основні інструкції та оператори мови запитів *Transact-SQL*.

Компонента роботи з файлами *XML* формату призначена для реалізації функцій та методів читання, редагування та створення файлів баз знань правил конвертації. Були перераховані спроектовані функції компоненти, описане поняття елементу *XML*-декларації та спроектовані алгоритми запису та читання даних з файлу формату **.xml*.

Компонента створення правил конвертації призначена для надавання користувачеві можливості створювати нові, завантажувати, редагувати та видаляти правила як імпорту, так і експорту даних. Були перераховані спроектовані функції компоненти, описана послідовність дій користувача при роботі з компонентою та спроектовані два формати формул конвертації даних.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ КОНВЕРТАЦІЇ ДАНИХ ПРИ ІНФОРМАЦІЙНОМУ МОДЕЛЮВАННІ СПОРУД

4.1. Розробка інтерфейсу користувача

Програмну систему конвертації даних було створено в середовищі розробки *Visual Studio* 2019, в якості шаблону побудови проектів обрано *Windows Forms Application C#*. Проект названо *SqlDatabaseTablesConverter*. До розроблених компонент програмної системи входять: програмна компонента відображення даних, програмна компонента створення *SQL*-запитів, програмна компонента роботи з файлами *XML* формату, програмна компонента створення правил конвертації та програмна компонента конвертування даних.

Для написання програми використано мови програмування *C++* та *C#*, та мова створення *SQL*-запитів *Transact-SQL*. *C++* застосовано для написання модулів конвертації, передачі *SQL* запитів та роботи з *XML* форматом, оскільки від швидкості виконання коду зазначених модулів залежить швидкість роботи усієї програми.

На мові програмування *C#* здебільшого реалізовані програмні складові інтерфейсу програмної системи, оскільки мова надає зручні та ефективні інструменти розробки візуальних складових програмного інтерфейсу. До складових інтерфейсу входять: основна форма програми, різноманітні компоненти інтерфейсу, об’єкти контролів. Реалізовано відповідні оброблювачі подій, модуль відображення даних та модуль інтерфейсу користувача.

У ході розробки програми також був використаний програмна система *Microsoft SQL Server Management Studio* (*SSMS*) для перегляду, отримання та управління об’єктами сервера, створення тестових баз даних та перевірки результатів виконання *SQL* запитів.

Програмну систему конвертації даних було розроблено для таких цілей:

– перегляд існуючих таблиць у вибраній базі даних, перегляд структурної схеми вибраної таблиці;

– перегляд вмісту даних вибраної таблиці, створення та редагування файлів баз знань правил конвертації даних *XML* формату;

– створення, редагування та видалення правил імпорту;

– створення, редагування та видалення правил експорту;

– перегляд створених правил конвертації даних;

– збереження правил конвертації даних в оперативні пам’яті програми;

– читання правил конвертації даних з файлу бази знань та їх запис в оперативну пам’ять програми;

– запис правил конвертації даних в файл бази знань з оперативної пам’яті програми;

– виконання процесу конвертації даних з вхідного формату даних у вихідний та перегляд результатів конвертації.

Усі файли проекту розташовані в папці, що має повний шлях – “*D:\Диплом Сім’яЯ.СП-224М\SqlDatabaseTablesConverter”*.

До файлів проекту входять рішення-проекту *SqlDatabaseTablesConverter.sln*, файли основної форми програми *TablesConvertionForm.cs*, *TablesConvertionForm.Designer.cs*, *TablesConvertionForm.resx* та елементи: *Controls*, *Datas*, *SqlRelated*, *TextAnayzer* та XmlRelated.

Папка *Controls* призначена для зберігання файлів з кодом класів типу *System.Forms.Control*. Класи даного типу виконують відображення на формах елементів програмного інтерфейсу. Папка містить автоматично створені середовищем розробки файли: *ConvertionControl.Designer.cs*, *ConvertionControl.resx*, *TablesDispayingControl.Designer.cs* та *TablesDispayingControl.resx*. Також папка містить додаткові файли, створені в ході написання програми: *ConvertionControl.cs* та *TablesDispayingControl.cs*.

Файли *ConvertionControl.Designer.cs* та *TablesDispayingControl.Designer.cs* – автоматично згенеровані середовищем розробки, в які записаний код створення візуального представлення об'єктів класу *System.Forms.Control*.

Файли ресурсів *ConvertionControl.resx* та *TablesDispayingControl.resx* – автоматично згенеровані середовищем розробки, в яких зберігаються дані ресурсів, що використовуються в класах *ConvertionControl* та *TablesDispayingControl*.

Файли *ConvertionControl.cs* та *TablesDispayingControl.cs* – створені в ході написання програми. Містять код класів *ConvertionControl* та *TablesDispayingControl*.

Папка *Datas* призначена для зберігання файлів з кодом класів, що є одиницями збереження даних в оперативній пам'яті. Має створені в ході написання програми файли *TemplateData.cs* та *TransferRule.cs*. з кодом класів типу *System.Forms.Control*. Файл *TemplateData* містить код, конструктор, методи та параметри класу *TemplateData*, що є аналогом шаблонів правил конвертації даних. Файл *TransferRule.cs* призначений для зберігання коду правил конвертації.

Папка *SqlRelated* призначена для зберігання усіх файлів, що входять до складу модуля створення *SQL*-запитів. Містить файл, створений в результаті розробки програми *MySqlConnection.cs*, з кодом класу *MySqlConnection*.

Папка *TextAnayzer* призначена для зберігання усіх файлів, що є складовими частинами модуля конвертації даних, та виконують функції аналізу текстових формул правил конвертації. Містить файл, створений в результаті розробки програми *TextAnalizer.cs*, з кодом класу *TextAnalizer*.

Папка *XmlRelated* призначена для збереження файлів, що входять до складу модуля роботи з файлами *XML* формату. Містить файли *XmlAccessor.cs*, *XmlTemplatesReader.cs* та *XmlTemplatesWriter.cs*, створені в ході розробки програми.

Файл *SqlDatabaseTablesConverter.sln* – файл рішення проекту, автоматично згенерований середовищем розробки при створенні проекту програми. Файл містить текстову інформацію, яку середовище використовує для пошуку та загрузки параметрів значень імен для збереження даних та *VSPackages* проекту. Коли користувач відкриває рішення, середовище попередньо циклічно переглядає інформацію проекту та наступних рішень в файлі **.sln*, щоб завантажити рішення, проекти та будь-яку необхідну інформацію.

Файл *TablesConvertionForm.cs* – містить текстовий код основної форми програми класу *TablesConvertionForm*. Клас файлу використаний для створення точки входу в програму конвертації даних.

Процес конвертації даних при інформаційному моделюванні споруд складається з етапів:

- вибору даних для конвертації;
- створення або редагування правил конвертації;
- конвертації даних та відображення результатів конвертації.

Оскільки етап конвертації даних не має візуального представлення, то інтерфейс програми реалізований між трьома вікнами: вікно вибору даних для конвертації, вікно створення правил конвертації та вікно відображення результатів конвертації.

Вікно відображення даних призначене для виведення користувачеві списку таблиць вибраної бази даних. Програма створює вікно одразу після запуску програми та вибору файлу бази даних. У лівій частині вікна відображається список таблиць. У правій частині у вигляді таблиці відображається структурна схема вибраної таблиці та її дані. Також вікно має кнопку “Створити правило конвертації”, що виконує перехід на наступне вікно програми. Форму вікна відображення таблиць бази даних зображено на рисунку 4.1.

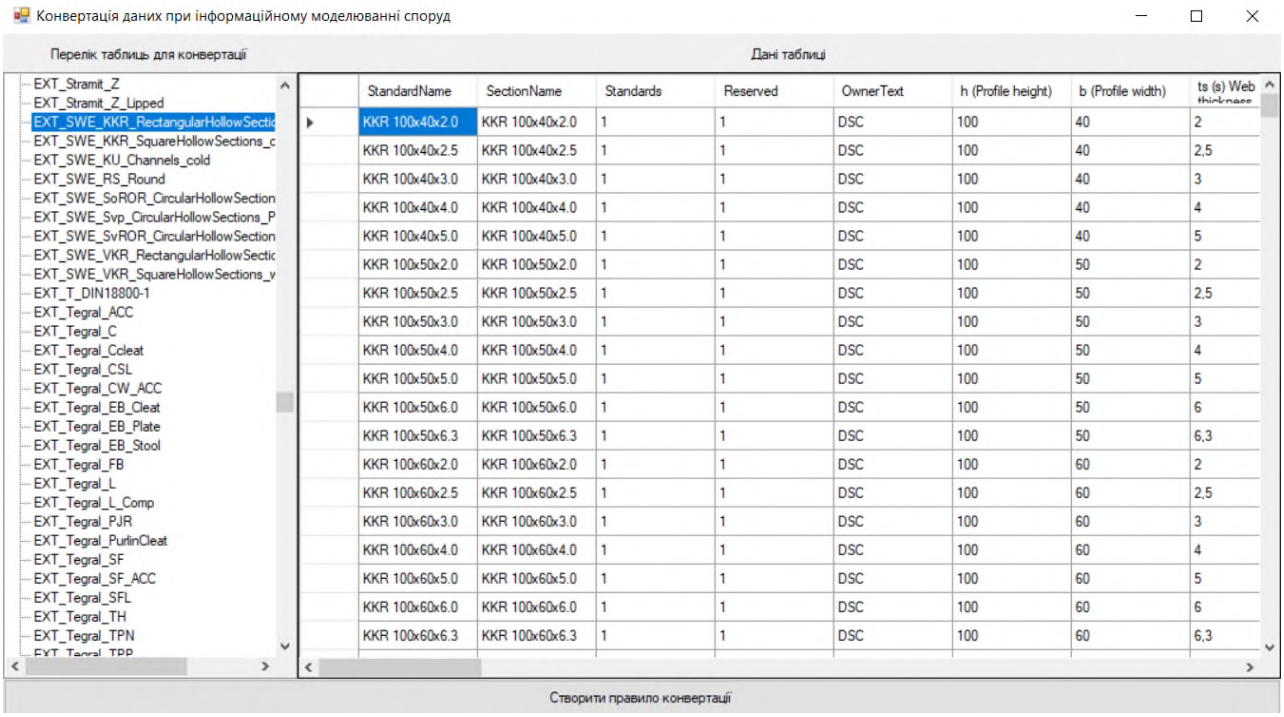

Рис. 4.1. Зображення форми вікна вибору даних для конвертації

Вікно створення правил конвертації виводить користувачеві структурні схеми таблиць вхідного та вихідного форматів, дає можливість завантажити, створити, редагувати та зберегти правила імпорту та експорту. Інтерфейс вікна реалізований в межах двох таблиць зі структурними схемами даних, дерева відображення правил конвертації та восьми кнопок, які виконують всі необхідні дії для зручного створення правил конвертації. Структурні схеми даних відображаються в лівій та правій частині вікна, в той час як кнопки розташовані по центру. Елемент, що відображає дерево правил конвертації розміщене між кнопками та правою структурною схемою даних.

Користувачу програми дається можливість виконувати дії, що реалізовані в таких кнопках інтерфейсу: “Створити правило експорту”, “Створити правило імпорту”, “Зберегти правила”, “Конвертувати дані”, “Повернутися до вибору таблиць” та “Вийти”. Форму вікна створення правил конвертації зображено на рисунку 4.2.

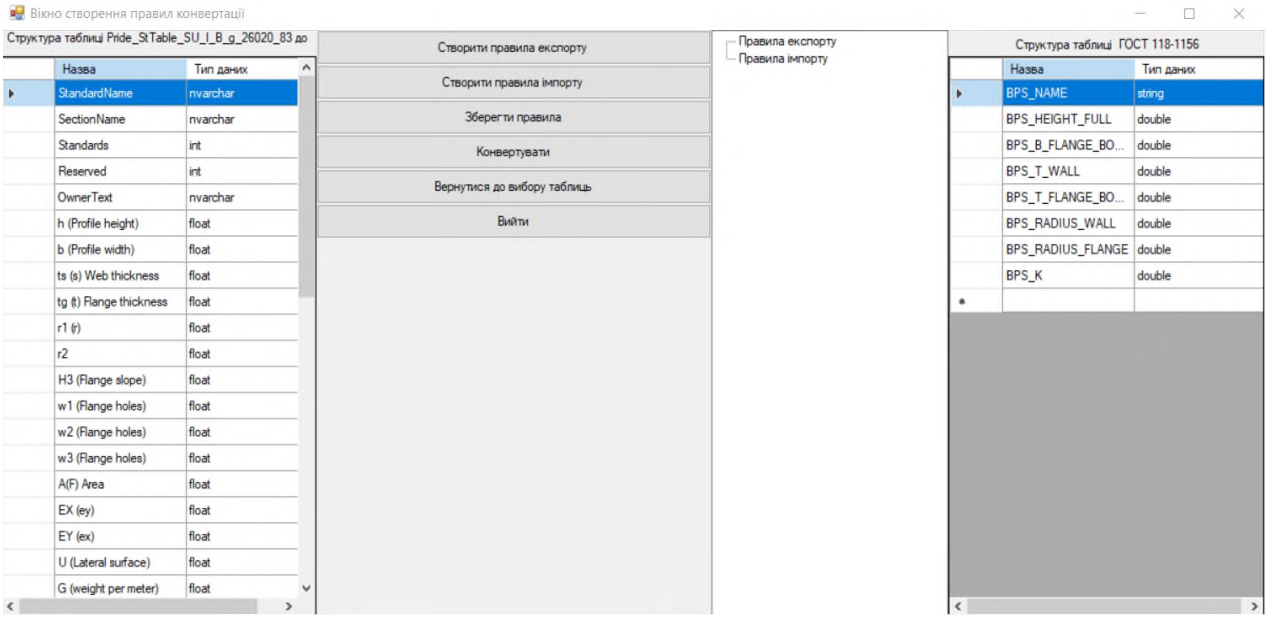

Рис. 4.2. Зображення форми вікна створення правил конвертації

Вікно відображення результатів конвертації виводить користувачеві структурну схему таблиці вихідного формату та дані, що були трансльовані. Інтерфейс вікна реалізовано в межах таблиці з результатами процесу конвертації та двома кнопками. Таблиця займає верхню та центральну частини вікна, а кнопки “Зберегти таблицю після конвертації ” та “Вийти” – нижню.

Користувачу програми дається можливість переглянути результати конвертації, зберегти їх в окремий файл, закрити програму або повернутися до вікна створення та редагування правил конвертації. Форму вікна відображення результатів конвертації зображено на рисунку 4.3.

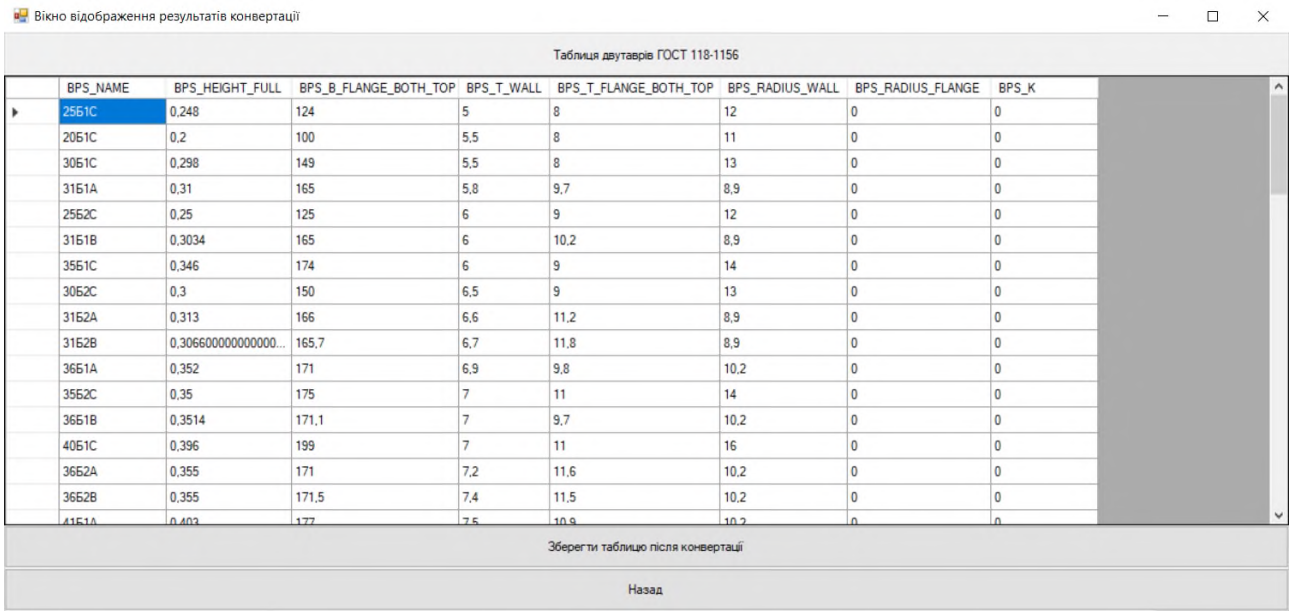

Рис. 4.3. Зображення форми вікна відображення результатів конвертації

Інтерфейс користувача програмної системи конвертації даних при інформаційному моделюванні споруд реалізовано за допомогою основної форми програми *TablesConversionForm* та елементів *ConversionControl*, *TablesDisplaiingControl* та *ResaultDisplaingControl*. Усі елементи керування простору імен *System.Windows.Forms*, що застосовуються для створення інтерфейсу користувача, оголошені в тілі перерахованих компонент. При запуску програми на основну форму програми додається компонента *TablesDisplaingControl*. При переході між компонентами типу *Control*, елементи, оголошені в їх тілі, додаються на основну форму програми. Переключення між різними вікнами модулів програмної системи відбувається через зміну вибраної компоненти типу *System.Forms.Control*.

Елемент *TablesConversionForm* – основна форма програми, що містить в якості полів об’єкти *mySqlConnection* класу *MySqlConnection* та *baseControl* класу *IControl*. Перехід від одного вікна програми до іншого реалізований через перевизначення об’єкту *baseControl*. Клас форми містить конструктор та такі методи: *TablesConvertionForm_FormClosing* та *NextControl*.

Метод *NextControl*() – функція, що виконує перехід програми з поточного вікна, на отримане в якості параметру типу *System.Forms.Control*. Метод видаляє

частину даних, виділених для компоненти поточного об’єкту вікна, та виділяє звільнену пам’ять під компоненти об’єкту вікна, що буде встановлене наступним.

Метод *TablesConvertionForm_FormClosing*() – оброблювач подій, що спрацьовує при закритті програми, здебільшого після виконання методу *Close*(). Звільняє виділену під процес оперативну пам’ять програми.

Елемент *TablesDispayingControl* – клас, в якому реалізовано інтерфейс вікна вибору даних для конвертації. Клас має конструктор, що отримує в якості параметру об’єкт основної форми програми та наступні компоненти: *tablesTreeView, tablesLabel, columnsLabel, convertTableButton, columnsDataGridView* та *mainTableLayoutPanel*.

Компонента *tablesLabel* – об’єкт системного класу *Label*, що відображає напис “Список таблиць для конвертації”.

Компонента *columnsLabel* – об’єкт системного класу *Label*, що відображає напис “Данні таблиці”.

Компонента *tablesTreeView* – об’єкт системного класу *TreeView*, що відображає список об’єктів таблиць вибраної бази даних. Для даної компоненти було створено два оброблювачі подій: *tablesTreeView_NodeMouseClick* та *tablesTreeView_NodeMouseDoubleClick.* Оброблювач подій *tablesTreeView_NodeMouseClick* спрацьовує при виборі таблиці зі списку, та виконує заповнення компоненти *columnsDataGridView* даними цієї таблиці. Оброблювач подій *tablesTreeView_NodeMouseClick* спрацьовує при подвійному натисканні на ім’я таблиці, та виконує перехід на наступне вікно створення правил конвертації.

Компонента *convertTableButton* – кнопка, що виконує перехід на наступне вікно створення правил конвертації. Поле *Text* об’єкту має значення “Створити правило конвертації”. Для компоненти створено оброблювач подій *convertTableButton_Click.*

Компонента *columnsDataGridView* – об’єкт класу *DataGridView*, що призначений для виведення структурної схеми та даних вибраної таблиці.

Компонента *mainTableLayoutPanel* – об’єкт класу *TableLayoutPanel*, що виконує роль контейнера, та містить всі вище перераховані компоненти вікна відображення таблиць вибраної бази даних.

Також клас має об’єкт *mySqlConnection* створеного класу *MySqlConnection* та метод *GetTablesColumns*. Об’єкт *mySqlConnection* реалізує модуль створення *SQL*-запитів. Метод *GetTablesColumns* визначає перелік вихідних форматів даних.

Елемент *ConversionControl* – клас, в якому реалізовано інтерфейс вікна створення правил конвертації. Клас має конструктор, що отримує в якості параметру об’єкт основної форми програми, об’єкт структурної схеми таблиці вхідного формату, об’єкт структурної схеми таблиці вихідного формату, а також назви таблиць. Також клас має компоненти, що реалізують інтерфейс вікна, основні серед яких: *ExportRuleButton, ImportRuleButton, saveRulesButton, convertDatasButton, importNextButton1, displayRulesButton, ExitButton returnToPrevoiusControlButton, rulesTreeView, inputFormat_dataGridView, outputFormat_dataGridView, ruleFormulaRichTextBox* та *ConversionTableLayoutPanel*.

Компонента *ExportRuleButton* – кнопка, що розпочинає створення правила експорту. Поле *Text* об’єкту має значення “Створити правило експорту”. Для компоненти створено оброблювач подій *ExportButton_Click*.

Компонента *ImportRuleButton* – кнопка, що розпочинає створення правила імпорту. Поле *Text* об’єкту має значення “Створити правило імпорту”. Для компоненти створено оброблювач подій *ImportButton_Click*.

Компонента *saveRulesButton* – кнопка, що зберігає щойно створене правило конвертації в оперативну пам’ять процесу програми. Поле *Text* об’єкту має значення “Зберегти правило”. Для компоненти створено оброблювач подій *saveRulesButton_Click_1*.

Компонента *convertDatasButton* – кнопка, що розпочинає процес конвертації даних з вхідного формату у вихідний, після чого виконує перехід на наступне вікно відображення результатів конвертації. Поле *Text* об’єкту має

значення “Конвертувати”. Для компоненти створено оброблювач подій *convertDatasButton_Click*.

Компонента *importNextButton1* – кнопка, що продовжує процес створення правила імпорту, та автоматично генерує текстову формулу правила по вибраним користувачем колонкам таблиць (полям даних). Після створення формули виводить на екран компоненту *ruleFormulaRichTextBox*, та заповнює її дані текстовим представленням формули. Поле *Text* об’єкту має значення “Далі”. Для компоненти створено оброблювач подій *importNextButton1_Click*.

Компонента *displayRulesButton* – кнопка, що виводить створене правило конвертації в компоненті *rulesTreeView*. Поле *Text* об’єкту має значення “Відобразити правило”. Для компоненти створено оброблювач подій *displayRulesButton_Click*.

Компонента *ExitButton*– кнопка, що припиняє процес створення правил конвертації, та виконує закриття програми. У разі, якщо в оперативній пам’яті процесу наявні не збережені правила конвертації, то користувачу виводиться повідомлення з відповідним попередженням. Поле *Text* об’єкту має значення “Вийти”. Для компоненти створено оброблювач подій *ExitButton_Click*.

Компонента *returnToPrevoiusControlButton* – кнопка, що припиняє процес створення правил конвертації та виконує перехід програми на попереднє вікно вибору даних для конвертації. У разі, якщо в оперативній пам’яті процесу наявні не збережені правила конвертації, то користувачу виводиться повідомлення з відповідним попередженням. Поле *Text* об’єкту має значення “Повернутися до вибору таблиці”. Для компоненти створено оброблювач подій *returnToPrevoiusControlButton_Click*.

Компонента *rulesTreeView* – виконує відображення збережених в оперативній пам’яті процесу правил конвертації. У разі, якщо у файлі бази знань правил конвертації уже існує шаблон, що визначає правила конвертації для вибраних структурних схем таблиць вхідного та вихідного форматів, то компонента виводить ці правила.

Компонента *inputFormat_dataGridView* – таблиця, в якій відображається структурна схема таблиці вхідного формату. Також елемент використовується в якості інструмента вибору колонок для створення правил.

Компонента *outputFormat_dataGridView* – таблиця, в якій відображається структурна схема таблиці вихідного формату. Також елемент використовується в якості інструмента вибору колонок для створення правил.

Компонента *ruleFormulaRichTextBox* – елемент відображення текстових даних. Призначений для відображення текстового вигляду формул правил конвертації, та надає користувачу перед збереженням можливість їх редагування.

Компонента *ConversionTableLayoutPanel* – об'єкт класу *TableLayoutPanel*, що виконує роль контейнера, та містить всі вище перераховані компоненти вікна створення правил конвертації. Автоматизує відношення розмірів між компонентами при масштабуванні вікна програми.

Серед параметрів класу *ConversionControl* слід виділити наступні: *xmlAccessor*, *templateDatas* та *changesMade*. Параметр *xmlAccessor* – об'єкт класу *xmlAccessor*, що є частиною модуля роботи з файлами *XML* формату. Параметр *templateDatas* виконує збереження правил конвертації даних в оперативній пам'яті процесу програми у вигляді списку об'єктів класу *TemplateData*. Параметр булевого типу *changesMade* зберігає інформацію про наявність не збережених правил конвертації.

Також в класі реалізовані наступні методи: *FillDataGredViews*, *FillCurrentTemplateData*, *DisplayExportRule*, *DisplayImportRule*, *ConvertDatasButtonClickMethod*, *GetColumnValueOrDouble*, *GetTypeByName*. Більшість функцій, які не можуть бути реалізовані в тілі створених оброблювачів подій, були винесені у перераховані методи.

Метод *FillDataGredViews* заповнює компоненти *inputFormat_dataGridView* та *outputFormat_dataGridView* структурними схемами таблиць вхідного та вихідного форматів відповідно. При хибності одного із форматів структурних схем виводить повідомлення з попередженням.

Метод *FillCurrentTemplateData* – виконує пошук необхідного шаблону правил користування з вибраної бази знань, правил конвертації засобами модуля роботи з *XML* форматом. У випадку коли необхідний шаблон був знайдений у файлі бази знань, відображає його правила в компоненті *rulesTreeView*.

Метод *DisplayExportRule* – виконує відображення створеного або відредагованого правила експорту в компоненті *rulesTreeView*.

Метод *DisplayImportRule* – виконує відображення створеного або відредагованого правила імпорту в компоненті *rulesTreeView*.

Метод *ConvertDatasButtonClickMethod* – запускає виконання алгоритму конвертації даних та подальший перехід на наступне вікно відображення результатів конвертації. У випадку, коли в оперативній пам’яті процесу програми не визначено жодного правила конвертації, виводить повідомлення з відповідним попередженням.

Метод *GetColumnValueOrDouble* – отримує значення колонки (поля даних таблиці), та по можливості приводить його до дробового вигляду.

Метод *GetTypeByName* – повертає тип одиниці даних, збереженої в оперативній пам’яті процесу.

Елемент *ResaultDisplaingControl* – клас, в якому реалізовано інтерфейс вікна відображення результату конвертації даних. Клас має конструктор, що отримує в якості параметру об’єкт основної форми програми та наступні компоненти: *saveConvertedTableButton*, *convertedTableDisplaingTableLayoutPanel*, *returnToFormulaPabelButton*, *createdTableDataGridView* та *createdTableNameLabel*.

Компонента *saveConvertedTableButton* – кнопка, що виконує збереження створених в результаті процесу конвертації даних у файлі вихідного формату. Поле *Text* об’єкту має значення “Зберегти таблицю після конвертації”. Для компоненти створено оброблювач подій *saveConvertedTableButton_Click*.

Компонента *returnToFormulaPanelButton* – кнопка, що виконує повернення до попереднього вікна створення правил конвертації. Поле *Text*

об’єкту має значення “Назад”. Для компоненти створено оброблювач подій *returnToFormulaPabelButton_Click*.

Компонента *createdTableNameLabel* – об’єкт класу *Label*, що виконує відображення назви таблиці вихідного формату.

Компонента *convertedTableDisplaingTableLayoutPanel* – об’єкт класу *TableLayoutPanel*, що виконує роль контейнера, та містить всі вище перераховані компоненти вікна відображення правил конвертації. Автоматизує відношення розмірів між компонентами при масштабуванні вікна програми.

При переході на вікно відображення результатів конвертації користувач може зберегти дані у файл вихідного формату або вернутися в попереднє вікно створення правил конвертації для редагування декількох правил конвертації.

4.2. Розробка програми відображення даних

Програмна компонента відображення даних спрощує інтерфейс програмної системи конвертації даних через відображення впорядкованих та інтуїтивно зрозумілих структур даних. Компонента відображає структурні схеми таблиць, дані таблиць, список об’єктів вибраної бази даних, текстову вигляд формул правил конвертації та дерево правил конвертації.

Компоненту реалізовано у межах елементів *ConversionControl*, *TablesDisplaiingControl* та *ResaultDisplaingControl*. А точніше тих складових компонент, що відповідають за відображення даних. До таких компонент входять об’єкти класів:

– *DataGridView* – системний клас, що відображає таблиці реляційної бази даних. Дозволяє організувати дані у вигляді таблиці, що містить колонки, строки та комірки. Дані можуть бути отриманими із бази даних, колекції, внутрішніх змінних або масивів;

– *TreeView* – системний клас, що відображає дані у вигляді деревовидного списку. Є сукупністю пов’язаних відношеннями структурних

піктограм в ієрархічному дереві. Зазвичай використовується для перегляду структури каталогів (папок) та інших подібних відношень;

– *ListBox* – системний клас, що відображає дані у вигляді списку. Ключовою властивістю елементу є компоненти *Items*, що зберігають набір всіх елементів списку;

– *ReachTextBox* – системний клас, що відображає текстову інформацію, в тому числі спеціальні символи текстової розмітки.

На вікні вибору даних для конвертації розташовані компоненти *tablesTreeView* та *columnsDataGridView*. Вікно створення правил конвертації містить компоненти *inputFormat_dataGridView*, *outputFormat_dataGridView*, *ruleFormulaRichTextBox* та *rulesTreeView*. Вікно відображення результатів виконання процесу конвертації даних містить компоненту *createdTableDataGridView*.

Для відображення структурних схем даних таблиць використовуються наступні правила:

– таблиця містить дві колонки;

– перша колонка називається “Назви”, та містить назви полів даних;

– друга колонка називається “Типи даних”, та містить назви типів даних в яких зберігаються дані полів.

Відображення правил конвертування у компоненті *rulesTreeView* типу *TreeView* реалізовано таким чином, що правила імпорту та експорту є підмножиною окремих гілок дерева. Гілка *ExportRules* містить перелік усіх збережених правил експорту, в той час як *ExportRules* – імпорту. Правила конвертації є їх відгалуженнями, та містять окреме відгалуження формули правила. Компоненту відображення дерева правил конвертації зображено на рисунку 4.4.

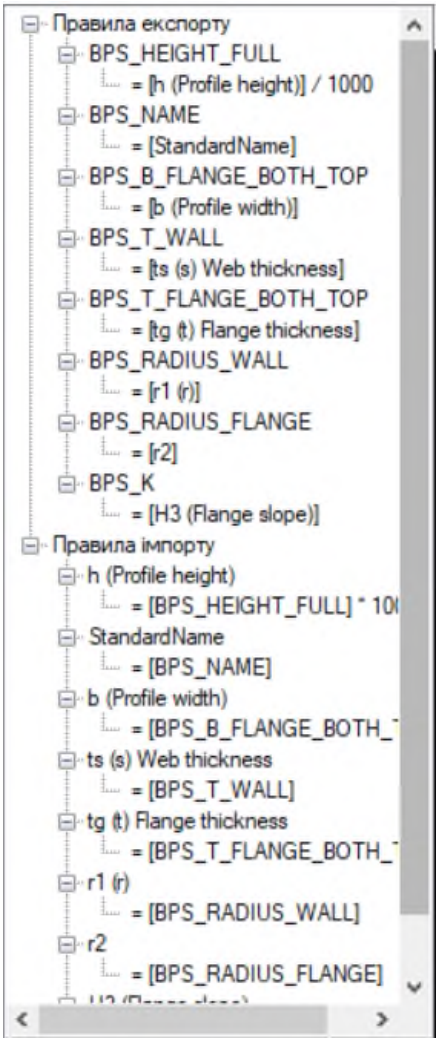

Рис. 4.4. Зображення компоненти відображення дерева правил конвертації

4.3. Розробка програми роботи з файлами *XML* формату

Програмну компоненту роботи з файлами *XML* формату створено для спрощення процесів читання та запису даних в файл бази знань правил конвертації. Компонента роботи з файлами *XML* формату створює, відкриває, розшифровує та редагує файли формату **.xml*. Основна функція модуля полягає в читанні з файлу бази знань правил конвертації даних та їх групування у вигляді шаблонів конвертації.

При розробці компоненти було створено клас *XmlAccessor*. Модуль реалізований у межах класу *XmlAccessor* з конструктором, в який в якості

першого аргументу передається повний шлях до файлу бази знань правил конвертації.

Об’єкт класу *XmlAccessor* створюється після вибору користувачем програмної системи таблиці з вхідною структурою даних. При цьому на екран виводиться вікно процесу створення або відкриття файлу формату *.*xml*, в якому користувач указує повний шлях до файлу бази знань правил конвертації. Якщо вікно не поверне жодного шляху, або поверне шлях, за яким система нічого не виявить, то об’єкт класу запропонує створити новий файл формату *.*xml*, та збереже його у директорію, що містить точку входу в програмну систему.

Якщо на момент створення об’єкту класу *XmlAccessor* у файлі формату *.*xml* містяться дані хоча б одного шаблону правил конвертації, то модуль завантажить правила в оперативну пам’ять процесу програми, та приведе їх до вигляду, зручного для редагування модулем створення правил конвертації.

При розробці класу *XmlAccessor* були використані наступні системні бібліотеки: *System.Xml*, *System.XPath* та *System.IO*.

Системна бібліотека *System.Xml* надає список класів для роботи з файлами *XML* формату, основними серед яких є *XmlFileStream*, *XmlTextReader*, *XmlException* та *XmlTextWriter*.

Клас *XmlFileStream* виконує підключення до файлу *.*xml* формату, та отримує в якості вхідного параметру об’єкт класу *FileStream*.

Клас *XmlTextReader* отримує в якості вхідного параметру об’єкт класу *XmlFileStream* та надає користувачу низку інструментів та методів для якісного виконання процесу читання даних. До методів, які були використані при розробці програмної системи входять: *Open*, *Read*, *Close*, *MoceNextToAttribute* та *MoveNextToSibling*.

Клас *XmlTextWriter* отримує в якості вхідного параметру об’єкт класу *XmlFileStream*, та надає користувачу низку інструментів та методів для якісного виконання процесу запису даних у файл. До методів, які були використані при розробці програмної системи входять: *WriteStartDocument*, *WriteStartElement*, *WriteString*, *WriteEndElement* та *WriteEndDocument*.

Клас *XmlException* використовується для попередження помилок, що виникають при роботі з файлами *XML* формату, та має список об'єктів типу *Exception*.

Системна бібліотека *System.Xml.XPath* надає список інструментів для знаходження зручного знаходження шляху до файлів формату **.xml*. При підключенні бібліотеки програміст може використовувати об'єкти наступних системних класів: *IXPathNavigable, IXPathException* та *IXPathItem*.

Клас *IXPathNavigable* виконує пошук та збереження повного шляху до файлів формату **.xml* та має параметр *Filter* зі значенням фільтра пошуку файлів конкретного формату "*XML Document* (**.xml*)|**.xml*|*All files* (*.*)|*.*".

Клас *IXPathException* використовується для попередження помилок, що виникають при роботі з повним шляхом до файлів *XML* формату, та має список об'єктів типу *Exception*. Клас *IXPathItem* – системний клас контейнер, у вигляді якого представлені усі об'єкти, необхідні для роботи з об'єктами класу *IXPathNavigable*.

В класі *XmlAccessor* реалізовано такі основні методи: *FindFile, WrightTemplates* та *ReadTemplates*. Метод *FindFile* класу *XmlAccessor* використовується у випадку, коли користувачем не визначено жодного файлу бази знань форматів як основного. Метод створює об'єкт *openFileDialog1* системного класу *OpenFileDialog* та виводить на екран вікно створення та відкриття файлу правил конвертації. Також метод для зручності фільтрує усі файли, в першу чергу шукаючи файли необхідного формату. Після того як користувач обирає необхідний файл та нажимає кнопку *OK*, його повний шлях передається в об'єкт *XmlAccessor*.

Метод *WrightTemplates* призначений для запису всіх наявних в оперативній пам'яті шаблонів правил користування в файл бази знань формату **.xml*. Спершу, створюється об'єкт *xmlTextWrighter* системного класу *XmlTextWrighter*, в який передається повний шлях до файлу бази знань, та який реалізовує можливість запису. Після виконання алгоритму запису даних шаблонів з оперативної пам'яті програмної системи в файл бази знань правил,

процес запису даних припиняється, а метод повертає інформацію про успішність його виконання.

Метод *ReadTemplates* призначений для читання в оперативну пам'ять програмної системи всіх наявних в файлі бази знань даних про шаблони конвертації. Спершу, створюється об'єкт *xmlTextReader* системного класу *XmlTextReader*, в який передається повний шлях до файлу бази знань, та який реалізовує можливість читання. Після виконання алгоритму читання даних шаблонів з файлу бази знань правил конвертації в оперативну пам'ять програми, метод повертає інформацію про успішність його виконання.

Для ефективного використання системних ресурсів при зберіганні даних правил конвертації в базах знань формату **.xml* розроблено розмітку *XML* формату, в яку входять вісім типів елементів та чотири параметри. Розмітка формату виглядає наступним чином:

<?*xml version*="1.0" *encoding*="*windows*-1251"> – *XML*-декларація;

< *Database* > – база знань;

< *Template* > – шаблон правил;

< *Schema*1 > – структурна схема таблиці вхідного формату;

< *Column* > – колонка таблиці вхідного формату;

< *Name* > "Назва колонки з таблиці вхідного формату" </ *Name* >

< *Type* > "Назва типу даних з таблиці вхідного формату" </ *Type* >

</ *Column* >

…

</ *Schema*1 >

< *Schema*2 > – структурна схема таблиці вихідного формату;

< *Column* > – колонка таблиці вихідного формату;

< *Name* > "Назва колонки з таблиці вихідного формату" </ *Name* >

< *Type* > "Назва типу даних з таблиці вихідного формату" </ *Type* >

</ *Column* >

…

</ *Schema*2 >

< *Transfer Rules*1 > – правила імпорту;

< *Conversation Rule* > – правило імпорту;

< *Column Name* > “Назва колонки у вхідному форматі, в яку дані будуть збережені” </ *Column Name* >

< *Rule* > “Правило за яким дані будуть записані” </ *Rule* >

< *Conversation Rule* >

…

</ *Transfer Rules1*>

< *Transfer Rules*2 > – правила експорту;

< *Conversation Rule* > – правило експорту;

< *Column Name* > “Назва колонки у вхідному форматі, в яку дані будуть збережені” </ *Column Name* >

< *Rule* > “Правило за яким дані будуть записані” </ *Rule* >

< *Conversation Rule* >

…

</ *Transfer Rules*2 >

</ *Template* >

…

</ *Database* >

де, *Database* – елемент, що містить шаблони конвертації даних між різними форматами; *Template* – елемент, що представляє собою шаблон конвертації даних між двома форматами даних; *Schema*1 – структурна схема таблиці вхідного формату; *Schema*2 – структурна схема таблиці вихідного формату; *Column* – елемент, що представляє собою колонку таблиці, та має два параметри: *Name* (назва колонки) та *Type* (тип даних в якому зберігається інформація у відповідній колонці); *Transfer Rules*1 – правила імпорту даних з вихідного формату у вхідний; *Transfer Rules*2 – правила експорту даних з вхідного формату у вихідний; *Conversation Rule* – елемент, що представляє собою правило конвертації, та має два параметри: *Column Name* (назва колонки в яку будуть записані дані) та *Rule* (правило за яким правила будуть записані).

Алгоритм запису даних в файл *XML* формату (рис. 3.3) використовується методом *WrightTemplates*, та реалізує проходження по усім шаблонам правил конвертації збереженим в оперативній пам’яті програми. Для реалізації алгоритму використовуються наступні команди:

Команда *xmlTextWriter.WriteStartDocument()*; виконує підключення до файлу бази знань формату **.xml*.

Команда *xmlTextWriter.WriteComment*("*LIRA* 10 *Templates data file*"); записує елемент *XML* декларації на початок файлу.

Команда *xmlTextWriter.WriteStartElement("Database")*; позначає початок запису регіону елемента бази даних.

Далі алгоритм реалізований у вигляді циклу, кожна ітерації якого є записом одного із шаблонів правил конвертації, а загальна кількість ітерацій визначається кількістю збережений в оперативній пам’яті процесу шаблонів правил конвертації.

Команда *xmlTextWriter.WriteStartElement("Template")*; позначає початок запису в файл регіону даних шаблону.

Команда *xmlTextWriter.WriteStartElement("Schema1")*; позначає початок запису в файл регіону даних структурної схеми таблиці вхідного формату.

Команда *xmlTextWriter.WriteStartElement("Schema2")*; позначає початок запису в файл регіону даних структурної схеми таблиці вихідного формату.

Команда *xmlTextWriter.WriteStartElement("Column")*; позначає початок запису в файл регіону даних колонки (поля даних таблиці).

Команда *xmlTextWriter.WriteStartElement("Name", "")*; позначає початок запису в файл параметру ім’я колонки.

Команда *xmlTextWriter.WriteString*(*keyValuePair.Key.ToString*()); записує значення параметру Name.

Команда *xmlTextWriter.WriteStartElement("Type", "")*; позначає початок запису в файл параметру тип даних колонки.

Команда *xmlTextWriter.WriteString*(*keyValuePair.Value.ToString*()); записує значення параметру *Type*.

Команда *xmlTextWriter.WriteStartElement("Conversation Rule")* позначає початок запису в файл регіону даних правила конвертації.

Команда *xmlTextWriter.WriteStartElement("ColumnName", "")*; позначає початок запису в файл параметру назви колонки.

Команда *xmlTextWriter.WriteString*(*transferRule.columnName*); записує значення параметру назви колонки.

Команда *xmlTextWriter.WriteStartElement("Rule", "")*; позначає початок запису в файл регіону даних формули конвертації.

Команда *xmlTextWriter.WriteString(transferRule.ruleFormula)*; записує значення параметру формули конвертації.

Команда *xmlTextWriter.WriteEndElement()*; закриває усі раніше оголошені регіони даних.

Команди *xmlTextWriter.WriteEndDocument*(); та *xmlTextWriter.Close*(); зберігає файл бази знань формату **.xml*.

Алгоритм читання даних з файлу *XML* формату (рис. 3.4) використовується методом *ReadTemplates* та реалізує проходження по усім шаблонам правил конвертації збереженим у файлі бази знань правил конвертації даних. Реалізація алгоритму включає наступні пункти, дії та правила:

– виконується читання об’єкту з файлу бази знань правил конвертації;

– якщо об’єкт є пустим, то метод завершує виконання алгоритму. Це свідчить про те, що у файлі не залишилося даних;

– виконується отримання значення поля *NodeType*;

– перевіряється умова, при якій значення поля *NodeType* дорівнює *XmlNodeType.Element*;

– отримується значення поля *Name*, у якому вказана назва поточного елементу;

– якщо поточним елементом є *Template*, то в оперативній пам’яті програми виділяється місце під новий об’єкт шаблону;

– якщо поточним типом елементу є будь-який інший тип елементів, визначений використаним форматом *XML* розмітки, то дані про тип елемента записуються в змінну *readingType* або *elementType*;

– перевіряється умова, при якій значення поля *NodeType* дорівнює *XmlNodeType.Text*;

– отримується значення поля *Value*, у якому збережені дані;

– якщо поточним елементом є параметр *Name* колонки, то його значення зчитується та додається в кінець списку *firstTableSchema.Names*.

– якщо поточним елементом є параметр *Type* колонки, то його значення зчитується та додається в кінець списку *firstTableSchema.Types*;

– якщо поточним елементом є параметр *ColumnName* колонки, то його значення зчитується та додається в кінець списку *firstSchemaTransferRules.ColumnNames*;

– якщо поточним елементом є параметр *Rule* колонки, то його значення зчитується та додається в кінець списку *firstSchemaTransferRules.Rules*.

Після виконання алгоритму читання шаблонів правил конвертації з файл бази знань формату **.xml* та його запису в оперативну пам’ять програмної системи, виконується метод *Close*() об'єкта *xmlTextReader* класу *XmlTextReader*. Це дозволяє очистити частину оперативної пам’яті, що використовувалась для підтримки доступу програмної компоненти до вмісту файлу.

4.4. Розробка програми реалізації *SQL* запитів

Програмну компоненту створення *SQL* запитів створено для реалізації можливості підключення до вибраної бази даних, створення запитів читання, редагування та збереження даних та виконання їх вибірки. Основна функція компоненти полягає в читанні з файлу бази даних формату **.mdf* таблиць з даними, їх структурних схем та даних.

При розробці компоненти було створено клас *MySqlConnection.* Компоненту реалізовано у межах класу *MySqlConnection* з конструктором, в який в якості першого аргументу передається повний шлях до вибраного файлу бази даних. Якщо шлях пустий або вказаного файлу не існує, то компонента відкриває діалогове вікно відкриття файлу бази даних, та реалізує можливість користувачу власноруч вибрати файл бази даних.

Об'єкт класу *MySqlConnection* створюється після вибору користувача програми файлу бази даних. Якщо на момент створення об'єкту класу *MySqlConnection* у файлі формату **.mdf* містяться дані хоча б однієї таблиці даних, то компонента завантажить список таблиць та виведе їх у вигляді списку в компоненті *tablesTreeView.*

При розробці класу *MySqlConnection* були використані наступні системні простори імен: *System.Data.Sql* та *System.Data.SqlClient*, оскільки разом вони надають повний набір інструментів для виконання поставлених перед модулем задач.

Простір імен *System.Data.Sql* надає для використання повний набір інструментів для роботи з реляційними базами даних, дозволяє створювати підключення до баз даних через створення об'єктів класу *SqlConnection* та виконувати команди, написані на мові *T-SQL.*

Простір імен *System.Data.SqlClient* надає можливість працювати з базами даних через *SQL Server* будь-якої версії. Відмінність полягає в тому, що перед підключенням до бази даних виконується реєстрація та подальший логін клієнта на встановленому *SQL* сервері.

При розробці класу *MySqlConnection* був створений параметр *sqlConnection* класу *SqlConnection* та наступні методи: *isOpened, getAllTablesNames, getColumns, GetDataTable* та *GetDataTableReader*.

Клас *SqlConnection* – об'єкт з'єднання з базою даних, що дозволяє клієнтському програмному забезпеченню взаємодіяти із програмним забезпечення сервера баз даних, незалежно від того, на якій машині він розміщений. Для надсилання команд та отримання відповідей потрібне

з’єднання, як правило у формі набору результатів. З’єднання створюється шляхом надання провайдеру рядка підключення, який є способом адресування конкретної бази даних, екземпляра серверу чи облікових даних для автентифікації користувача. Рядок включає назву серверу, назву бази даних, порядкового індексу клієнта та його паролю.

Метод *isOpened* показує чи відкрите підключення. Використовується перед створенням команд-запитів. Повертає значення *true* тоді, коли підключення бази даних відкрите, та *false* в протилежному випадку.

Метод *getAllTablesNames* призначений для отримання списку наявних у вибраній базі даних таблиць. Спершу метод перевіряє стан підключення до бази даних, потім створює команду з необхідним для виконання запиту текстом. Після створення об’єкта *dataTableReader* системного класу *DataTableReader* та його перевірки на наявність строк, метод формує список назв таблиць із результатів виконання команди.

Метод *getColumns* призначений для отримання структурної схеми об’єкту таблиці. Метод в якості вхідного параметру отримує строкове значення назви таблиці. Після перевірки з’єднання та створення команди, метод, створює об’єкт класу *DataTableReader* та формує об’єкт структурної схеми. Був розроблений наступний код команди:

```
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'tableName'
```

де, перший рядок команди визначає колонки, які входять до складу структурної схеми, наступний рядок визначає системну таблицю з якої будуть зчитані дані, а останній рядок визначає назву таблиці, структурну схему якої повинна повернути команда.

Метод *GetDataTable* призначений для отримання даних вказаної таблиці. Метод отримує в якості вхідного параметру назву таблиці. Після виконання запиту до бази даних, створюється об’єкт *sqlDataAdapter* системного класу

SqlAdapter, який заповнює даними об’єкт таблиці системного класу *DataSet*. Метод використовує наступний код *SQL*-запиту:

*Select * From* [‘*tableName*’]

де, *tableName* – змінна, в якій збережено назву таблиці, а оператор * – означає, що будуть повернуті дані усіх наявних в таблиці колонок.

Метод *GetDataTableReader* призначений для виконання отриманого в якості вхідного параметру об’єкта *SQL*-запиту, та повернення результату виконання у вигляді об’єкту класу *DataTableReader*. Спершу метод перевіряє з’єднання з базою даних, потім виконує код запиту та заповнює об’єкт *schemaTable* системного класу *DataTable* його результатом.

4.5. Розробка програми створення правил конвертації

Програмну компоненту створення правил конвертації створено для реалізації можливості завантаження, створення, редагування, збереження та видалення правил конвертації з файлу бази знань формату *.*xml* та оперативної пам’яті програми. Основна функція компоненти полягає в наданні користувачу програми зручних інструментів для створення правил конвертації, та надання інструментів для адміністрування баз знань правил конвертації.

Компоненту реалізовано таким чином, щоб по можливості зменшити кількість кроків, які повинен виконати користувач програми для створення одного правила. При використанні модуля передбачений наступний порядок користувача дій: вибір операції експорту або імпорту, вибір колонки з структурної схеми таблиці вхідного формату, вибір колонки з структурної схеми таблиці вихідного формату, редагування формули правила, збереження створеного правила в оперативну пам’ять програми та вивід на екран, створення необхідної кількості правил для повного та правильного транслювання даних між таблицями, збереження створених правил в файл бази знань правил конвертації та виконання процесу конвертації.

Схема алгоритму створення правил конвертації зображено на рисунку 4.5.

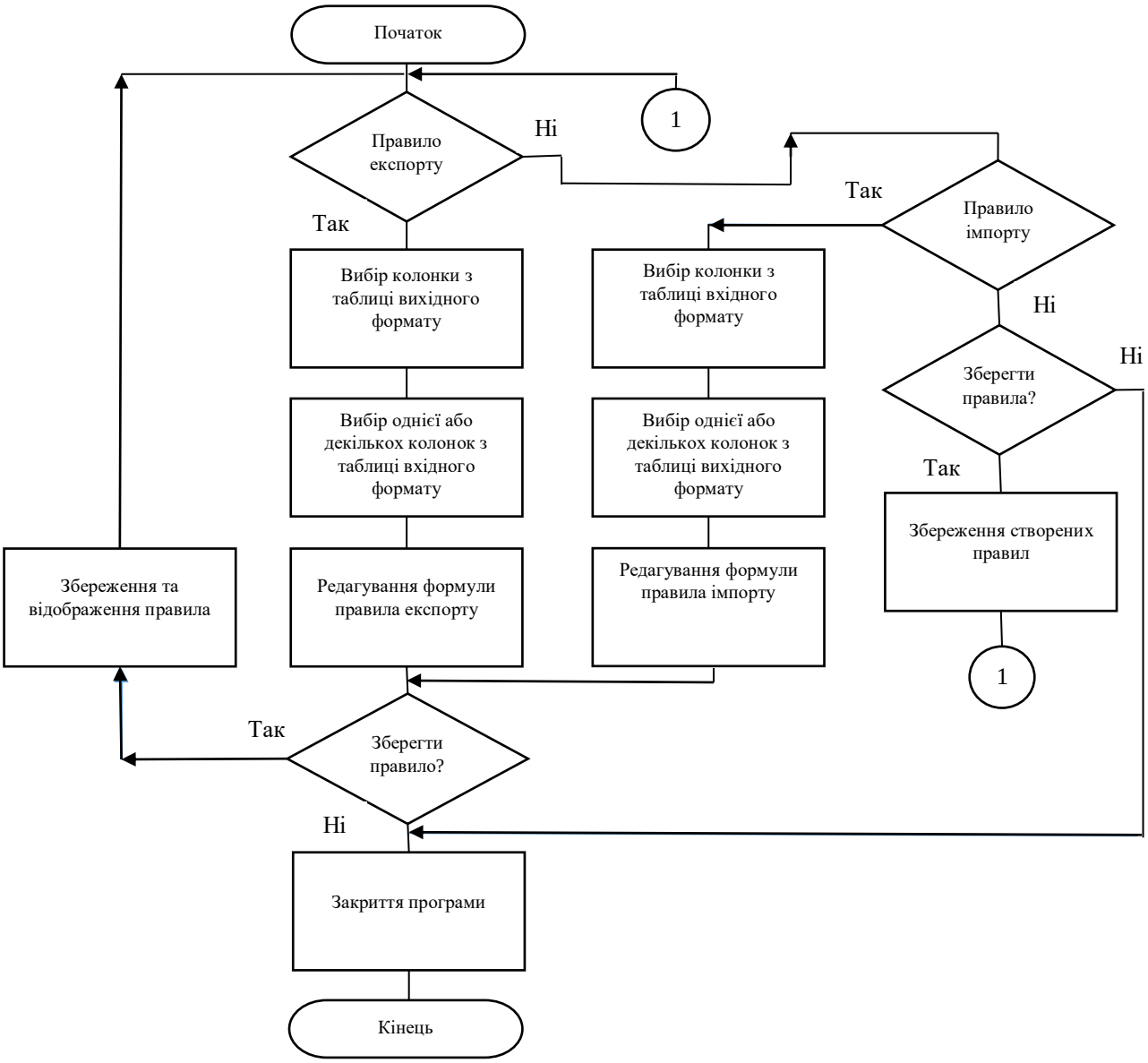

Рис. 4.5. Схема алгоритму створення правил конвертації

Компоненту реалізовано для вікна створення правил конвертації та відповідного елементу програмного інтерфейсу *ConvertionControl*. Вікно форми програмної компоненти створення правил конвертації зображено на рисунку 4.6.

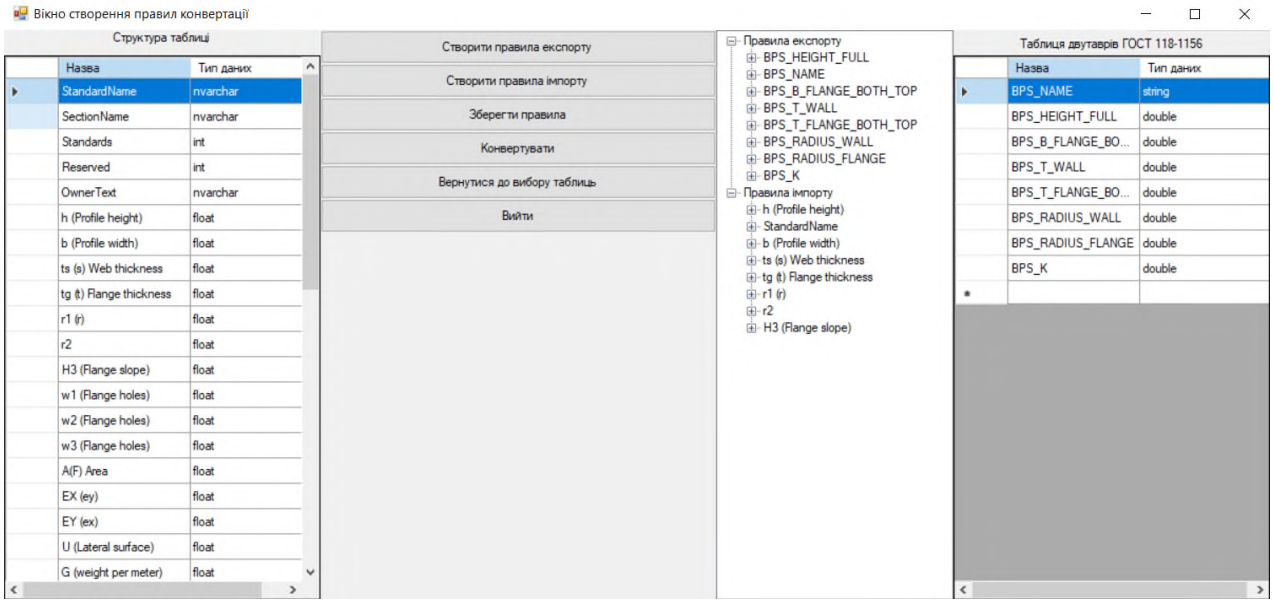

Рис. 4.6. Вікно форми програмної компоненти створення правил конвертації

Для зберігання шаблонів правил конвертації в оперативній пам’яті програми були створені класи *TransferRule* та *TemplatesData*. Класи дозволяють структурувати дані та реалізують інструменти роботи з ними.

Клас *TransferRule* – контейнер, призначений для зберігання даних правил конвертації. Зберігає дані про назву колонки, в яку будуть записані дані, та формулу строкового типу даних. Має конструктор, що отримує в якості вхідних параметрів дві змінні строкового типу: *ColumnName* та *ruleFormula*.

Клас *TemplatesData* – контейнер, призначений для зберігання даних шаблонів конвертації. Зберігає два об’єкти структурних схем таблиць вхідного та вихідного форматів та два списки правил конвертації, що містять правила експорту та імпорту. Клас має наступні параметри: *firstTableSchema*, *secondTableSchema*, *firstSchemaTransferRules* та *secondSchemaTransferRules*.

Параметр *firstTableSchema* – об’єкт системного класу *Dictionary*, що містить структурну схему таблиці вхідного формату у вигляді списку пар відношень *key-value*. Кожне відношення містить назву колонки та її відповідного типу даних.

Параметр *secondTableSchema* – об’єкт системного класу-контейнера *Dictionary*, що містить структурну схему таблиці вихідного формату у вигляді

списку пар відношень *key-value*. Кожне відношення містить назву колонки та її відповідного типу даних.

Параметр *firstSchemaTransferRules* – об’єкт системного класу-контейнера *List*, що містить список правил імпорту даних. Кожне правило імпорту є об’єктом створеного класу *TemplatesData*.

Параметр *secondSchemaTransferRules* – об’єкт системного класу-контейнера *List*, що містить список правил експорту даних. Кожне правило експорту є об’єктом створеного класу *TemplatesData*.

Також клас *TemplatesData* реалізує методи для ліпшого орієнтування в даних шаблону конвертації. До таких метод входять: *Defines*, *Order*, *FirstChemaIsEqual* та *SecondChemaIsEqual*.

Метод *Defines* – функція, що отримує в якості вхідного параметру структурну схему таблиці. Повертає булеве значення, що несе інформацію про те, чи визначає клас шаблону правила конвертації між даною структурною схемою таблиці та будь-якою іншою.

Метод *Order* – функція, що визначає чи структурні схеми таблиць розташовані в класі у вірному порядку.

Метод *FirstChemaIsEqual* – функція, що визначає, чи отримана структурна схема дорівнює об’єкту структурної схеми, що зберігається у параметрі *firstTableSchema*. Тобто, чи визначає шаблон конвертування вказану структурну схему як структурну схему таблиці вхідного формату.

Метод *SecondChemaIsEqual* – функція, що визначає, чи отримана структурна схема дорівнює об’єкту структурної схеми, що зберігається у параметрі *secondTableSchema*. Тобто, чи визначає шаблон конвертування вказану структурну схему як структурну схему таблиці вихідного формату.

Клас компоненти створення правил конвертації *ConvertionControl* реалізує наступні методи та оброблювачі подій: *importNextButton1_Click*, *FillDataGredViews*, *FillCurrentTemplateData*, *ExportButton_Click*, *DisplayExportRule*, *ImportButton_Click*, *DisplayImportRule*, *ExitButton_Click*,

exportNextButton1_Click, *saveRulesButton_Click*_1, *cancelingButton_Click* та *returnToPrevoiusControlButton_Click*.

Оброблювач подій *ExportButton_Click* спрацьовує при натисненні кнопки *ExportButton*, та розпочинає процес створення правила експорту. Метод відображає порядок дій при виборі колонок зі структурних схем таблиць та кнопку *exportNextButton1*.

Оброблювач подій *exportNextButton1_Click* спрацьовує при натисненні кнопки *exportNextButton1*, та продовжує процес створення правила експорту. Метод відображає інформацію про вибрані колонки в компонентах *exportRuleLabel3* та *ColumnToDefineLabel*, компоненту *ruleFormulaRichTextBox* з виведеним текстом формули експорту та кнопки *displayRulesButton* та *cancelingButton*. При натисненні однієї із кнопок компонента зберігає відредаговане правило експорту або припиняє процес створення правила відповідно.

Метод *DisplayExportRule* використовується у тілі оброблювача подій *displayRulesButton_Click*, виконує збереження правила експорту та його відображення в компоненті *rulesTreeView*. Після завершення процесу створення правила експорту відображає головне меню вікна.

Оброблювач подій *ImportButton_Click* спрацьовує при натисненні кнопки *ImportButton*, та розпочинає процес створення правила імпорту. Метод відображає порядок дій вибору колонок з структурних схем таблиць та кнопку *importNextButton1*.

Оброблювач подій *importNextButton1_Click* спрацьовує при натисненні кнопки *importNextButton1* та продовжує процес створення правила імпорту. Метод відображає інформацію про вибрані колонки в компонентах *importRuleLabel3* та *ColumnToDefineLabel*, компоненту *ruleFormulaRichTextBox* з виведеним текстом формули імпорту та кнопки *displayRulesButton* та *cancelingButton*. При натисненні однієї із кнопок компонента зберігає відредаговане правило імпорту або припиняє процес створення правила відповідно.

Метод *DisplayImportRule* використовується у тілі оброблювача подій *displayRulesButton_Click*, виконує збереження правила імпорту та його відображення в компоненті *rulesTreeView*. Після завершення процесу створення правила імпорту, відображає головне меню вікна.

Оброблювач подій *saveRulesButton_Click*_1 спрацьовує при натисненні кнопки *saveRulesButton*, та виконує збереження створених правил конвертації у файл бази знань засобами модуля роботи з файлами *XML* формату. У випадку, коли користувач не створив жодного правила конвертації, або модулю роботи з файлами *XML* формату по будь-яким причинам не вдалося зберегти правила, то користувачу відображаються відповідні повідомлення.

Метод *FillCurrentTemplateData* виконується в конструкторі елементу *ConvertionControl* та виконує відображення завантажених з файлу бази знань шаблонів конвертації правил у компоненті *rulesTreeView*.

Метод *FillDataGredViews* виконується в конструкторі елементу *ConvertionControl*, та виконує відображення структурних схем таблиць вхідного та вихідного форматів у компонентах *inputFormat_dataGridView* та *outputFormat_dataGridView* відповідно.

Оброблювач подій *cancelingButton_Click* спрацьовує при натисненні кнопки *cancelingButton* та припиняє процес створення правила. Відображає у вікні головне меню модуля створення правил конвертації.

Оброблювач подій *returnToPrevoiusControlButton_Click* спрацьовує при натисненні кнопки *returnToPrevoiusControlButton*, зберігає усі створені правила конвертації у файл бази знань та виконує перехід на попереднє вікно вибору даних для конвертації.

Процес конвертації розпочинається при натисненні кнопки *convertDatasButton* та реалізований у тілі методу *ConvertDatasButtonClickMethod*. Після виконання процесу конвертації компонента виконує перехід на наступне вікно відображення результатів конвертації.

Метод *ConvertDatasButtonClickMethod* реалізує алгоритм конвертації даних. Він має доступ до даних таблиці вхідного формату, та збережених правил

конвертації даних. У разі, якщо в оперативній пам’яті програми немає жодної строки даних таблиці або жодного правила конвертації, то метод відображає відповідне повідомлення. Метод виділяє пам’ять під об’єкт таблиці з структурною схемою вихідного формату та транслює в неї дані вхідної таблиці за визначеними формулами правил конвертації.

Алгоритм конвертації даних реалізовано в тілі циклу, кожна ітерація якого транслює строку даних з вхідної таблиці у вихідну. На початку кожної ітерації до таблиці вихідного формату додається пустий рядок даних. Потім у тілі ще одного циклу виконується процес трансляції даних комірок, кількість ітерацій якого відповідає кількості правил конвертації. Усі змінні назв колонок заміняються на значення комірок, що зберігаються у цих колонках, після чого отримане у результаті обчислення формули значення, записується у комірку вихідної строки. Схема алгоритму конвертації даних зображена на рисунку 4.7.

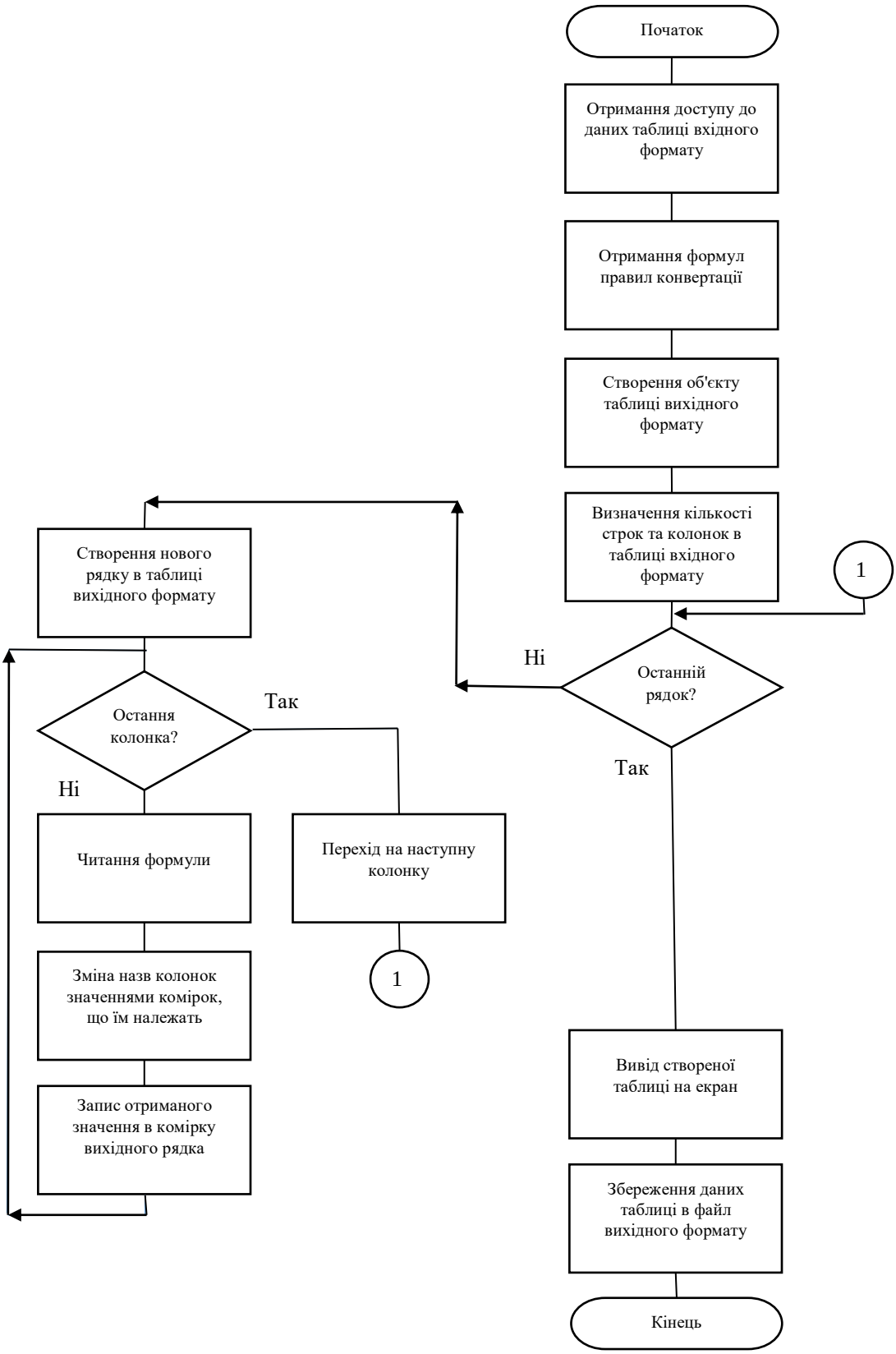

Рис. 4.7. Схема алгоритму конвертації даних

4.6. Визначення порівняльних характеристик при конвертації даних

Розроблена програмна система конвертації даних може бути використана як повноцінний програмний продукт або утиліта, що виконує конвертації даних при інформаційному моделюванні споруд. Також розглянуто можливість використання модулів створення *SQL*-запитів, роботи з файлами *XML* формату та створення правил конвертації відокремлено від модулів інтерфейсу користувача та відображення даних.

Такий спосіб використання реалізовано при проектуванні модуля синхронізації даних програми *LIRA* 10.12. Відмінність запровадженого процесу конвертації даних від старого полягає в тому, що стара технологія при конвертації об’єктів таблиць виконувала транслювання даних по вшитим в код програми відповідностям назв колонок із вхідної та вихідної таблиць, в той час як нова технологія використовує базу знань правил конвертації та шаблони при конвертації таблиць.

Переваги застосування розроблених модулів конвертації даних такі: можливість виконувати конвертацію даних між будь-якими форматами даних, можливість використання формул при створенні правил конвертації даних, можливість використовувати кілька баз знань шаблонів правил конвертації, можливість транслювати дані з двох і більше колонок таблиці, збільшення швидкості виконання процесу конвертації даних при збільшенні кількості форматів даних, відсутність необхідності користувачу знати будь-які мови програмування, мови створення запитів до баз даних чи мови розмітки файлів.

До недоліків застосування розробленої програмної системи входять: збільшення часу виконання процесу конвертації при конвертації простих форматів даних, необхідність створення правил конвертації даних та насичений інтерфейс програмної системи конвертації даних.

Для порівняння показників роботи старого та розробленого модулів конвертації даних проведено серію з десяти тестів. Кожен тест полягав у

виконанні конвертації формату даних з різною кількістю строк даних. Метою тестів є визначення середнього часу виконання процесу конвертації.

Для визначення часу виконання конвертації даних було використано інструменти *DiagnosticTools* середовища розробки *VisualStudio* 2019. Обов'язковою умовою використання інструментів діагностики є наявність початкового коду розробленої програмної системи конвертації даних та початкового коду модуля синхронізації баз даних програми *LIRA* 10.12 [19]. Результат виконання тесту конвертації таблиці засобами модуля синхронізації баз даних зображено на рисунку 4.8.

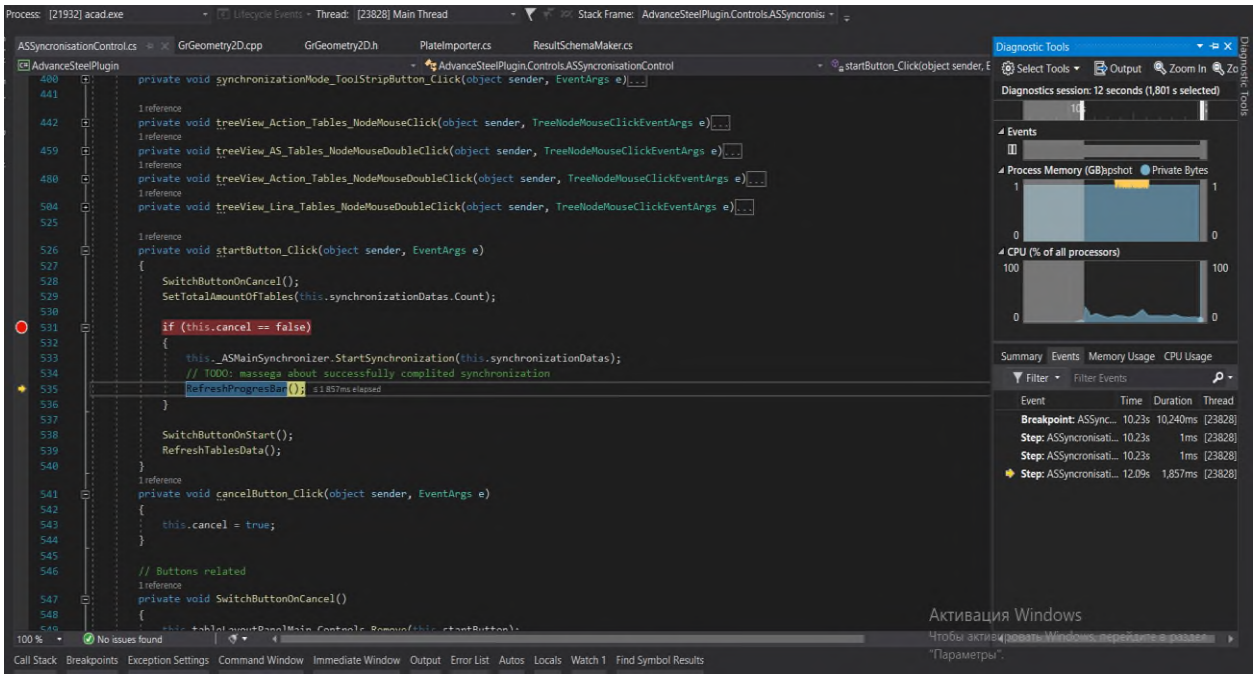

Рис. 4.8. Зображення показників часу конвертації даних засобами існуючого модуля

Процес конвертації даних таблиці виконується в тілі методу *StartSynchronization*. Час виконання процесу конвертації відображено на часовій шкалі засобами діагностики середовища розробки у правій частині зображення.

На часовій діаграмі використання системних ресурсів центрального процесору початок виконання процесу конвертації чітко виражений першим

різким стрибком використання *CPU*, що відбувся на десятій секунді процесу, та триває до закінчення сеансу діагностики.

Із зображення видно, що час конвертації таблиці, яка містить сто строк даних, засобами модуля синхронізації даних, займає 1.8 секунд. Також слід зазначити, що максимальний показник використання ресурсів центрального процесору не перевищує 25 відсотків.

Результат виконання тесту конвертації даних засобами модулів розробленої програмної системи конвертації даних зображено на рисунку 4.9.

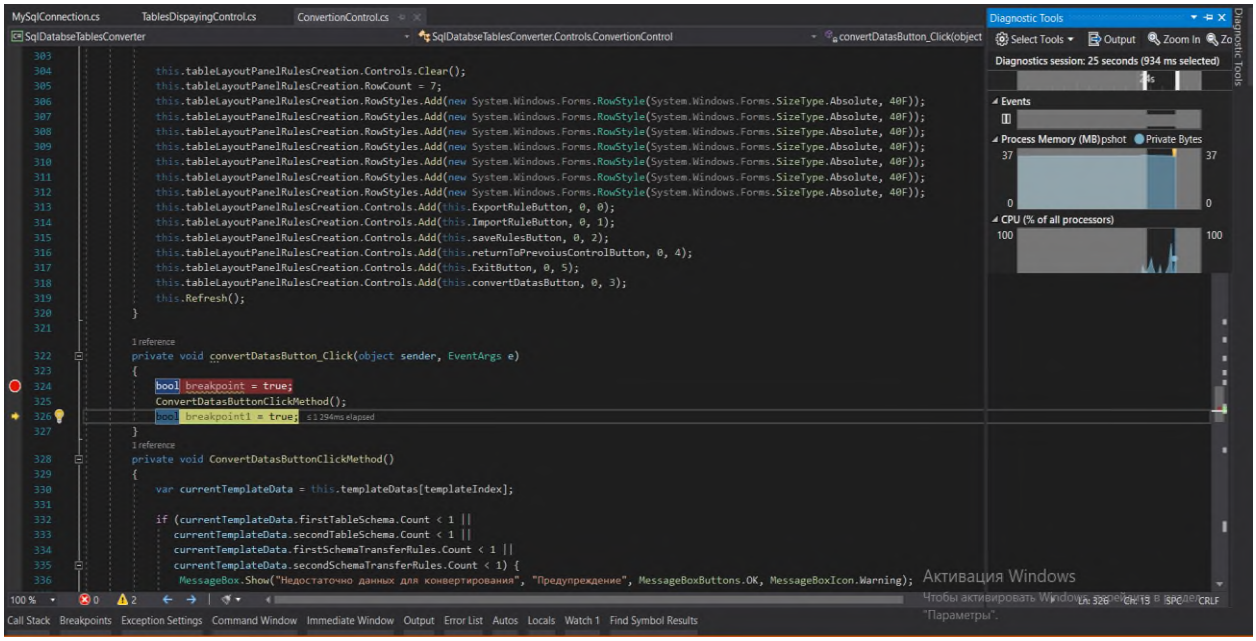

Рис. 4.9. Зображення показників часу конвертації даних засобами модулів розробленої програмної системи конвертації даних

Процес конвертації даних таблиці виконується в тілі методу *ConvertDatasButtonClickMethod*. Час виконання процесу конвертації відображено на часовій шкалі засобами діагностики середовища розробки у правій частині зображення.

На часовій діаграмі використання системних ресурсів центрального процесору початок виконання процесу конвертації чітко виражений першим

різким стрибком використання *CPU*, що відбувся на 24 секунді процесу, та триває до закінчення сеансу діагностики.

Із зображення видно, що час конвертації таблиці, яка містить сто строк даних, засобами модулів розробленої програмної системи конвертації даних, займає 95 мілісекунд. Також слід зазначити, що максимальний показник використання ресурсів центрального процесору дорівнює 68 відсоткам.

Графік залежності часу виконання процесу конвертації від кількості строк даних зображені на рисунку 4.10.

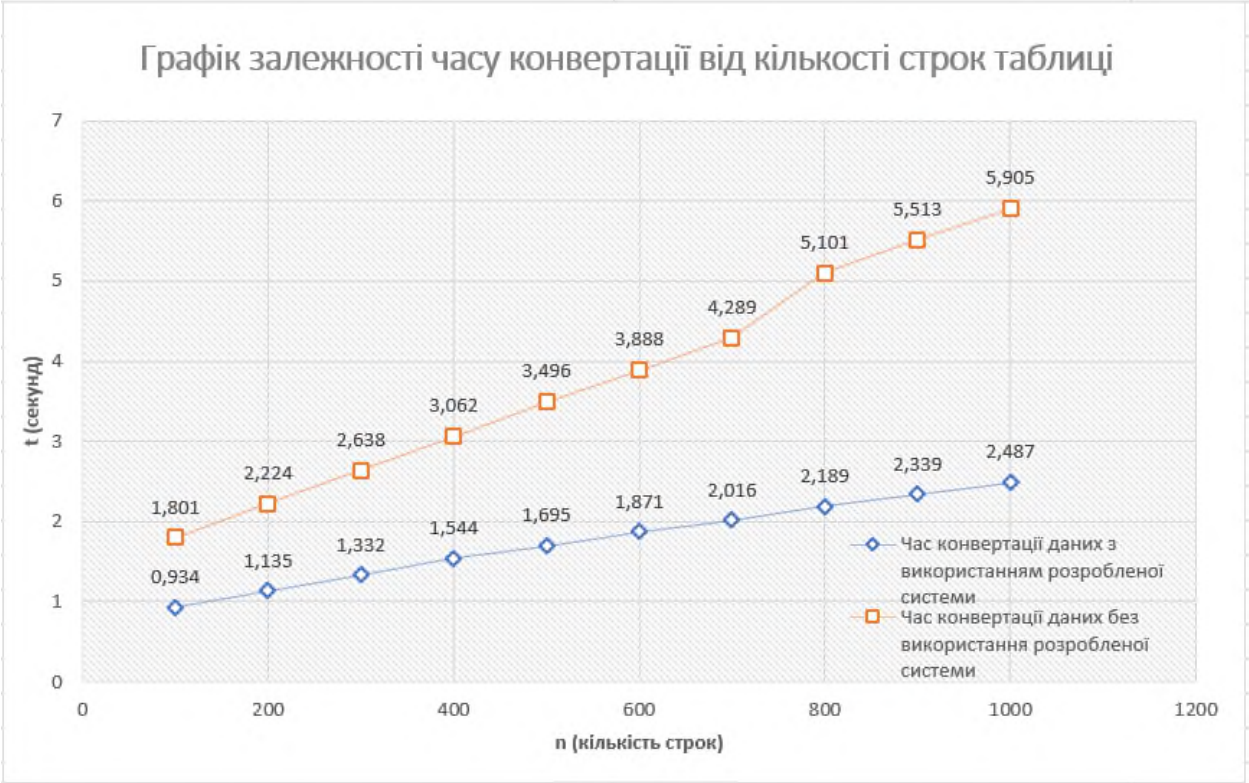

Рис. 4.10. Графік залежності часу конвертації від кількості строк таблиці

На графіках зображено зміни часу виконання процесу конвертації даних в залежності від зміни кількості строк таблиці. Було виведено формулу розрахунку середнього часу виконання процесу конвертації даних (4.1).

$$Tc = \frac{\left(\sum_{i=1}^{10} \left(T_{con}{}^{i}\right)\right)}{10} \tag{4.1}$$

де *Tc* – середній показник часу процесу конвертації за десять тестів

10 – кількість виконаних тестів;

i – ітератор перебору;

$T_{con}{}^{i}$ – час виконання процесу конвертації даних *i*-того тесту.

Виконавши набір тестів швидкості конвертації даних було розраховано наступні порівняльні характеристики:

$$T_{c\ (\text{попередній})} = 3.791 \text{ с.}$$

$$T_{c\ (\text{новий})} = 1.754 \text{ с.}$$

де $T_{c\ (\text{попередній})}$ – середній час виконання процесів конвертації даних засобами модуля синхронізації даних, а $T_{c\ (\text{новий})}$ – середній час виконання процесів конвертації даних засобами розроблених модулів програмної системи конвертації даних.

Слід зазначити, що час конвертації даних засобами старого модуля змінюється лінійно по відношенню до збільшення кількості строк таблиці. Також час конвертації може залежати від кількості можливих форматів таблиць даних. При виконанні конвертації даних засобами модуля синхронізації даних, у модулі було реалізовано можливість конвертації двадцяти різних форматів таблиць вхідного формату. Тобто, було визначено транслювання даних для двохсот відповідностей колонок.

Таким чином, збільшення кількості визначених пар колонок вплинуло б на час пошуку потрібної пари, та привело б до збільшення загального часу виконання процесу конвертації. Визначення повного часу виконання процесу конвертації даних виконується за такою формулою.

$$T_{con} = \sum_{i=1}^{N_{str}} \left(\sum_{j=1}^{Nc} \left(T_v{}^{ij} + T_{tran}{}^{ij} \right) \right)$$

де T_{con} – повний час виконання процесу конвертації даних таблиці засобами модуля синхронізації даних;

Nstr – кількість строк таблиці вхідного формату;

N_c – кількість колонок в структурній схемі таблиці вихідного формату;

$T_v{}^{ij}$ – час пошуку відповідності колонок з структурних схем таблиць вхідного та вихідного форматів;

$T_{tran}{}^{ij}$ – час транслювання даних з комірки вхідної строки в комірку вихідної строки;

i – ітератор перебору строки;

j – ітератор перебору колонок.

Натомість, при конвертації засобами розроблених модулів конвертації, час виконання процесів конвертації демонструє поступове зменшення. Визначення часу виконання процесу конвертації засобами модулів розробленої програмної системи конвертації даних виконується за такою формулою.

$$T_{con} = \sum_{i=1}^{N_{str}} (T_{reading}{}^{i} + \sum_{j=1}^{N_c} (T_f{}^{ij} + T_{tran}{}^{ij}))$$

де T_{con} – повний час виконання процесу конвертації даних таблиці засобами модулів розробленої програмної системи конвертації даних;

Nstr – кількість строк таблиці вхідного формату;

N_c – кількість колонок в структурній схемі таблиці вихідного формату;

$T_{reading}$ – час пошуку необхідних правил в базі знань шаблонів конвертації;

$T_f{}^{ij}$ – час читання формули правила конвертації;

$T_{tran}{}^{ij}$ – час транслювання даних з комірки вхідної строки в комірку вихідної строки;

i – ітератор перебору строки;

j – ітератор перебору колонок.

Зміни часу виконання процесу конвертації при поступовому збільшенні кількості строк даних зображено рисунку 4.11.

Рис. 4.11. Діаграма зміни часу конвертації при зростанні кількості строк даних

Час конвертації даних засобами старого модуля змінюється лінійно по відношенню до збільшення кількості строк таблиці. Також, можна помітити, що збільшення часу процесу конвертації даних засобами розроблених модулів програмної системи конвертації даних поступово зменшується.

Слід очікувати продовження прогресії до досягнення граничного значення, після чого значення зміни часу конвертації даних при збільшенні кількості строк стане сталим.

Це обумовлено тим, що відображений на графіку зміни часу конвертації даних засобами розроблених модулів час включає як реальний час конвертації, так і час пошуку в базі знань шаблонів відповідних правил конвертації. Загальна частка часу пошуку потрібних правил зменшується порівняно зі збільшенням кількості строк таблиці, що підлягає конвертації.

Модулі програмної системи конвертації даних слід використовувати при конвертації баз даних з великою кількістю повторюваних форматів структурних схем таблиць та середньою кількістю строк, що перевищує тисячу записів в базі даних.

4.7. Висновки до розділу

В даному розділі був описаний процес розробки програмної системи конвертації даних, перераховані та описані використані інструменти та технології, детально описані розроблені програмні компоненти, їх призначення, складові компоненти, методи та оброблювачі подій. Було детально описане середовище розробки *Visual Studio* 2019, шаблон проектування *Windows Forms Application C#* та всі використані мови програмування, розмітки та створення запитів до бази даних. Проект було виконано з дотриманням діючих стандартів та положень [20], [21]

Програмна система конвертації даних розроблена для таких цілей: перегляд існуючих об’єктів даних в вибраній базі даних, перегляд структурної схеми вибраного об’єкту таблиці, перегляд вмісту даних вибраного об’єкту таблиці, створення та редагування файлів баз знань правил конвертації даних *XML* формату, створення, редагування та видалення правил імпорту, створення, редагування та видалення правил експорту, перегляд створених правил конвертації даних, збереження правил конвертації даних в оперативній пам’яті програми, читання правил конвертації даних з файлу бази знань та їх запис в оперативну пам’ять програми, запис правил конвертації даних в файл бази знань з оперативної пам’яті програми та виконання та перегляд результатів процесу конвертації.

Інтерфейс програми реалізований трьома вікнами: вікно вибору даних для конвертації, вікно створення правил конвертації та вікно відображення результатів конвертації. Вони призначені для виконання відповідних етапів

конвертації даних. Детально описані компоненти інтерфейсу програми, їх розташування, та описана послідовність дій при роботі з вікнами.

Проект програмної системи було названо *SqlDatabaseTablesConverter*. До розроблених компонент програмної системи входять: компонента відображення даних, компонента створення *SQL*-запитів, компонента роботи з файлами *XML* формату, компонента створення правил конвертації та компонента конвертування даних.

Компонента відображення даних була розроблена для спрощення інтерфейсу програмної системи конвертації даних через відображення впорядкованих та інтуїтивно зрозумілих структур даних. Компонента відображає структурні схеми таблиць, дані таблиць, список об'єктів вибраної бази даних, текстову форму формул правил конвертації та дерево правил конвертації. Були перераховані та описані елементи, параметри, методи та оброблювачі подій компоненти відображення даних.

Компонента роботи з файлами *XML* формату був розроблений для процесів читання та запису даних в файл бази знань правил конвертації. Компонента роботи з файлами *XML* формату створює, відкриває, розшифровує та редагує файли формату **.xml*. Його основна функція полягає в читанні з файлу бази знань правил конвертації даних та їх групування у вигляді шаблонів конвертації.

Було описано основний клас модуля *XmlAccessor*, його параметри та простори імен та бібліотеки, що були використані при його розробці. Були описані наступні методи класу: *FindFile*, *WrightTemplates* та *ReadTemplates*.

Розроблено та описано формат розмітки файлу бази знань шаблонів правил конвертації, його основні елементи та такі команд: *WriteStartDocument, WriteComment, WriteStartElement, WriteString, WriteEndElement* та *WriteEndDocument*. Також були описані алгоритми читання та запису даних в файл бази знань шаблонів правил конвертації.

Компонента реалізації *SQL* запитів був розроблений для реалізації підключення до вибраної бази даних, створення запитів читання, редагування та

збереження даних та виконання їх вибірки. Основна функція компоненти полягає в читанні з файлу бази даних формату *.*mdf* таблиць з даними, їх структурних схем та даних. Було описано застосовані для розробки модуля простори імен, програмні інструменти, системні класи та такі методи: *isOpened*, *getAllTablesNames*, *getColumns*, *GetDataTable* та *GetDataTableReader*.

Компонента створення правил конвертації було розроблено для реалізації можливості завантаження, створення, редагування, збереження та видалення правил конвертації з файлу бази знань формату *.*xml* та оперативної пам'яті програми. Основна функція модуля полягає в наданні користувачу програми зручних інструментів для створення правил конвертації, та надання інструментів для адміністрування баз знань правил конвертації.

Було описано порядок дій при роботі з компонентою, вікно створення правил конвертації та розроблені класи *TransferRule* та *TemplatesData*. Перераховані та детально описані параметри, методи та оброблювачі подій, що використовуються в основному класі модуля. Також було розроблено та описано алгоритм створення правил конвертації та алгоритм конвертації даних.

Дослідження порівняльних характеристик було виконано через порівняння основних характеристик часу виконання процесу конвертації між існуючим модулем синхронізації даних програми *PC Lira* 10.12 та модулями розробленої програмної системи конвертації даних. До основних переваг застосування розроблених програмних компонент належать: можливість виконання конвертації даних між будь-якими форматами даних, використання формул при створенні правил конвертації, використання баз знань, транслювання даних з двох та більше колонок таблиці та швидкість виконання процесу конвертації.

Було виведено формулу знаходження часу виконання процесу конвертації таблиць (4.1), формули знаходження середнього часу конвертації (4.2 та 4.3) та проведено серію з десяти тестів з метою дослідження часу виконання процесів конвертації при зміні кількості строк таблиці, що конвертується. Результати дослідження порівняльних характеристик зображені на графіку залежності часу

конвертації від кількості строк таблиці (рис. 4.10) та діаграмі зміни часу конвертації при збільшенні кількості строк таблиці (рис. 4.11).

Також, було зроблено висновки щодо ефективності використання програмних компонент розробленої системи конвертації даних в порівнянні з існуючою програмою синхронізації баз даних та очікуваних змін часу синхронізації при подальшому збільшенні кількості строк таблиці, що конвертується. А також, зазначені особливості цільового використання розробленої програмної системи конвертації даних при інформаційному моделюванні споруд.

ВИСНОВКИ

Під час виконання дипломної роботи було детально розглянуто поняття інформаційного моделювання споруд, переваги та недоліки використання *BIM*-технології, етапи життєвого циклу проекту будівництва та переваги використання *BIM*-технології порівняно з традиційними системами комп’ютерного проектування.

Розглянуто існуючі засоби зберігання інформації при інформаційному моделюванні споруд. До таких засобів увійшли: параметри проекту, інформаційні моделі, об’єкти моделі, конструкції, параметри, бази знань та журнали подій. Порівняно переваги та недоліки винесення баз знань за межі моделі.

Розглянуто поняття систем керування базами даних (СУБД). Детально порівняно переваги та недоліки таких баз даних: *MySQL*, *Microsoft SQL Server*, *PostgreSQL*, *MongoDB* та *Microsoft Access*. До переваг перерахованих бази даних відносять легкість використання, безпечність, швидкість виконання запитів, підтримка багатьох операційних систем, інструменти візуалізації даних, динамічні запити та гнучкі мови виконання запитів до бази даних.

Розглянуто існуючу технологію конвертації даних, виведено мету та цілі виконання процесу конвертації. Також перераховані наступні етапи конвертації даних: створення файлу моделі вихідного формату, зчитування усіх необхідних даних з файлу моделі, створення параметрів конвертації, експорт усіх баз знань, експорт налаштувань та усіх об’єктів моделі.

Розглянуто архітектурні, конструкторські та обчислювальні програмні системи основані на використанні *BIM*-технології. Детально описані архітектурні програмні системи *Revit* та *ArchiCAD*, які дозволяють працювати над проектом одночасно декільком спеціалістам різного напряму, та конструкторські *PC LIRA* 10.10 та Advance Steel, за вирішення усього спектру завдань, пов'язаних з усіма етапами проектування та будівництва споруди.

Усі програмні рішення в галузі конвертації даних були поділені на вбудовані програмні компоненти конвертації в програмних системах керування баз даних та відокремлені програми або утиліти, призначені для конвертації файлів різних форматів. Були детально описані вбудовані програмні компоненти конвертації даних *EMS SQL Manager for SQL Server* та *Access2PostgreSQL PRO,* оскільки разом вони охоплюють дві бази даних, що часто використовуються в сімействі операційних системах *Windows*. Та відокремлені програмні системи *Access2MySQL SYNC* та *MyXMLData* через наявність у вільному доступі, швидкість проведення конвертації та спрямованість виконання процесів конвертації для вузького списку форматів.

Розглянуті та детально описані підходи для вирішення проблем конвертації, що основані на використанні баз знань правил конвертації, перехідних форматів, вбудованих інструментів конвертації та резервного копіювання як адміністрування. Усі перераховані підходи дозволяють спростити процес створення програмних рішень у галузі конвертації даних або зменшити використання системних ресурсів під час виконання процесів транслювання даних.

Детально розглянуто поняття конвертації даних, прямої та непрямої конвертації даних. Перераховані та детально пояснені основні проблеми конвертації даних, що включають: багатомодельність представлених даних, різницю в логічних структурах даних, використання різних мов для представлення текстової інформації та наявність різних типів форматів і постійний розвиток даних в процесі їх експлуатації.

Розглянуто поняття програмної системи конвертації даних при інформаційному моделюванні споруд, перераховані та детально описані основні операції, які має виконувати розроблена система конвертації даних, та перераховані основні інструменти, що були використані під час проектування та розробки програми. Основним інструментом є середовище розробки *Microsoft Visual Studio* 2019.

Спроектовані та описані п’ять програмних компонент. Програмна компонента відображення даних призначений для відображення вибраних в базі даних об’єктів таблиць, їх колонок, строк та структурних схем. Програмна компонента реалізації *SQL* запитів призначений для створення підключення до бази даних та виконання запитів написаних на мові *Transact-SQL*. Програмна компонента роботи з файлами *XML* формату призначений для реалізації функцій та методів читання та редагування файлів баз знань правил конвертації. Програмна компонента створення правил конвертації призначений для надавання користувачеві можливості створювати нові, завантажувати, редагувати та видаляти правила як імпорту, так і експорту даних

Для всіх програмних компонент були перераховані розроблені функції, параметри, методи, оброблювачі подій та простори імен, що були використані під час створення модулів. Також був наведений приклад структурної схеми таблиці двутаврів, описаний узагальнений алгоритм конвертації даних, детально описане поняття транзакції, описані основні інструкції та оператори мови запитів *Transact-SQL,* описане поняття елементу *XML*-декларації, спроектовано алгоритми запису та читання даних з файлу формату **.xml*, описана послідовність дій користувача при роботі з модулем та спроектовано два формати формул конвертації даних.

Був описаний процес розробки програмної системи конвертації даних, перераховані та описані використані інструменти та технології, детально описані розроблені модулі, їх призначення, складові компоненти, методи та оброблювачі подій. Було детально описане середовище розробки *Visual Studio* 2019, шаблон проектування *Windows Forms Application C#* та всі використані мови програмування, розмітки та створення запитів до бази даних.

Розроблено інтерфейс користувача, що реалізований між трьома вікнами: вікно вибору даних для конвертації, вікно створення правил конвертації та вікно відображення результатів конвертації. Вони призначені для виконання відповідних етапів конвертації даних. Детально описані компоненти інтерфейсу програми, їх розташування, та описана послідовність дій при робото з вікнами.

Розроблено та описано формат розмітки файлу бази знань шаблонів правил конвертації, його основні елементи та такі команди: *WriteStartDocument*, *WriteComment*, *WriteStartElement*, *WriteString*, *WriteEndElement* та *WriteEndDocument*. Також були описані алгоритми читання та запису даних в файл бази знань шаблонів правил конвертації.

Дослідження порівняльних характеристик було виконано через порівняння основних характеристик часу виконання процесу конвертації між існуючою програмою синхронізації даних програми *PC Lira* 10.12 та програмними компонентами розробленої програмної системи конвертації даних. До основних переваг застосування розроблених програмних компонент входять: можливість виконання конвертації даних між будь-якими форматами даних, використання формул при створенні правил конвертації, використання баз знань, транслювання даних з двох та більше колонок таблиці та швидкість виконання процесу конвертації.

Було виведено формули знаходження часу виконання процесу конвертації таблиць, формулу знаходження середнього часу конвертації та проведено серію з десяти тестів з метою дослідження часу виконання процесів конвертації при зміні кількості строк таблиці, що конвертується. Результати дослідження порівняльних характеристик зображені на графіку залежності часу конвертації від кількості строк таблиці (рис. 4.10) та діаграмі зміни часу конвертації при збільшенні кількості строк таблиці (рис. 4.11).

Також, було зроблено висновки щодо ефективності використання програмних компонент розробленої системи конвертації даних, в порівнянні з існуючими програмними компонентами синхронізації даних, очікувані зміни часу синхронізації при подальшому збільшенні кількості строк таблиці, що конвертується. А також, зазначені особливості цільового використання розробленої програмної системи конвертації даних при інформаційному моделюванні споруд.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Талапов В. Основы *BIM*: введение в информационное моделирование зданий. Пресс: М. ДМК, 2011. – 392 с.

2. Теличенко В. И. Описание предметной области строительства в информационных технологиях/ В. И. Теличенко, А. С. Павлов. – вид. *MGSU*, 2005. – 196 с.

3. *Kozlov I. M. Otsenka ekonomicheskoy effektivnosti vnedreniya informatsionnoho modelirovaniya zdaniy* . Новосибирск, 2010. – 308 с.

4. Трач Р. В. Інформаційне моделювання в будівництві (*BIM*): сутність, етапи становлення та перспективи розвитку/ Р. В. Трач.: 2014. – 440 с.: рис.; Предм. Указ.: с. 223 – 260.

5. *Green B. M. How Building Information Modeling is Contributing to Green Design and Construction. New York*: *McGraw-Hill Construction*, 2010. – 352 с.

6. Гогерчак Григорій, Інформаційні системи та бази даних: навчальний посібник, 2007. – 108 с.

7. Стивенз Р. К., Джоунс Э. Функции *SQL*. Справочник программиста. Киев: СПб. "Диалектика", 2007. – 768 с.

8. *Ben-gan I. Microsoft SQL Server* 2012 *T-SQL*. *Berlin*, 2015. – 413 с.

9. Джелен Б. , Майкл А. Сводные таблицы в Microsoft *Excel*. : Пер. с англ. – М. : ООО “И. Д. Вильямс”, Москва, 2007. – 320 с.

10. Махалов Є. В. Основи проектування баз даних. Москва, 2011. – 174 с.

11. *Phillips P. P., Burkett H. Data Conversion*: *Calculating the Monetary Benefits*. *San Francisco*, 2008. – 131 с.

12. *Krol J. Data conversion 101*: *Improving data accuracy*. *New York*, 2009. – 207 с.

13. Губич Л. В., Петкевич Н. И., Ковалев М. Я. Внедрение на промышленных предприятиях информационных технологий поддержки. Минск: РУП "Издательский дом", 2012. – 187 с.

14. Гусаренко А. С. Модели создания документов в формате *Office Open XML* на основе ситуационного подхода. Уфа: УГАТУ, 2015. – 57 с.

15. Шапошников И. Справочник *Web*-мастера. *XML*/ за ред. СПб. БФХ-Петербург, 2001. – 304 с.

16. Гогунський В. Д., Колеснікова К. В. Інформатика, основи програмування і застосування *ЕОМ*. Одеса: Наука і техніка, 2004. – 60 с.

17. Челяпин А. Проектное управление в сфере информационных технологий. Москва, 2009. – 189 с.

18. Шеховцов В. А. Операційні системи / за ред. НАН України. Київ: *BHV*, 2005. – 575 с.

19. Куссуль Н. Н., Шелестов С. В. Интеллектуальные вычисления в задачах обработки данных. Киев: "Наукова думка", 2018. – 337 с.

20. ГОСТ 2.301-68. Единая система конструкторской документации. Форматы. – Введ. 2002–01–01. – М. : Вид.-во стандартов, 2006. – 27 с.

21. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / Держстандарт України. – Вид. офіц. – [Чинний від 1995-02-23]. – Київ, 2007. – 86 с.

22. Бойчено С. В., Іванченко О. В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: Видавництво НАУ, 2017. – 63 с.

Додаток А

КОД МОДУЛЯ РОБОТИ З ФАЙЛАМИ *XML* ФОРМАТУ

```
public XmlAccessor(string _filePath = null) {
        if (_filePath == null || _filePath == String.Empty)
            _filePath = FindFile();
       //Creating new file in amoung Lira 10 steel base files
       if (_filePath == null || _filePath == String.Empty)
            _filePath = "D:\\Diplom_YaroslavSimya\\Templates.xml";

        this.filePath = _filePath;
    }
public List<TemplateData> ReadTemplates() {
        this.fileStream = new FileStream(filePath, FileMode.OpenOrCreate,
FileAccess.Read);
        if (!fileStream.CanRead)
            return null;
        using (XmlTextReader xmlTextReader = new XmlTextReader(fileStream)) {
          if (xmlTextReader == null)
            return null;
          xmlTextReader.WhitespaceHandling = WhitespaceHandling.None;
          var templates = new List<TemplateData>();
          var readingType = eReadingType.eNone;
          var elementType = eElementType.eNone;
          string keyBuffer = null;
          // Parse the file and display each of the nodes.
          try {
          while (xmlTextReader.Read()) {
          switch (xmlTextReader.NodeType) {
```

```
        case XmlNodeType.Element: {
                    if (xmlTextReader.Name == "Template")
                        templates.Add(new TemplateData());
                    else if (xmlTextReader.Name == "Schema1")
                        readingType = eReadingType.eSch1;
                    else if (xmlTextReader.Name == "Schema2")
                        readingType = eReadingType.eSch2;
                 else if (xmlTextReader.Name == "TransferRules1")
                        readingType = eReadingType.eTransferRule1;
                    else if (xmlTextReader.Name == "TransferRules2")
                        readingType = eReadingType.eTransferRule2;
                else if (xmlTextReader.Name == "Name")
                    elementType = eElementType.eName;
                    else if (xmlTextReader.Name == "Type")
                        elementType = eElementType.eType;
                    else if (xmlTextReader.Name == "ColumnName")
                        elementType = eElementType.eColumnName;
                    else if (xmlTextReader.Name == "Rule")
                        elementType = eElementType.eRule;
                    break;
          }
           case XmlNodeType.Text: {
                if (xmlTextReader.Value == String.Empty || xmlTextReader.Value ==
"\r\n")
                     break;

                    if (readingType == eReadingType.eSch1 && elementType ==
eElementType.eName)
                     keyBuffer = xmlTextReader.Value;
```

```
                if (readingType == eReadingType.eSch1 && elementType ==
eElementType.eType)
                    templates.Last().firstTableSchema.Add(keyBuffer,
xmlTextReader.Value);
                if (readingType == eReadingType.eSch2 && elementType ==
eElementType.eName)
                    keyBuffer = xmlTextReader.Value;
                if (readingType == eReadingType.eSch2 && elementType ==
eElementType.eType)
                    templates.Last().secondTableSchema.Add(keyBuffer,
xmlTextReader.Value);
                if (readingType == eReadingType.eTransferRule1 && elementType
== eElementType.eColumnName)
                    keyBuffer = xmlTextReader.Value;
                if (readingType == eReadingType.eTransferRule1 && elementType
== eElementType.eRule)
                    templates.Last().firstSchemaTransferRules.Add(new
TransferRule(keyBuffer, xmlTextReader.Value));
                if (readingType == eReadingType.eTransferRule2 && elementType
== eElementType.eColumnName)
                    keyBuffer = xmlTextReader.Value;
                if (readingType == eReadingType.eTransferRule2 && elementType
== eElementType.eRule)
                    templates.Last().secondSchemaTransferRules.Add(new
TransferRule(keyBuffer, xmlTextReader.Value));
            }
            break;
            }
            }
        }
        catch (System.Xml.XmlException e) {}
```

```
        if (xmlTextReader != null)
            xmlTextReader.Close();

        return templates;
        }
    }
  public bool WriteTemplates(List<TemplateData> templates /*,
WrightMode.eOnlyNew*/)
  {
        this.fileStream = new FileStream(filePath, FileMode.OpenOrCreate,
FileAccess.Write);
        if (!fileStream.CanWrite)
           return false;

        using (XmlTextWriter xmlTextWriter = new XmlTextWriter(fileStream,
Encoding.Default))
        {
        if (xmlTextWriter == null)
            return false;
        xmlTextWriter.Formatting = Formatting.Indented;
        //xmlTextWriter.Indentation = 4;
        xmlTextWriter.WriteStartDocument();
        //Start of writing
        xmlTextWriter.WriteComment("LIRA 10 Templates data file");
        // Write first element
        xmlTextWriter.WriteStartElement("Database");
        oreach (TemplateData template in templates)
        {
          xmlTextWriter.WriteStartElement("Template");
```

```
            //Writing first table schema
            xmlTextWriter.WriteStartElement("Schema1");
            foreach (KeyValuePair<string, string> keyValuePair in
template.firstTableSchema) {
                xmlTextWriter.WriteStartElement("Column");
                xmlTextWriter.WriteStartElement("Name", "");
                xmlTextWriter.WriteString(keyValuePair.Key.ToString());
                xmlTextWriter.WriteEndElement();
                xmlTextWriter.WriteStartElement("Type", "");
                xmlTextWriter.WriteString(keyValuePair.Value.ToString());
                xmlTextWriter.WriteEndElement();
                xmlTextWriter.WriteEndElement();

            }
            xmlTextWriter.WriteEndElement();
            //Writing first table schema
            xmlTextWriter.WriteStartElement("Schema2");
            foreach (KeyValuePair<string, string> keyValuePair in
template.secondTableSchema) {
                xmlTextWriter.WriteStartElement("Column");
                xmlTextWriter.WriteStartElement("Name", "");
                xmlTextWriter.WriteString(keyValuePair.Key);
                xmlTextWriter.WriteEndElement();
                xmlTextWriter.WriteStartElement("Type", "");
                xmlTextWriter.WriteString(keyValuePair.Value);
                xmlTextWriter.WriteEndElement();
                xmlTextWriter.WriteEndElement();

            }
```

```
 xmlTextWriter.WriteEndElement();
 //Writing first schema transfer rules
 xmlTextWriter.WriteStartElement("TransferRules1");
 foreach (TransferRule transferRule in template.firstSchemaTransferRules) {
        xmlTextWriter.WriteStartElement("Rule");
        xmlTextWriter.WriteStartElement("ColumnName", "");
        xmlTextWriter.WriteString(transferRule.columnName);
        xmlTextWriter.WriteEndElement();
        xmlTextWriter.WriteStartElement("Rule", "");
        xmlTextWriter.WriteString(transferRule.ruleFormula);
        xmlTextWriter.WriteEndElement();
        xmlTextWriter.WriteEndElement();
 }
 xmlTextWriter.WriteEndElement();
 //Writing second schema transfer rules
 xmlTextWriter.WriteStartElement("TransferRules2");
 foreach (TransferRule transferRule in template.secondSchemaTransferRules) {
        xmlTextWriter.WriteStartElement("Rule");
        xmlTextWriter.WriteStartElement("ColumnName", "");
        xmlTextWriter.WriteString(transferRule.columnName);
        xmlTextWriter.WriteEndElement();
        xmlTextWriter.WriteStartElement("Rule", "");
        xmlTextWriter.WriteString(transferRule.ruleFormula);
        xmlTextWriter.WriteEndElement();
        xmlTextWriter.WriteEndElement();
}
xmlTextWriter.WriteEndElement();
xmlTextWriter.WriteEndElement();
```

```
            }
        xmlTextWriter.WriteEndElement();
        //End of writing
        xmlTextWriter.WriteEndDocument();
        xmlTextWriter.Close();
        return true; }
}
private string FindFile() {
       var openFileDialog = new System.Windows.Forms.OpenFileDialog();
        openFileDialog.Title = "Окно создание или открытия файла для правил конвертации";
        openFileDialog.CheckFileExists = false;
        openFileDialog.CheckPathExists = false;
        openFileDialog.Filter = "XML Document (*.xml)|*.xml|All files (*.*)|*.*";
        var showDialogResault = openFileDialog.ShowDialog();
        if (showDialogResault != DialogResult.OK)
           return null;
        else
           return openFileDialog.FileName;
}
```