Київський національний університет імені Тараса Шевченка
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

Модернізація та інтеграція із зовнішніми каналами комунікації системи публікації новин Університету

Кваліфікаційна робота магістра
студента 2 року навчання
спеціальності 123 «Комп'ютерна інженерія»
Ірини КОПИЛ
_____(підпис)

Науковий керівник
к.т.н., асистент
Євген СЛЮСАР
_____(підпис)

Рецензент
к.ф.-м.н., доцент
Ім*я П РІЗВИЩЕ
_____(підпис)

До захисту допускаю
Протокол засідання кафедри від
"___" _____2023р. №_____

Завідувач кафедри
к.ф.-м.н., доцент
Юрій БОЙКО

КИЇВ 2023

РЕФЕРАТ

Кваліфікаційна робота містить 78 сторінок, 49 рисунків, 4 додатки, використано 17 інформаційних джерел.

ASP.NET CORE, АВТОМАТИЗАЦІЯ, ENTITY FRAMEWORK CORE, MySQL, MSSQL, .NET CORE, СИСТЕМИ КЕРУВАННЯ БАЗАМИ ДАНИХ, НОВИНИ

Об’єктом даної роботи є автоматизація процесу публікації дописів у стрічці новин інформаційного сайту. Предметом роботи є програмний засіб для керування публікацією новин на веб-сайті Університету.

Метою роботи є модернізація існуючої системи що виконує керування публікацією новин та її інтеграція з соціальними мережами.

Інструменти розроблення – інтегроване середовище розробки SQL Server Management Studio, вільно поширюваний фреймворк для створення веб-застосувань ASP.NET Core MVC, мова програмування C#, база даних MSSQL, мова запитів Transact-SQL.

Результати роботи – проведено аналітичний огляд методів інтеграції інформаційних систем та способів міграції даних, модернізовано програмний застосунок, який дозволяє користувачам керувати публікацією новин Університету в соціальних мережах у вигляді веб-інтерфейсу.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

БД	– база даних
СКБД	– система керування базами даними
EF	– Entity Framework
ORM	– об’єктно-реляційне відображення (Object-Relational Mapping)
SQL	– Structured query language (мова структурованих запитів)
MSSQL	– Microsoft SQL
MVC	– паттерн проектування Model-View-Controller

ВСТУП

Для зручної та успішної роботи майже кожна організація представлена веб-сторінкою у мережі Інтернет. Наш Університет не виключення, і також вся актуальна інформація про підрозділи, навчання та різноманітні ресурси присутні на веб-сайті. Для зручного керування публікаціями у стрічці новин було модернізовано панель адміністрування у вигляді веб-застосування. Вона дає змогу значно спростити процеси підготовки до публікації офіційних новин та більш доцільно використовувати ресурс співробітників. Процес модернізації системи також вимагає оновлення системи керування базами даних. Оновлення системи керування базами даних є важливим для забезпечення безпеки, продуктивності та сумісності системи, а також для використання переваг нових функцій і можливостей, які можуть допомогти ефективніше керувати даними.

В наш час світ має гостру необхідність диверсифікації засобів зовнішньої комунікації за рахунок поширення інформації у різних незалежних одне від одного засобах масової інформації. Така необхідність зумовлена періодичною недоступністю основних інформаційних ресурсів. Організації та підприємства окрім офіційних веб-сайтів використовують соціальні мережі для вирішення цього питання. Інтеграція з соціальними мережами може надати ряд додаткових переваг як організаціям, так і їх користувачам. Наприклад, охоплення ширшої аудиторії – платформи соціальних медіа мають мільярди користувачів у всьому світі, що може дозволити охопити набагато більшу аудиторію ніж за використання лише веб-сайтів.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ СИСТЕМ

1.1 Архітектура системи

Розглянемо архітектуру існуючої системи, реалізованої у вигляді веб-застосунку, зображену на рисунку 1.

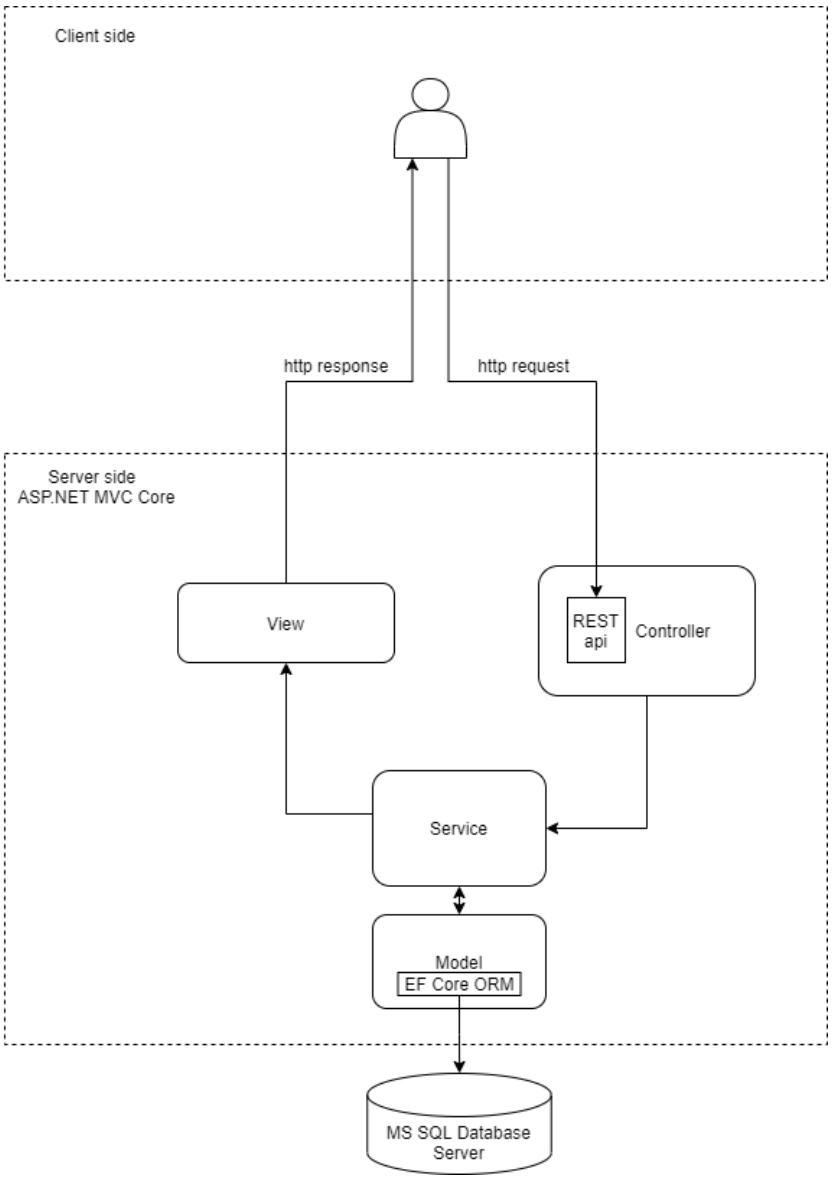


Рис. 1 – Загальна архітектура додатку

Структура веб-додатку передбачає наявність користувацької частини (front-end) та серверної (back-end), яка у свою чергу взаємодіє з базою даних та містить в собі всю бізнес-логіку.

Архітектура, що зображена на рисунку 1, дозволяє скоротити час

проектування системи, маючи попередньо готові елементи системи, збільшуючи надійність запроектованих процесів і програм обробки логіки, що скорочує час роботи з програмою.

Ефективні системи автоматизації включають такі основні компоненти: інтерфейс користувача та логіку програми.

Архітектурний шаблон модель-представлення-контролер (MVC, Model-View-Controller) визначає наступні групи частин застосунку – моделі, представлення, контролери. Цей патерн дозволяє розділити вирішення задач розробки на три основні блоки.

Модель - це компонент, що представляє дані з якими працює користувач. Можуть бути як прості моделі представлення, що передаються між представленням та контролером, так і моделі домену, що містять дані бізнес-логіки, операції та правила для дій над цими даними. Модель також відповідає за збереження загального стану системи та узгодження даних.

Представлення використовуються для перетворення певної частини моделі в інтерфейс користувача. Представлення містять в собі логіку, необхідну для відображення елементів моделі користувачеві, і ніщо крім цього.

Контролери оброблюють вхідні запити, виконують операції над моделлю та визначають представлення, які необхідно згенерувати для користувача.

При використанні шаблону MVC, користувацькі запити адресуються контролеру, який в свою чергу відповідає за роботу з моделями, для того, щоб виконати дії користувача та отримати результат запиту. Крім того, контролер обирає представлення для відтворення його користувачу, а також надає всі необхідні йому моделі.

Кожна частина архітектури MVC самодостатня, що відповідає принципу єдиної відповідальності. Логіка, що маніпулює даними в моделі міститься лише в моделі. Логіка, яка відображає дані лише в представленнях, а код, який обробляє запити користувачів міститься тільки в контролері. З чітким поділом між кожною з частин, додаток є легшим для підтримки і розширення протягом свого життєвого циклу, незалежно від того, наскільки великою є система. Такий

архітектурний шаблон дає можливості для розробки модуля інтеграції з соціальними мережами.

1.2 Програмна реалізація системи

Система реалізована на платформі ASP.NET MVC – це компонент платформи .NET Framework, що призначений для побудови високопродуктивних веб-застосувань. Фреймворк має власне середовище виконання, що сумісне лише з сімейством операційних систем Windows. Для доступу до системи кінцевих користувачів її необхідно опублікувати на сервер – відповідно такий, що буде сумісний з Windows-платформою. Для цього використовується IIS.

Internet Information Services - це веб-сервер загального призначення від Microsoft, який працює в системах Windows, що дозволяє публікацію веб-застосувань в мережу Інтернет.

Перевагами використання такого варіанту хостингу є простота публікації на сервер та зручний засіб управління опублікованими сайтами в одному місці. Проте, використання даного веб-серверу під інші платформи розробки є неможливим, адже IIS підтримує використання лише фреймворків .NET та PHP.

Розповсюдження додатків, розроблених за допомогою ASP.NET MVC можливе лише за рахунок IIS, який в свою чергу підтримується лише Windows-платформою.

Для роботи з базою даних використовується ORM – Entity Framework. Використання Entity Framework дає всі переваги ORM – робота з базою даних відбувається на рівні об’єктів, а відображення показує як властивості об’єкта пов’язані з таблицями і їх зв’язками в базі даних. Код, що генерується за допомогою ORM є оптимізованим внутрішніми засобами технології, що зменшує час на додаткове тестування, а отже для невеликих проектів є гарним рішенням задачі роботи додатку з базою даних. З іншого боку, через те, що Entity Framework є компонентом .NET Framework, продукти які розробленні з його використанням можуть працювати лише на системах з підтримкою .NET. Варто відзначити, що Entity Framework не має спеціального вбудованого засобу для автоматичного

підключення до використовуваної в проекті бази даних MySQL, вимагаючи додаткового використання механізмів, що дають такий доступ. Більше того, проект використовує підхід Database first, при якому модифікація даних можлива спочатку у самій базі даних, а потім вже в об’єктах системи, що значно збільшує необхідність контролю та витрати часу на внесення змін.

Для реалізації авторизації та аутентифікації проект використовує технологію ASP.NET Identity, що містить в собі єдину систему та набір шаблонів для зберігання інформації про користувачів у базі даних, інтеграцію з іншими системами авторизації, а також одним з компонентів Identity є провайдер ролей, що дає можливість керувати доступом користувачів на програмному рівні додатку. В поточній системі можливості керування користувачами не використовується.

В якості сховища даних – використовується використовується система керування реляційними базами даних MySQL. Найбільшими перевагами MySQL є безкоштовне поширення та підтримка багатьох платформ. З іншого боку, в базу даних не закладено певні обмеження функціоналу, які могли б бути доволі корисними при розробці, а також, не дивлячись на відкрите ліцензування, розробка системи є доволі повільною.

1.3 Структура проекту системи

На рисунку 6 зображена структура рішення існуючої системи керування публікацією новин.

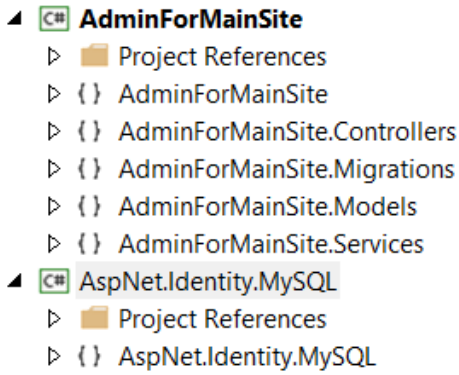


Рис. 2 – Структура рішення

З рисунку 2 можна побачити, що система складається з двох проектів, які містять в собі набір просторів імен для логічного розділення компонентів. Така структура додатку зумовлена необхідністю використання додаткового проекту для використання технології Identity та підключення до бази MySQL. Проект AspNet.Identity.MySQL містить в собі опис моделей та скрипти, необхідні для створення відповідних сутностей і таблиць, що використовуватимуться далі засобами Identity.

Така структура зменшує логічну зв’язаність двох компонентів рішення, що дає більше можливостей для допуску помилок при масштабуванні, а також збільшує час на розгортання одного рішення, адже всі кроки по публікації системи на хостинг необхідно виконати двічі.

1.4 Структура баз даних

Схему таблиць та зв’язків між сутностями, визначеними для зберігання даних процесу управління публікації новини можна побачити на рисунку 3.

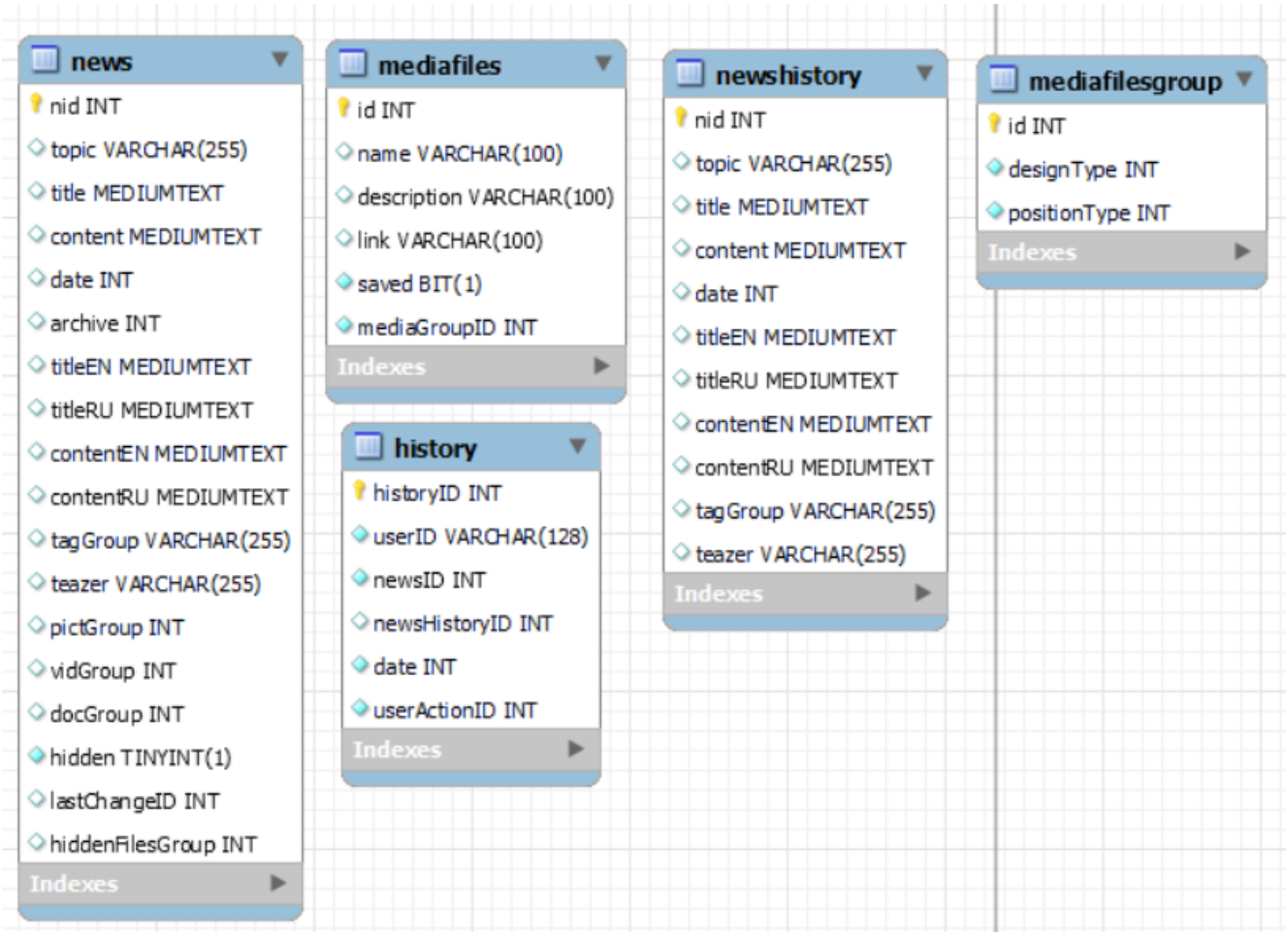


Рис. 3 – Схема сутностей та зв’язків для керування новинами

З рисунку 3 можна побачити структуру бази даних для зберігання сутностей предметної області, що стосуються публікації новин. Варто звернути увагу на відсутність зв’язків між таблицями за допомогою зовнішніх ключів, що суперечить основному поняттю цілісності даних.

Таке порушення проектування тягне за собою відсутність необхідних обмежень при встановленні взаємозв’язків між даними, а зрозуміти які зв’язки існують між сутностями можна лише з назв полів у таблиці. Крім того, для використання Enity Framework за таких умов необхідно вказувати всі можливі варіанти пошуку об’єктів, що робить час виконання запитів значно більшим, адже структура запиту не може бути оптимізованою.

На рисунку 4 зображена схема сутностей та зв’язків для керування користувачами панелі. Така структура таблиць є частиною технології ASP.NET Identity.

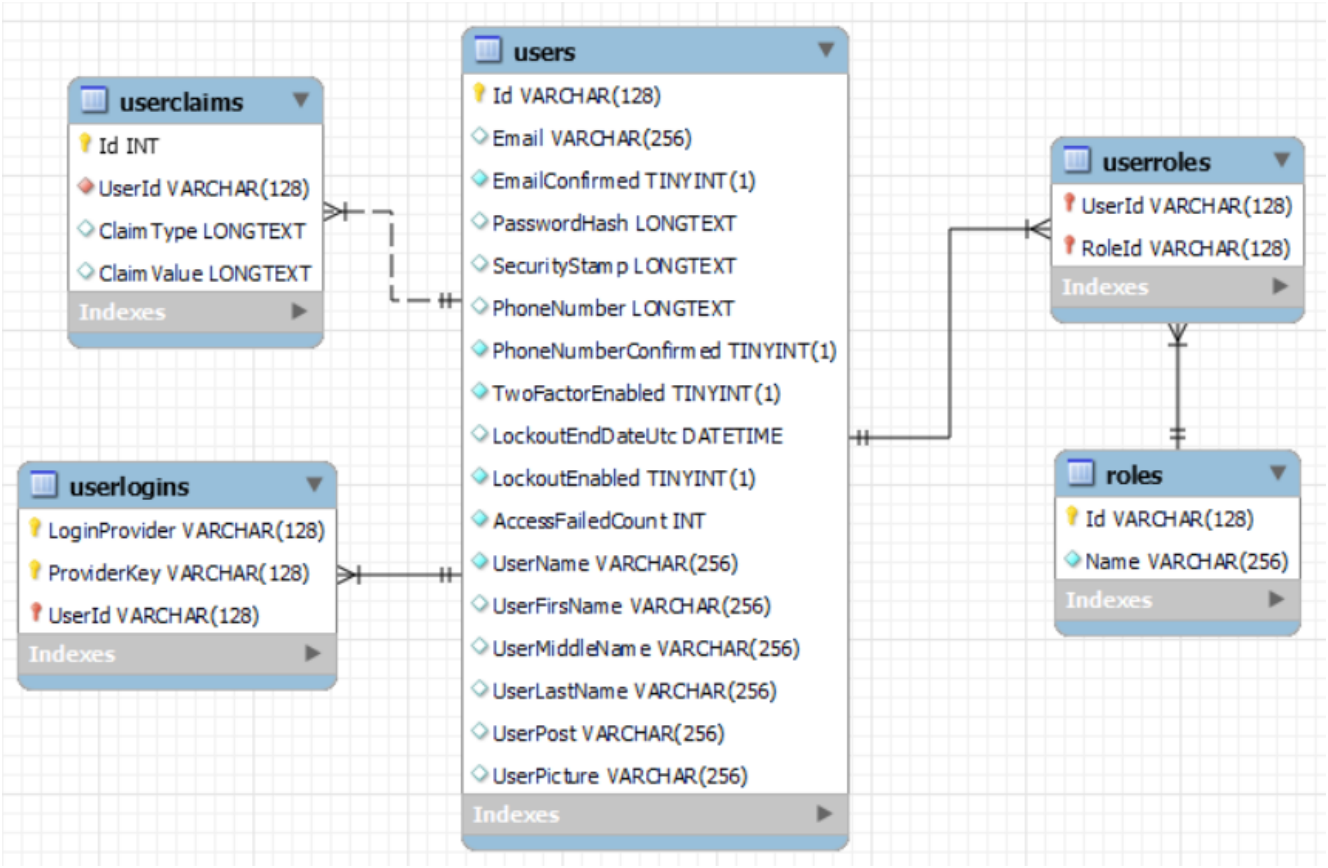


Рис. 4 – Схема сутностей та зв’язків для керування користувачами

З рисунку 4 видно, що сутності пов’язані між собою відповідно до всіх принципів ER-моделювання, що дає змогу ефективного використання всіх програмних засобів, наданих ASP.NET Identity.

Варто зазначити, що дані схеми відповідають двом різним незалежним базам даних, що знову ж таки суперечить принципам збереження цілісності даних.

1.5 Структура модернізованої системи

Для оптимізації розгортання об’єднано два проекти, один з яких був призначений для адміністративної роботи з користувачами. На рисунку 5 можна побачити оптимізовану структуру.

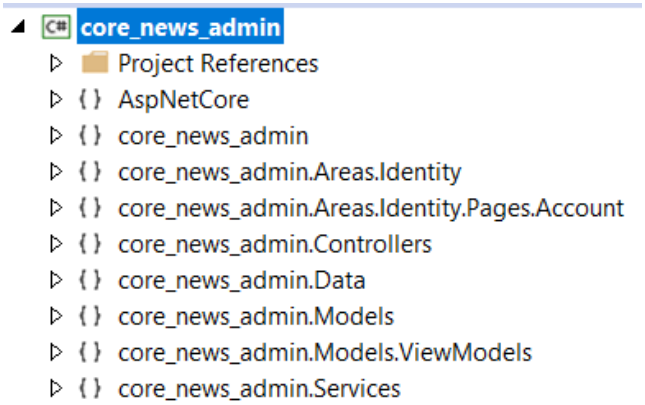


Рис. 5 – Структура оптимізованого проекту

Структура, зображена на рисунку 5 дає зручність у налаштуванні та підтримуванні працездатності системи, адже необхідність розгортання та підключення до бази даних існує лише одного разу, а всі зміни вносяться в єдину систему, що покращує також фіксацію контролю версій та залежностей між складниками системи.

1.6 Структура бази даних модернізованої системи

Для збереження об’єктів панелі управління публікацією новин використовується Microsoft SQL Server.

MS SQL Server – це система керування базами даних, яка використовує для організації баз даних реляційну модель. Реляційні СКБД передбачають зберігання даних у вигляді таблиць, між якими організовано зв’язки.

На рисунку 6 зображена загальна база даних для повноцінної системи з модулями сутностей новин і сутності, призначені для зручної реалізації авторизації та керування користувачами системи.

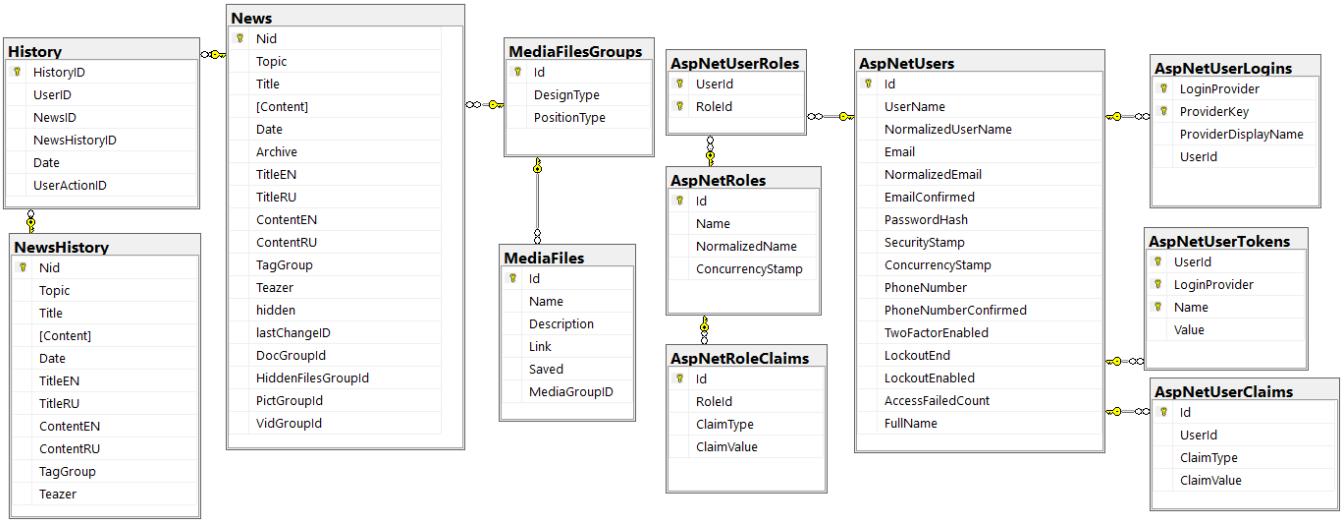


Рис. 6 – Схема бази даних додатку

Через зміну структури баз даних виникає необхідність розробки механізму перенесення даних. Відповідно також через зміну системи керування базами даних необхідно провести аналіз сумісності систем та розробити план і спосіб перенесення даних.

1.7 Порівняння систем керування базами даних

Статистичні дані досліджень компанії Statista показують – MySQL та Microsoft SQL займають відповідно 2 та 3 місце в рейтингу найбільш використовуваних систем керування реляційними базами даних [1].

Обидві системи є масштабованими, підтримують різні мови програмування та типи даних. MySQL є системою з відкритим вихідним кодом, на відміну від Microsoft SQL, який вимагає ліцензування. Крім того, MySQL є кросплатформним і підтримується більшістю операційних систем, таких як Windows, Linux, macOS, тоді як Microsoft SQL наразі може бути розгорнутий на платформі Windows або ж у вигляді контейнерів. Наприклад, Microsoft SQL Server 2019 може бути розгорнутим для наступних дистрибутив ОС Linux: RHEL (версії 7.3-7.6), SELS v12 SP2, Ubuntu 16.04 LTS.

Проте, оскільки MSSQL був розроблений саме для Windows-проектів, він найчастіше використовується для інтеграції з додатками, що застосовують платформу .NET. Також, MSSQL дозволяє налаштовувати Entity Framework класи

в програмному коді застосунків, що дозволяє будувати оптимізовані LINQ-запити. У випадку використання MySQL необхідно застосовувати сторонні бібліотеки для цієї можливості.

Для резервного копіювання при роботі з MySQL необхідно вилучати дані за допомогою SQL-запитів. СКБД дозволяє блокувати базу даних від модифікації на час створення резервної копії, що знижує можливість пошкодження або втрати даних. Такий механізм має основний недолік – відновлення даних з копії може займати багато часу, оскільки потребує виконання багатьох SQL-запитів. MSSQL не блокує доступ до бази даних під час резервного копіювання, що дає змогу користувачам архівувати та відновлювати дані великого об’єму.

Також MySQL не дає можливості відміни запущених запитів користувачам, можлива лише відміна всього процесу. MSSQL дозволяє припинення виконання запиту і використовує механізм транзакції для забезпечення цілісності даних.

З точки зору безпеки обидві системи підтримують шифрування, автентифікацію та авторизацію користувачів, але MSSQL надає розширені налаштування доступу та класифікації даних , як наприклад керування ролями користувачів чи визначення рівнів доступу до власне рядків у таблицях.[2]

1.8 Постановка задачі

Проаналізувавши існуючі системи, можна виділити наступні задачі:

- Розробка та реалізація плану міграції даних між системами управління базами даних MySQL та Microsoft SQL Server
- Вибір та проектування внутрішніх компонентів існуючої системи, що забезпечать інтеграцію з соціальними мережами
- Програмна реалізація інтеграції на основі проведеного аналізу способів інтеграції автоматизованих систем

РОЗДІЛ 2. ПЕРЕНЕСЕННЯ ДАНИХ ТА ПРОЄКТУВАННЯ ІНТЕГРАЦІЇ З СОЦІАЛЬНИМИ МЕРЕЖАМИ

2.1 Типи рішень для переміщення даних

З ростом розміру організацій зазвичай зростає кількість інформаційних систем, що застосовуються, і відповідно зростає кількість даних, які мають різні моделі, типи, призначення та час зберігання. Зі збільшенням кількості ресурсів, які використовуються постають питання способів обробки, керування даними, витрат та якості даних для аналізу і використання в різнорідних системах. Вирішенням подібних задач є оптимізація існуючих інформаційних систем, або ж зменшення їх кількості для зниження витрат. Переміщення та консолідація даних, а також планування їх взаємодії, можуть покращити будь-який бізнес-процес – від аналітики до розробки додатків.

Основними типами рішення для переміщення даних є оновлення, міграція, консолідація, дистрибуція та інтеграція, що відрізняються за наступними параметрами:

- **Кількість джерел даних**. Перетворення даних з вихідних до кінцевих систем можуть бути реалізовані у вигляді зв’язків один-до-одного (міграція або оновлення), багато-до-одного (консолідація), багато-до-багатьох (інтеграція). Така класифікація дозволяє розрізнити схожі типи рішень для переміщення даних.
- **Структура вхідних і вихідних моделей.** При переміщенні даних необхідно враховувати різницю між вихідною та цільовою структурами даних. Час перетворення та переміщення даних пропорційний різниці моделей та їх кількості.

Тип	**Відношення кількості джерел даних і кінцевих систем**	**Різноманітність моделей даних**	**Приклад**
Консолідація	Багато до одного	Або гомогенні, або гетерогенні	Об’єднання кількох систем в одну, що включають в себе

			однорідні бази даних.
Міграція	Один до одного	Гомогенні	Зміна системи керування базами даних. Оскільки платформи різко відрізняються, моделі даних відрізнятимуться.
Дистрибуція	Один до багатьох	Або гомогенні, або гетерогенні	Дані необхідно розподілити з однієї центральної системи на кілька периферійних відповідно до використання.
Оновлення	Один до одного	Зазвичай гомогенні	Зміна версії комплексної системи, включаючи умову попереднього одночасного використання різних версій систем.
Інтеграція	Багато до багатьох	Гетерогенні	Взаємний обмін даними між різними системами без їх зміни. (складність задачі значно росте, коли збільшується кількість неоднорідних джерел даних та кінцевих систем)

Таблиця 1 – Типи рішень переміщення даних

Міграція даних – це процес, що включає етапи вилучення, вибору і перетворення даних між інформаційними системами. Зазвичай виконується одноразово та в одному напрямку між вихідною та цільовою системами.

Консолідація даних передбачає об’єднання даних між окремими системами в меншу кількість джерел або ж навість в одне джерело. Консолідація використовується для спрощення керування системою або використання системи, що має здатність до масштабування. Часто цей процес вимагає зведення даних з різних джерел до схожої структури, що є підпроцесом міграції.

Оновлення даних зазвичай відбувається в єдиний крок – копіювання даних. Проте, під час оновлення даних також може бути необхідним застосувати процес

міграції для встановлення відповідності даних. Наприклад, у випадку коли версії програмного забезпечення значно відрізняються структурою чи налаштуванням.

Дистрибуція даних передбачає копіювання всього набору даних або його частини до одного або кількох місць розташування. Наприклад, копіювання відбувається до центрального сховища або системи прийняття рішень, що дозволяє надавати частковий доступ користувачам для подальшого аналізу, без впливу на продуктове середовище. Дані можуть бути трансформовані або відсортовані для використання в кінцевій системі.

Інтеграція даних передбачає процес поєднання кількох непов'язаних джерел даних для надання їх користувачу єдиного доступу до них. Складність процесу зростає у різних ситуаціях, наприклад, у випадку об'єднання даних досліджень з різних сховищ. Для цього необхідним етапом є трансформація даних, що також є частиною міграції даних.

Отже, міграція даних (або частини цього процесу) є зазвичай складовими інших типів рішень для переміщення даних [3].

2.2 Класифікація міграції баз даних

Наразі немає єдиного стандарту щодо міграції даних, оскільки кожен окремий процес міграції даних є унікальним за своїми вихідними системами, кінцевими користувачами, цільовими системами та вимогами, що висуваються до них. При цьому варто звернути увагу, що, не зважаючи на обраний підхід, основною задачею міграції даних є збереження всіх процесів, що використовують ці дані, незмінними.

2.2.1 Реплікація

Реплікація – це процес, що забезпечує продовжуваного копіювання даних з однієї бази даних в множину інших для забезпечення їх відповідності в реальному часі. Програмне забезпечення вилучає та завантажує дані в цільову систему. Під час реплікації дані можуть фільтруватись чи трансформуватись. Процес вимагає контролю та моніторингу.

Реплікація може використовуватись для розповсюдження даних для додатків в одному і тому самому, або різних середовищах. Таке розповсюдження може бути як копіюванням, так і комплексним перетворенням даних для забезпечення вимог відповідних середовищ. Наприклад, застосунки мають кілька версій, які потребують одночасної підтримки. Розповсюдження забезпечить використання історичних даних в нових версіях, поки не буде виконане оновлення.

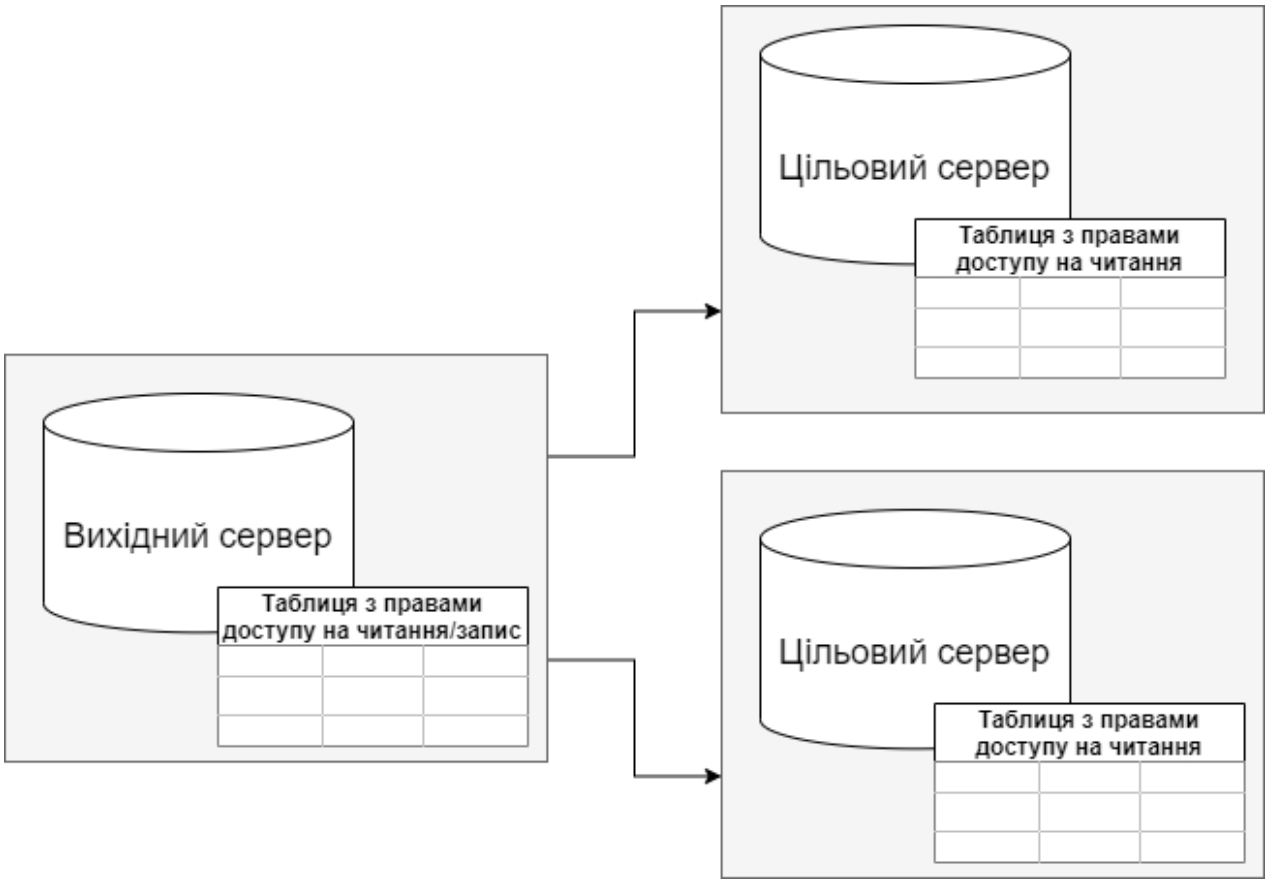


Рис. 7 – Розповсюдження даних за допомогою реплікації

На рисунку 7 можна побачити що цільові сервери можуть отримувати різні набори даних з вихідного джерела.

Реплікація також може застосовуватись для збору даних з різних розподілених систем в одну, для подальшої обробки, аналізу, звітності, або використання у інших застосунках. Наприклад, кожен підрозділ університету має програму, що контролює успішність студентів, а університет має застосунок, що потребує агрегованих даних за всіма студентами.

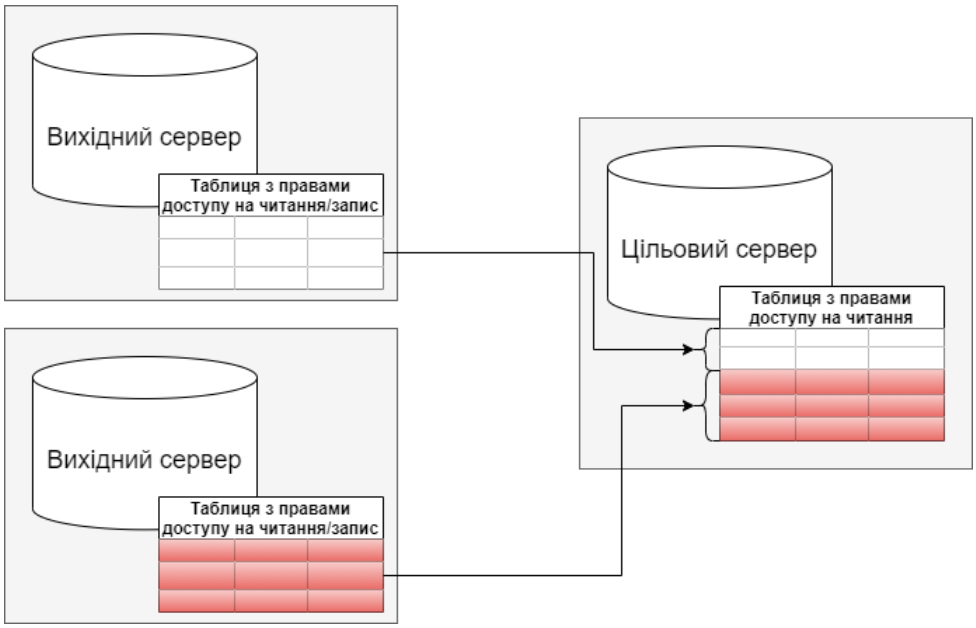


Рис. 8 – Консолідація даних за допомогою реплікації

З рисунку 8 можна побачити реплікацію даних з двох вихідних джерел, що реалізує консолідацію (об’єднання) даних в одному цільовому. Варто зазначити, що структури даних у вихідних серверах мають бути однаковими.

Одним з варіантів використання реплікації є двосторонній обмін даними (або ж «master -slave») – вихідний сервер розповсюджує дані до цільових серверів, якщо на них відбуваються зміни, оновлені дані реплікуються назад до вихідного, через який потрапляють до інших екземплярів цільових серверів.

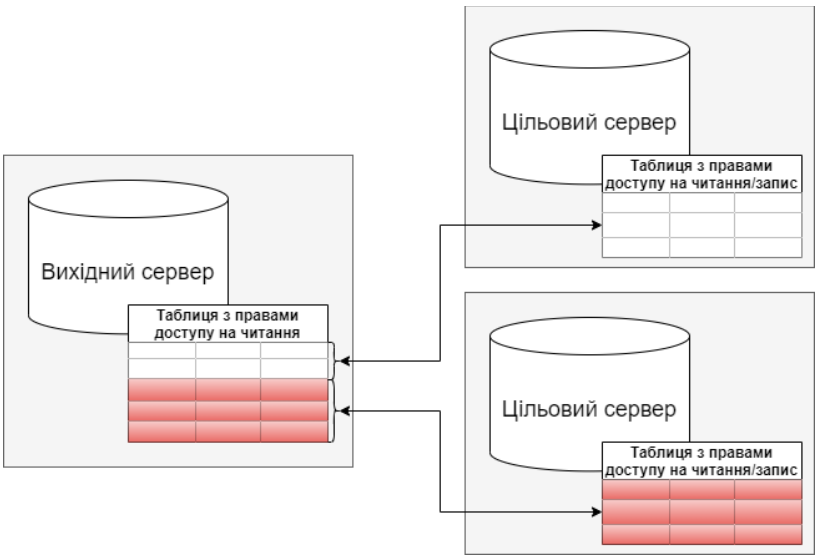


Рис. 9 – Двонапрямлена реплікація

На рисунку 9 зображено двонапрямлену реплікацію даних з визначеним

керуючим вихідним сервером.

Перевагою застосування реплікації є можливість зменшення часу простою під час перенесення даних, або є взагалі його уникнення за рахунок того що вихідна версія залишається активною.

Проте, у всіх випадках використання основною вимогою до сховищ даних є їхня повна або часткова сумісність, відповідно системи керування базами даних повинні мати одну і ту саму платформу і типи даних [5]. Зазвичай, реплікація має визначений час початку і не має кінцевого часу, оскільки використовується для потокового переміщення даних. Вона може бути зупинена або перетворена на міграцію даних.

2.2.2 Процес ETL

Процес ETL – витяг (з англ. extract), трансформація (з англ. transform), завантаження (з англ. load), складається з 3 етапів та вирішує задачу інтеграції даних від вихідного сховища до цільового. Основними етапами є вилучення даних, їх перетворення та відповідно завантаження.

- Вилучення даних – початковий етап, що виконує вилучення даних з цільового джерела. На початку етапу необхідно визначити типи даних, їх структуру та зв’язки між ними. Як структуровані, так і не структуровані дані завантажуються в проміжне сховище, яке може бути тимчасовим, або зберігати архів даних.
- Трансформація – етап, що забезпечує застосування правил, які забезпечують цілісність та якість даних. Після етапу трансформації дані мають бути підготовленими для подальшого використання і сумісними для застосування у цільовому середовищі. Наприклад, системи керування базами даних підтримують різні кодування. В такому випадку на етапі трансформації необхідно узгодити набори символів для запобіганню втрати інформації.
- Завантаження – останній етап процесу. Дані, що піддались трансформації, завантажуються у цільове сховище або повністю, або частинами з певними часовими інтервалами [4].

Процес ETL може бути реалізований будь-якими програмними засобами залежно від вихідної та цільової платформ збереження даних. Варто зазначити, що їх повна сумісність є необов’язковою, а відповідність правилам переміщення забезпечує етап трансформації. Розрізняють кілька видів ETL-процесу: мануальний, автоматизований, та їх комбінація.

Мануальний ETL-процес. Реалізація мануального ETL-процесу передбачає власну імплементацію та програмування етапів вилучення, трансформації та завантаження. Мануальний підхід вимагає спеціального попереднього аналізу, побудови та документування архітектури кожного з кроків для всіх джерел даних, що дозволяє створити унікальне вирішення задачі, оптимізоване з точки зору продуктивності виконання та подальшої підтримки, або застосувати додаткові інструменти, що не підтримуються автоматизованими засобами. Проте, варто пам’ятати, що для різних типів даних необхідний окремий опис кроків виконання міграції. Витрати ресурсів на підтримку та подальший розвиток такого рішення збільшуються з ростом кількості даних та сховищ.

Автоматизований ETL-процес. Автоматизація дозволяє зберігати та документувати кожну ітерацію процесу, забезпечуючи створення необхідних даних для аналізу та візуалізації. Час на розробку такого рішення значно скорочується за рахунок автоматичної генерації коду ETL-етапів. Замість окремих дискретних кроків, які виконуються мануально, всі етапи ETL-процесу приховані засобом автоматизації, що відповідно дозволяє проводити міграцію даних в один крок. Коли з’являється необхідність проведення нової міграції згенерований код може змінюватись в залежності від параметрів налаштування, але залишатись застосовним для нового середовища.

Відповідність мігрованих даних стандартам забезпечується збереженням точних та актуальних технічних метаданих (вихідне джерело, які зміни і трансформація була проведена, кінцеве джерело). Наприклад, перехід до іншого типу системи керування базами даних – від реляційної до типу ключ-значення потребує повного оновлення програмного коду ETL-процесу при зміні типів даних та їх структур. Автоматизовані засоби в даному випадку дозволяють

зберегти етап вилучення, а зміни потребуються наприклад на кроці трансформації, що забезпечує створення схеми бази даних, перетворення даних та завантаження. Час, що необхідний на реалізацію змін, відповідно зменшується [6].

2.2 Узгодженість міграції

Міграція баз даних має бути узгодженою, що означає відповідати наступним умовам:

- Повнота – всі дані, або їх підмножина, що визначені для перенесення, мігровані.
- Відсутність дублів – в цільовій системі дані містяться без повторень (дублів), кожна їх частина перенесена один і тільки один раз.
- Послідовність – цільова база даних піддається змінам у тому ж порядку, що і вихідна. Послідовність змін забезпечує узгодженість даних [7].

2.3 Методи інтеграції систем

Сукупність програмних засобів, архітектурних шаблонів, інструментів та методів, що дозволяють доставку даних та доступ до них реалізують інтеграцію систем. Інформаційні сервіси можуть замінюватись, оновлюватись та видалятись з інтеграції. Методи програмних та системних інтеграцій дозволяють узгоджено та ефективно провести інтеграцію. Програмні компоненти і підсистеми сполучаються у визначений документований спосіб для забезпечення надійного поєднання програмних засобів та елементів систем. Рівні інтеграції та план розробки визначають чи спроектовані елементи готові для верифікації та валідації результатів [8].

2.3.1 Інтеграція на рівні бази даних

Спосіб інтеграції на рівні бази даних передбачає єдине середовище (базу даних) для зберігання та застосування даних. Централізована база даних дозволяє надавати спільний доступ до даних застосункам, що його потребують.

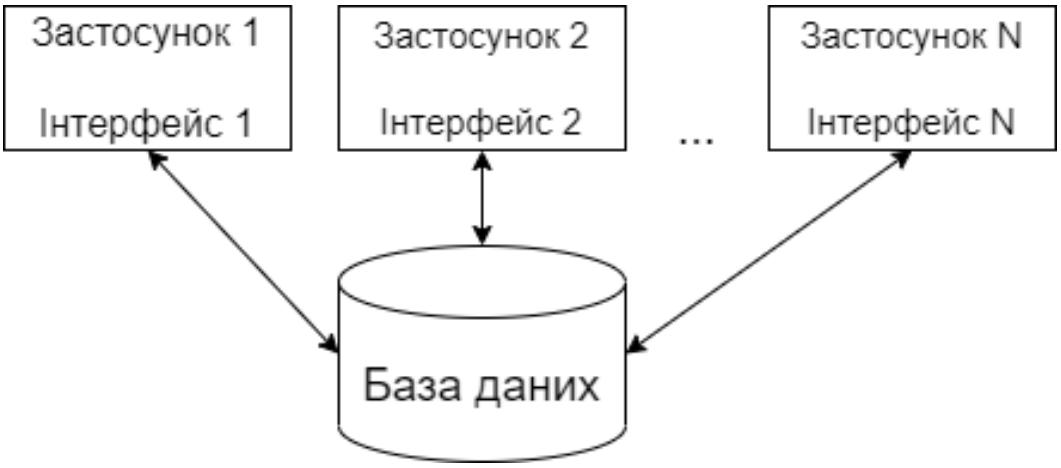


Рис. 10 Схема інтеграції на рівні даних

На рисунку 10 зображена схема інтеграції на рівні даних. Такий підхід дозволяє уникнути обов'язкової модифікації програмної реалізації систем для впровадження інтеграції, оскільки спільна база даних надає узгоджене джерело даних. Необхідність додаткової синхронізації даних між застосунками, що їх використовують, зникає. Формат даних та спосіб доступу визначається системою керування базами даних, але сучасні системи для розробки додатків підтримують різні інструменти доступу до баз даних. Для уникнення конфліктів під час роботи з даними застосовується механізм транзакцій [9].

Недоліком способу інтеграції на рівні даних є необхідність забезпечення всіх потреб застосунків, які використовують єдине джерело даних. Схема бази даних має бути комплексною, загальною та універсальною, отже необхідно підтримувати надлишок типів даних. Ріст бази даних може призвести до збільшення часу виконання запитів, надмірних потреб у пам'яті, підвищення складності розвитку та витрат на обслуговування [10].

2.3.2 Інтеграція за допомогою передачі файлів

Інтеграція за допомогою передачі файлів забезпечується генеруванням вихідною системою файлів з відповідною частотою, які містять дані, потрібні для обробки цільовою системою. Спосіб є кросплатформним, оскільки файли є вбудованим універсальним механізмом зберігання даних, доступ до якого реалізований сучасними системами розробки. Слід підкреслити, що процесу

передачі і читання файлу передує його форматування для забезпечення сумісності. Раніше використовувались формати Cobol системи, текстові, з часом виникли стандарти форматів файлів обміну. Сучасними форматами є XML та JSON.

Рис. 11 Схема інтеграції на основі обміну файлами

На рисунку 11 зображена схема інтеграції на основі обміну файлами. Перевагою файлового обміну є узгоджений формат даних обміну без деталей програмної реалізації їх генерації. Обробка покладається на інтегратори або цільові системи, що дає можливість вносити зміни в процеси формування, читання, маніпулювання без впливу систем одна на одну за умови збереження спільного формату файлу. Оскільки файли є частиною майже будь-якої системи відсутня необхідність додаткових інструментів. Зазвичай визначається графік запису/читання файлів для зменшення навантаження (щогодинно, щоденно).

Однак негативною стороною використання файлів є покладанням на розробників імплементації механізмів обробки файлів. Назви файлів і каталогів мають бути унікальними і скоординованими між системами. Крім того, варто синхронізувати час життя файлів та правил одночасного доступу, щоб запобігти ситуації коли файл вже створений і доступний до звернення, але до нього ще відбувається запис. Відсутність синхронізації може задовольняти користувачів систем, оскільки затримка передачі даних між розподіленими системи зазвичай враховується при проектуванні, але варто брати до уваги необхідність споживачів в оновленні даних при генерації вхідних даних. Чим довший час обміну файлами, тим більшою може стати проблема невідповідності даних. Звісно, можна задати частоту створення файлів досить високою, але в такому разі ресурси на обробку файлів значно збільшуються зі швидкістю їх генерації. Можна застосувати інший

спосіб інтеграції. Для кращого узгодження форматів даних можна використати спільну базу даних, для частого обміну невеликими обсягами даних варто застосувати обмін повідомленнями.

2.3.3 Інтеграція за допомогою обміну повідомленнями

Інтеграція за допомогою обміну файлів та спільної бази даних дозволяє обмін даними, проте не функціями. Часто існує потреба у безпечному обміні також процесами і функціональністю при цьому зберігаючи ізольованість систем. Обмін файлами дозволяє виконувати ці умови, але повільний обмін є вирішальним фактор для відмови від такого способу інтеграції. Інтеграція на рівні бази даних вирішує питання швидкодії, проте всі системи мають високу залежність, що порушує гнучкість розвитку систем та безпеку.

Рис. 12 Інтеграція за допомогою обміну повідомлень

Обмін повідомленнями використовується для надійної швидкодійної асинхронної передачі даних. Деталі реалізації систем приховані, отже зберігається принцип впровадження залежностей. Механізм повторного відправлення дозволяє підвищити рівень відмовостійкості та забезпечити надійність доставки повідомлень. Асинхронний обмін дозволяє не чекати готовності до обміну всіх систем, хоча і робота з розділеними системами повільніша, це підштовхує до розробки частин інтеграції з високою зв’язністю, маючи чітко визначені підсистеми. Трансформація повідомлень може бути виконана під час передачі, що дозволяє відокремити розробку систем та інтеграції між ними. Перетворення

передбачає можливість трансляції повідомлень між кількома системами з різними структурами зберігання і обробки даних. Обмін повідомленнями є достатньо швидким і застосовний для випадків коли час передачі має бути мінімальним (наприклад у фінансових банківських системах).

Системи обміну повідомленнями мають проблеми з затримками, оскільки використовується асинхронний підхід. Відлагодження, валідація та тестування інтеграції мають додаткову складність. Можливість перетворення повідомлень за межами вихідних та цільових систем призводить до розробки додаткових рішень, що оброблятимуть модифіковані дані. Також можуть з’являтись додаткові питання, як-от механізми передачі даних, допустимі формати, адресатів, що для вирішення потенційно потребуватимуть розробки додаткових рішень.

2.3.4 Інтеграція за допомогою виклику віддалених процедур

Зазвичай, кожна система розробляється окремо, використовуючи різні платформи та технології, часто не пов’язані одна з одною. В той же час, користувачі систем можуть потребувати їх взаємодії між собою у гнучкий спосіб. Підхід RPC (Remote Procedure Call) дозволяє викликати окрему функцію в іншому застосунку, передаючи необхідні параметри з вказанням як саме їх треба обробити. Виклик віддаленої процедури дозволяють визначені інтерфейси (функції) [11].

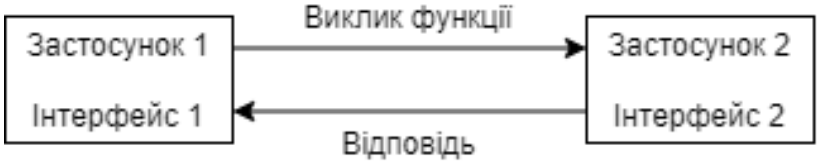


Рис. 13 Інтеграція за допомогою виклику віддалених процедур

Системи, що взаємодіють, можуть бути реалізовані як окремі компоненти без порушення принципу інкапсуляції. Якщо програма потребує певних даних, відбувається безпосередній виклик процедури, що дозволяє зберігати цілісність даних. Наразі популярною реалізацією інтеграції за допомогою виклику віддалених процедур є веб-сервіси, які використовують стандарти SOAP або

REST. Зручною особливістю є робота з протоколом HTTP, що забезпечує уніфікований спосіб роботи з даними. Системи можуть надавати кілька інтерфейсів для роботи з наборами даних для різних користувачів. Така можливість надає варіанти гнучкої підтримки. Підхід REST широко застосовується і використовує методи запитів, заголовки та коди відповідей стандарту HTTP. Зазвичай використовується формат тіла повідомлення XML чи JSON [12].

Одночасно з тим, варто пам’ятати про узгодження інтерфейсів. Послідовність виклику віддалених процедур може збільшувати залежність систем між собою та ускладнювати оновлення та зміну систем.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ

3.1 Фази процесу міграції даних

Процес міграції даних є комплексною задачею і потребує розробки детального плану. Час, що витрачається на кожну з фаз залежить від складності системи та інструментів що використовуються. Деякі кроки можуть виконуватись паралельно, бути не послідовними і повторюватись. Можна виділити наступні основні фази міграції бази даних:

- Резервне копіювання даних
- Планування, аналіз вихідної та цільової систем
- Виконання міграції
- Валідація даних в цільовій системі

3.2 Резервне копіювання даних

Резервне копіювання є важливою частиною процесу міграції, хоча і не обов'язковою. Наявність резервних копій дозволяє надійно та безпечно запобігати втраті даних. Помилки можуть ставатись через користувацькі дії, несправність апаратного забезпечення чи інші зовнішні фактори. Резервні копії також дозволяють переносити дані між різними серверами.

3.3.1 Утиліта mysqldump

Для вихідної системи керування базами даних розробником передбачений інструмент створення резервних копій mysqldump.

mysqldump є програмою командного рядка і дозволяє створювати логічні резервні копії. Формат збереження резервної копії – набір запитів SQL, які необхідно виконати якщо потрібно відтворити дані та структуру бази даних. Також можливо налаштувати вивід у XML форматі, CSV або іншими роздільниками.

```
C:\Users\Iryna>mysqldump --user=root --password=[redacted] --all-databases > D:\Uni\dump.sql
```

Рис. 14 Приклад використання програми mysqldump

На рисунку 14 можна побачити приклад використання mysqldump для створення резервної копії всіх баз даних, що розташовані на поточному сервері. Mysqldump дозволяє створювати резервні копії одної або кількох баз даних для їх перенесення на інший сервер [13].

Имя	Размер	Сжат	Тип	Изменён	CRC32
..			Папка с файлами		
newsEditors.sql	9 400	?	Microsoft SQL Server Query File	03.03.2023 10:16	
fresh.sql	242 950 470	?	Microsoft SQL Server Query File	03.03.2023 10:16	

Рис. 15 Резервні копії баз даних продуктового серверу з розміром

Після створення застосування утиліти на продуктовому сервері маємо два файли з розширенням SQL з відповідними розмірами, кожен з яких являє собою резервну копію даних.

```
1   -- MySQL dump 10.13  Distrib 5.5.47, for debian-linux-gnu (x86_64)
2   --
3   -- Host: localhost    Database: fresh
4   -- ------------------------------------------------------
5   -- Server version   5.5.47-0ubuntu0.12.04.1
6
7   /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
8   /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
9   /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
10  /*!40101 SET NAMES utf8 */;
11  /*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
12  /*!40103 SET TIME_ZONE='+00:00' */;
13  /*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
14  /*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
15  /*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
16  /*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
```

Рис. 16 Заголовок файлу резервної копії

На рисунку 16 можна побачити заголовок файлу резервної копії, що містить версію утиліти, версію MySQL серверу, назву серверу де розгорнуто базу даних та назву бази даних, а також набір параметрів, як-от кодування та часовий пояс який треба використовувати.

```
169     --
170     -- Table structure for table `history`
171     --
172
173 •   DROP TABLE IF EXISTS `history`;
174 •   /*!40101 SET @saved_cs_client     = @@character_set_client */;
175 •   /*!40101 SET character_set_client = utf8 */;
176 •   CREATE TABLE `history` (
177       `historyID` int(10) NOT NULL AUTO_INCREMENT,
178       `userID` varchar(128) COLLATE cp1251_ukrainian_ci NOT NULL,
179       `newsID` int(10) NOT NULL,
180       `newsHistoryID` int(10) DEFAULT NULL,
181       `date` int(10) NOT NULL,
182       `userActionID` int(10) NOT NULL,
183       PRIMARY KEY (`historyID`),
184       KEY `userID_idx` (`userID`),
185       KEY `newsID_idx` (`newsID`),
186       KEY `news_oldID_idx` (`newsHistoryID`)
187     ) ENGINE=InnoDB AUTO_INCREMENT=31478 DEFAULT CHARSET=cp1251 COLLATE=cp1251_ukrainian_ci;
188 •   /*!40101 SET character_set_client = @saved_cs_client */;
```

Рис. 17 Приклад формату зберігання структури таблиці

На рисунку 17 можна побачити згенерований запит для відтворення структури таблиці history з відповідними типами даних, набором символів, обмеженнями у вигляді первинного та зовнішніх ключів та відношеннями з іншими таблицями.

```
189
190     --
191     -- Dumping data for table `history`
192     --
193
194 •   LOCK TABLES `history` WRITE;
195 •   /*!40000 ALTER TABLE `history` DISABLE KEYS */;
196 •   INSERT INTO `history` VALUES (360,'74e0a82b-1574-42e0-831a-4d3f7fcf1b73',7872,NULL,1467345391,4),(361,'74e0a82b-1574-42e0
197 •   INSERT INTO `history` VALUES (15137,'bfb8c160-bd8b-4998-a0d0-67179a8c4341',10442,NULL,1559128114,1),(15138,'a3bd1302-1043
198 •   INSERT INTO `history` VALUES (29562,'bfb8c160-bd8b-4998-a0d0-67179a8c4341',12357,NULL,1662717094,1),(29563,'a3bd1302-1043
199 •   /*!40000 ALTER TABLE `history` ENABLE KEYS */;
200 •   UNLOCK TABLES;
```

Рис. 18 Приклад запитів, що завантажують дані

На рисунку 18 зображений приклад запитів завантаження даних з резервної копії у таблицю history.

3.2 Планування

3.2.2 Система MySQL Workbench

MySQL Workbench дозволяє забезпечити роботу з MySQL Server.

MySQL Workbench – це система, призначена для роботи з системою керування реляційними базами даних MySQL версії 5.5 і вище, і надає графічний користувацький інтерфейс, що зображений на рис 19.

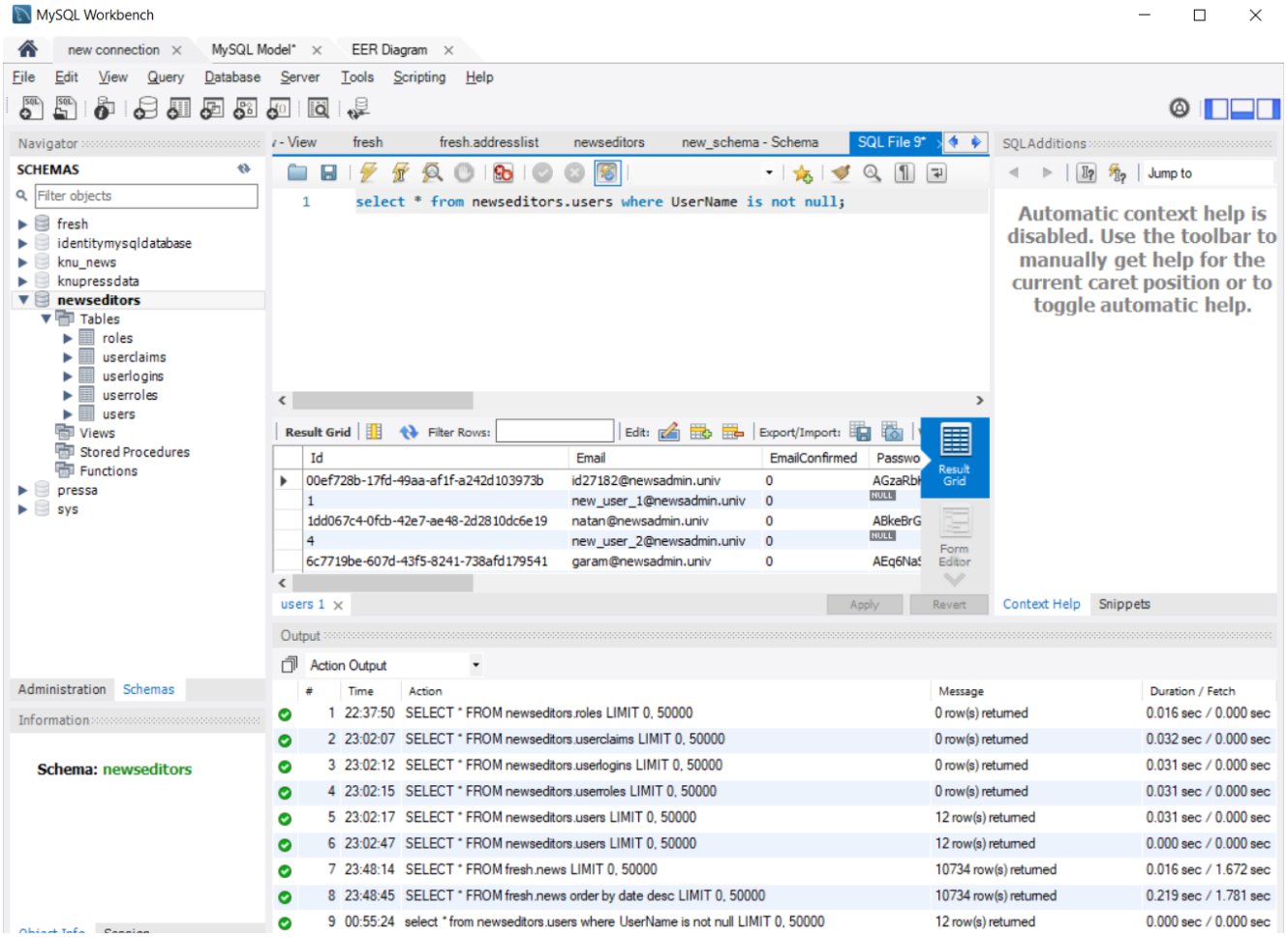


Рис. 19 Користувацький інтерфейс системи MySQL Workbench

З рисунку 19 можна побачити, що MySQL Workbench надає наступні основні можливості: працювати з SQL-запитами для серверів та баз даних, створювати бази даних та їх структури за допомогою графічного інтерфейсу, адмініструвати сервери баз даних та надає інструмент для міграції на MySQL з інших СКБД. MySQL Workbench дозволяє замінити застосунок командного рядка для роботи з СКБД MySQL [14].

3.2.3 Система Microsoft SQL Server Management Studio

Для взаємодії з Microsoft SQL Server було обрано середовище SQL Server Management Studio. SSMS – це система, що надає користувацький інтерфейс для керування, налаштування, доступу та адміністрування Microsoft SQL Server, що можна побачити на рисунку 20.

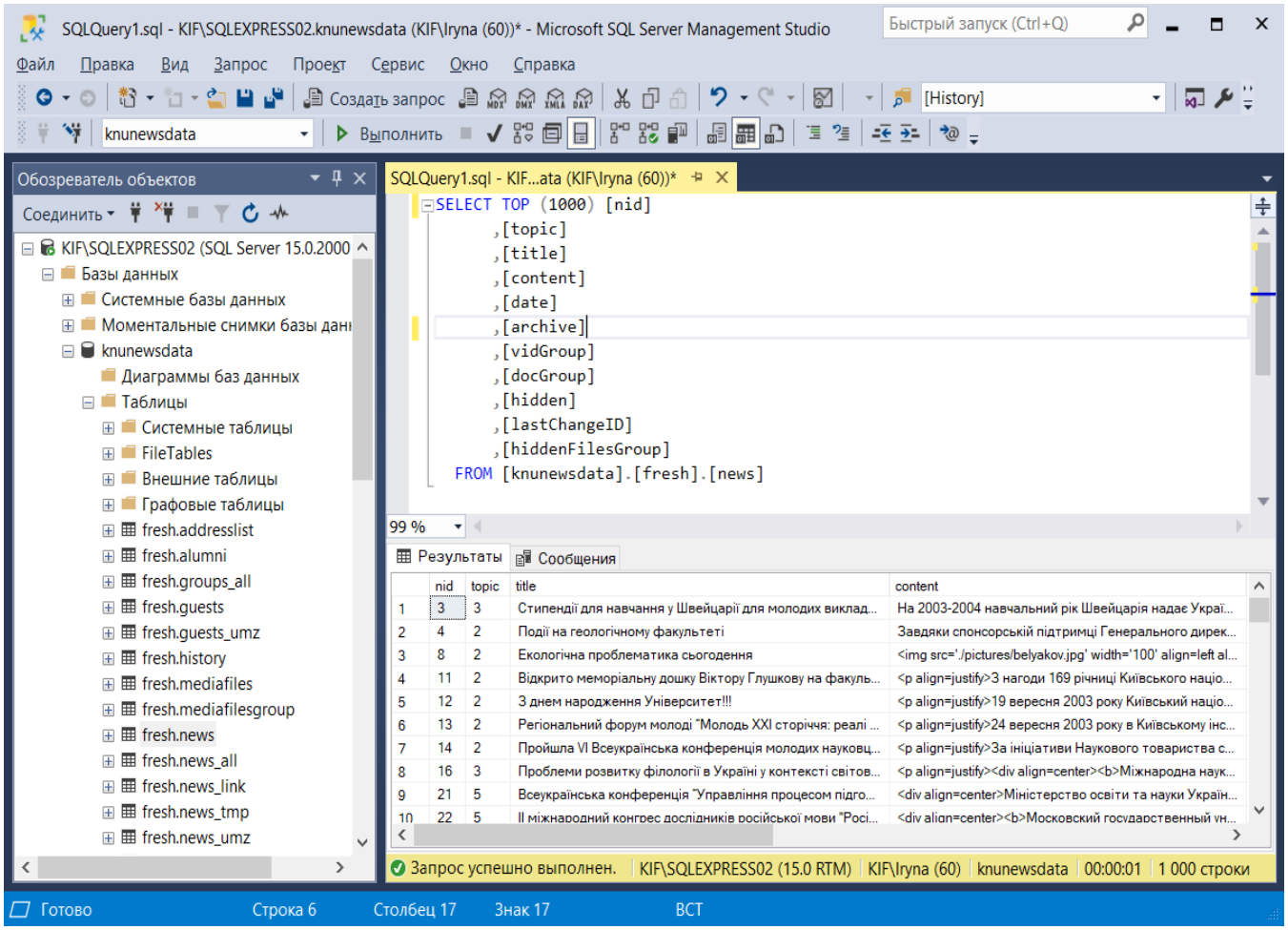


Рис. 20 Середовище SQL Server Management Studio

З рисунку 20 також можна побачити, що крім графічного інтерфейсу для взаємодії з системою керування базами даних, інтегроване середовище надає редактор запитів для роботи з Microsoft SQL Server.

3.2.4 Система Microsoft SQL Server Migration Assistant for MySQL

Для забезпечення ETL-процесу міграції даних обрано застосунок Microsoft SQL Server Migration Assistant for MySQL (далі SSMA), що є автоматизованим

засобом міграції даних і дозволяє виконати її з СКБД MySQL на Microsoft SQL Server. Графічний інтерфейс можна побачити на рисунку 21.

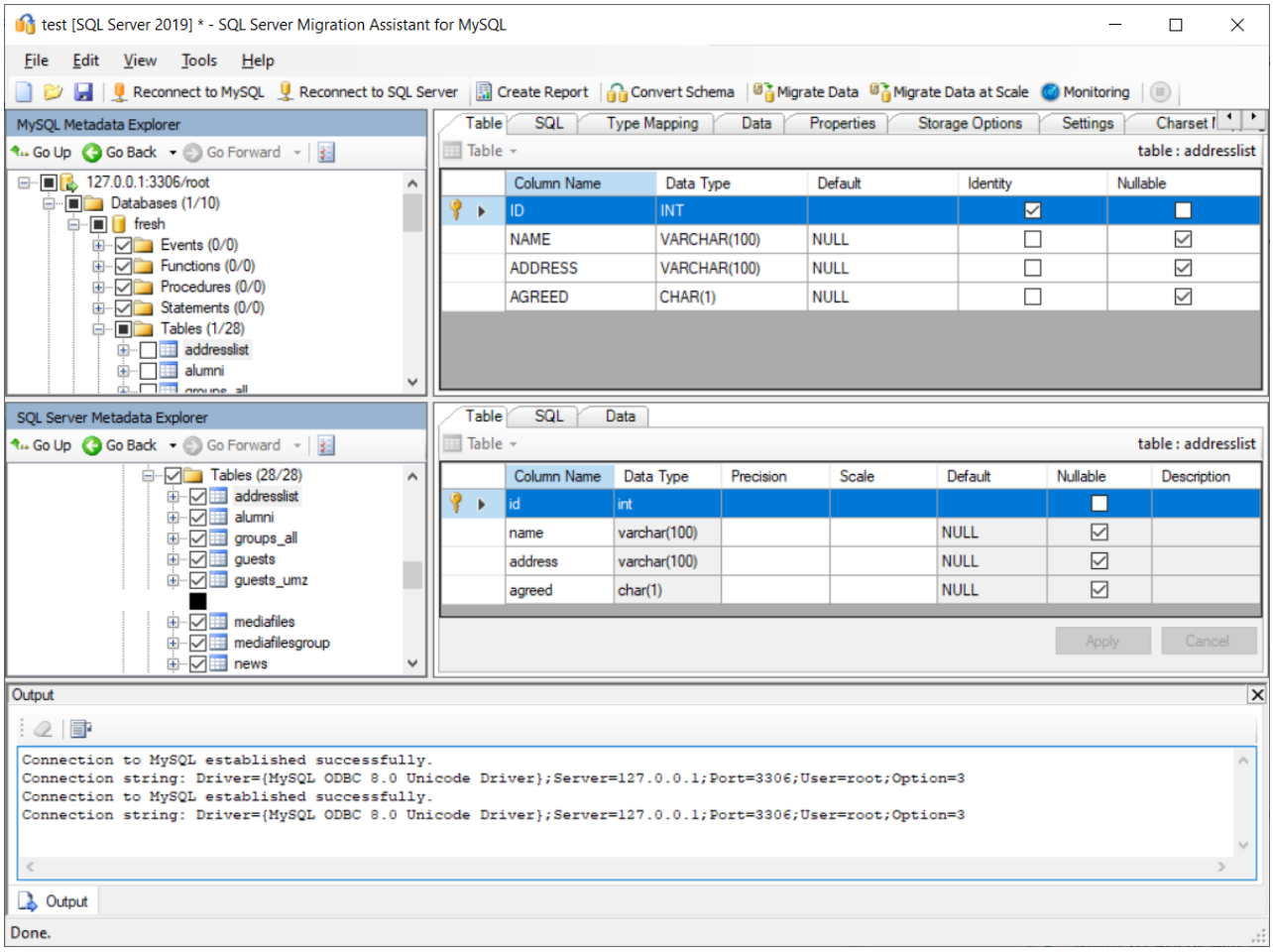


Рис. 21 Інтерфейс Microsoft SQL Server Migration Assistant for MySQL

З рисунку 21 можна побачити засоби автоматизації аналізу вихідної системи (Create Report), міграції структури бази даних (Convert Schema) та власне міграції даних (Migrate Data). Перевагою цього засобу є можливість вибору версій цільової системи та конекторів які підтримують різні кодування даних. Це є важливим і основним критерієм при виборі інструменту, оскільки дані зберігаються українською мовою і потребують збереження повнити після зміни СКБД. Варто зазначити, що для підключення до MySQL використовується MySQL ODBC 8.0 Unicode Driver (v 8.0.32).

3.2.5 Аналіз вихідної системи

Як було зазначено раніше, в якості вихідної системи керування базами

даних використовується MySQL. Загальну структуру бази даних fresh, структуру таблиць, типи даних, які використовуються, та зв’язків між сутностями, визначеними для зберігання даних процесу управління публікації новини можна побачити на рисунку 7 розділу 1.

Таблиця News відображає сутність Новина з темою, заголовком, змістом, заголовком і змістом англійською та російською мовою, тизером, групами медіа-файлів, прапором прихована/опублікована, остання зміна. Варто зазначити що зміст новин (поле Content) зберігається у форматі HTML-розмітки, що можна побачити на рисунку 22.

Рис. 22 Формат збереження новини в базі даних

На рисунку 22 зображений формат зберігання новини в базі даних. HTML-розмітка зберігається у полі з типом даних text.

Таблиця NewsHistory містить всі зміни, що відбувались з новиною. Таблиця History слугує для збереження відношення між користувачем, його дією, новиною, та записом в історії змін. Таблиця mediafiles містить назву, опис, посилання на файл та зовнішній ключ на таблицю mediafilesgroup, що вказує тип розміщення та дизайн медіафайлу.

Проте, варто зазначити що база даних також містить історичні таблиці, зображені на рис. 23, що не використовуються в панелі керування новинами, але оскільки вони містять дані, до видалення вони не підлягають.

Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length	Max Data Length	Index Length	Dat
addresslist	MyISAM	10	Dynamic	198	42	8.2 KiB	256.0 TiB	4.0 KiB	
alumni	MyISAM	10	Dynamic	32	144	4.5 KiB	256.0 TiB	2.0 KiB	
groups_all	MyISAM	10	Dynamic	2	20	40.0 bytes	256.0 TiB	3.0 KiB	
guests	MyISAM	10	Dynamic	134	236	31.0 KiB	256.0 TiB	4.0 KiB	
guests_umz	MyISAM	10	Dynamic	513	197	98.8 KiB	256.0 TiB	9.0 KiB	
history	InnoDB	10	Dynamic	31238	84	2.5 MiB	0.0 bytes	4.5 MiB	
mediafiles	InnoDB	10	Dynamic	31934	115	3.5 MiB	0.0 bytes	0.0 bytes	
mediafilesgroup	InnoDB	10	Dynamic	4588	39	176.0 KiB	0.0 bytes	0.0 bytes	
news	MyISAM	10	Dynamic	10734	7362	75.4 MiB	256.0 TiB	26.1 MiB	
news_all	MyISAM	10	Dynamic	1	548	548.0 bytes	256.0 TiB	10.0 KiB	
news_link	MyISAM	10	Fixed	1	10	10.0 bytes	2560.0 TiB	5.0 KiB	
news_tmp	MyISAM	10	Dynamic	132	2932	378.0 KiB	256.0 TiB	370.0 KiB	
news_umz	MyISAM	10	Dynamic	5	308	1.5 KiB	256.0 TiB	2.0 KiB	
newshistory	MyISAM	10	Dynamic	15683	9242	138.2 MiB	256.0 TiB	34.7 MiB	
photo_series_all	MyISAM	10	Dynamic	2	20	40.0 bytes	256.0 TiB	2.0 KiB	
photo_topics_all	MyISAM	10	Dynamic	1	28	28.0 bytes	256.0 TiB	2.0 KiB	
photos	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TiB	1.0 KiB	
photos_all	MyISAM	10	Dynamic	3	501156	1.4 MiB	256.0 TiB	8.0 KiB	
photos_link	MyISAM	10	Fixed	3	14	42.0 bytes	3584.0 TiB	6.0 KiB	
polls_all	MyISAM	10	Dynamic	1	268	268.0 bytes	256.0 TiB	2.0 KiB	
stats_day	MyISAM	10	Fixed	2	7	14.0 bytes	1792.0 TiB	2.0 KiB	
topics_all	MyISAM	10	Dynamic	4	26	104.0 bytes	256.0 TiB	2.0 KiB	
umz	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TiB	1.0 KiB	
umz_topics_all	MyISAM	10	Dynamic	12	25	300.0 bytes	256.0 TiB	2.0 KiB	
umzguest	MyISAM	10	Dynamic	8	137	1.1 KiB	256.0 TiB	2.0 KiB	
umzphoto	MyISAM	10	Dynamic	29	20	580.0 bytes	256.0 TiB	2.0 KiB	
users_all	MyISAM	10	Dynamic	3	50	152.0 bytes	256.0 TiB	7.0 KiB	
votes_all	MyISAM	10	Dynamic	6	42	256.0 bytes	256.0 TiB	2.0 KiB	

Рис. 23 Список таблиць бази даних fresh

На рисунку 23 зображений повний список таблиць бази даних з назвою таблиці, кількістю рядків, розміром даних, що потребують міграції. Всього міститься 28 таблиць.

На рисунку 8 зображена схема сутностей та зв’язків бази даних newseditors для керування користувачами панелі. Така структура таблиць є частиною технології ASP.NET Identity для .NET Framework.

База даних містить структуру, зображену на рис. 3 розділу 1. Даний набір таблиць є автозгенерованим. Хоча для даних таблиць технологія ASP.NET Identity передбачає набір функцій, як-от керування ролями, раніше він не використовувався, що можна побачити на рисунку 24.

Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length	Max Data Length	Index Length
roles	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	0.0 bytes
userclaims	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	32.0 KiB
userlogins	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	16.0 KiB
userroles	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	16.0 KiB
users	InnoDB	10	Dynamic	12	1365	16.0 KiB	0.0 bytes	0.0 bytes

Рис. 24 Вміст таблиць бази даних newseditors

Як можна побачити з рисунку 24, всі таблиць мають порожню множину значень, окрім таблиці користувачів users.

3.2.6 Аналіз цільової системи

Для збереження об’єктів панелі управління публікацією новин використовується Microsoft SQL Server (версія 2019) – система керування базами даних, що використовує для організації баз даних реляційну модель. Структури даних, що відповідають за збереження предметно області мають залишитись незмінними задля забезпечення цілісності даних. Як було зазначено, під час процесу модернізації було оновлено систему керування користувачами і впроваджено використання ASP.NET Identity Core, що має додаткові можливості налаштування безпеки і тому згенерована структура відрізняється від вихідної структури бази даних, що зображено на рисунку 25.

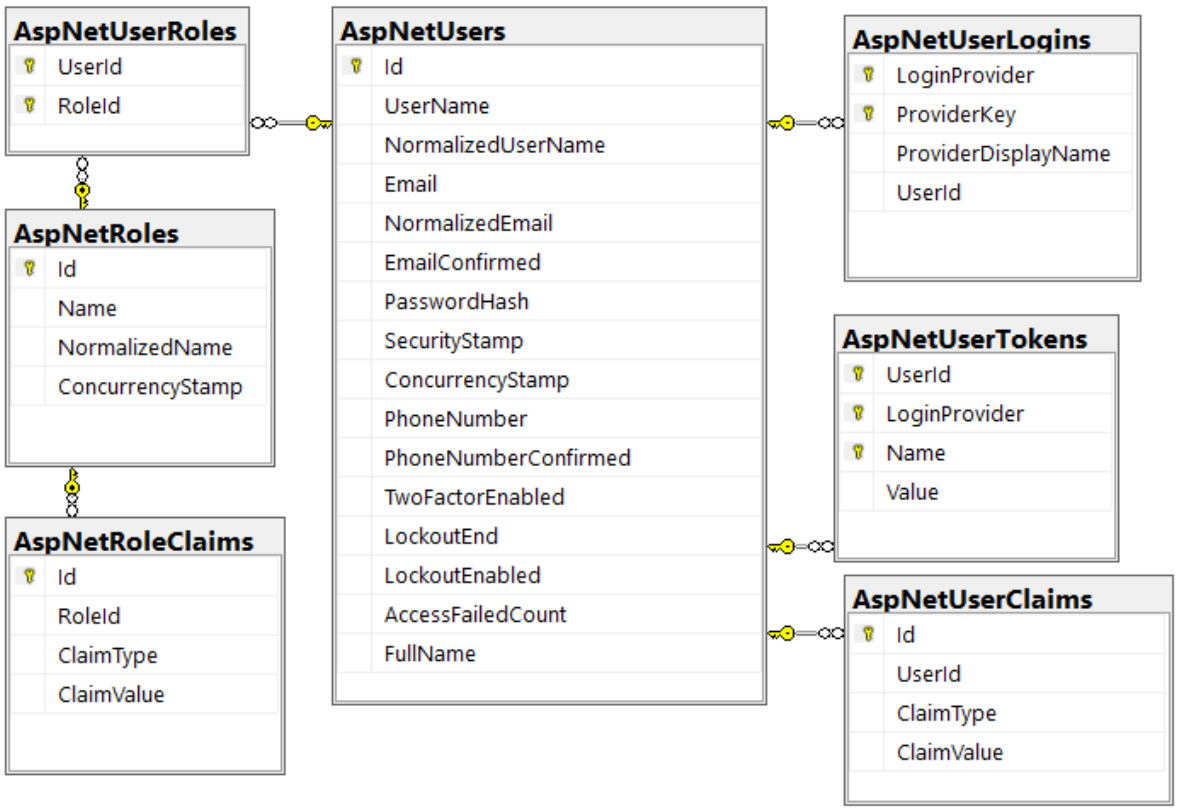


Рис. 25 Структура збереження даних користувачів модернізованої панелі адміністрування

З рисунку 25 можна побачити, що основна таблиця AspNetUsers має набір

додаткових полів NormalizedUserName, NormalizedEmail, ConcurrencyStamp та прибрані всі поля, що не використовувались системою. Оскільки структури даних не відповідають одна одній, для цієї таблиці має застосовуватись додаткова трансформація під час міграції системи керування базами даних.

3.2.7 Оцінка об’єктів вихідної бази даних

Для оцінки міграції використано звіт SSMA. Зміст містить статистичні дані про перетворення, попередження та перетворення, які необхідно виконати для міграції структури таблиці. Для прикладу розглянемо таблицю mediafiles, зображену на рисунку 26. Звіт з інших таблиць містить аналогічну структуру і доданий в додатку 1.

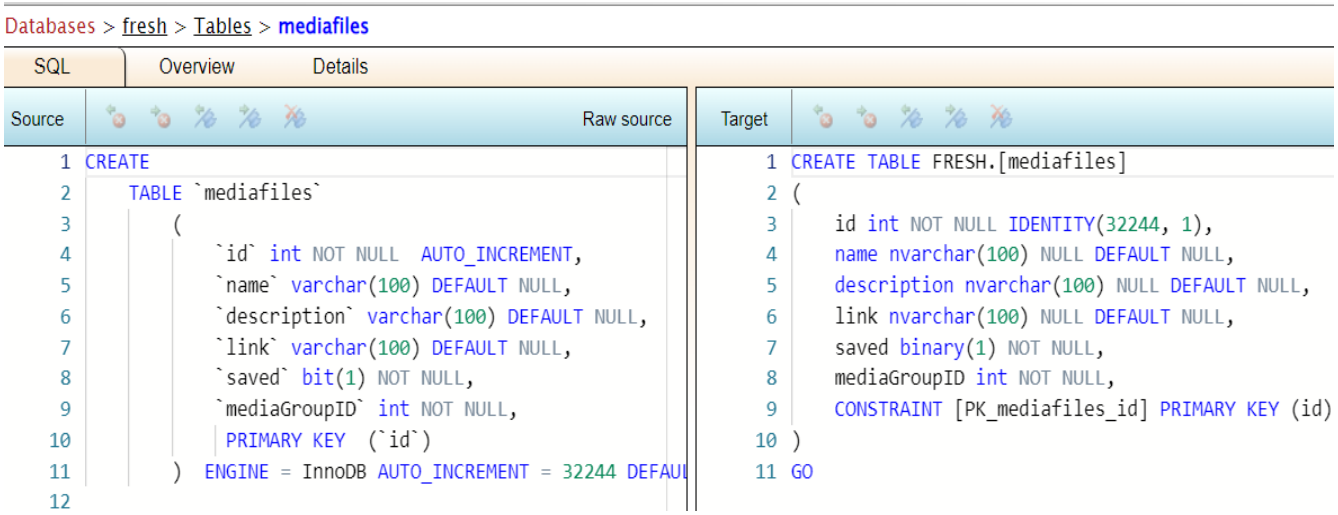
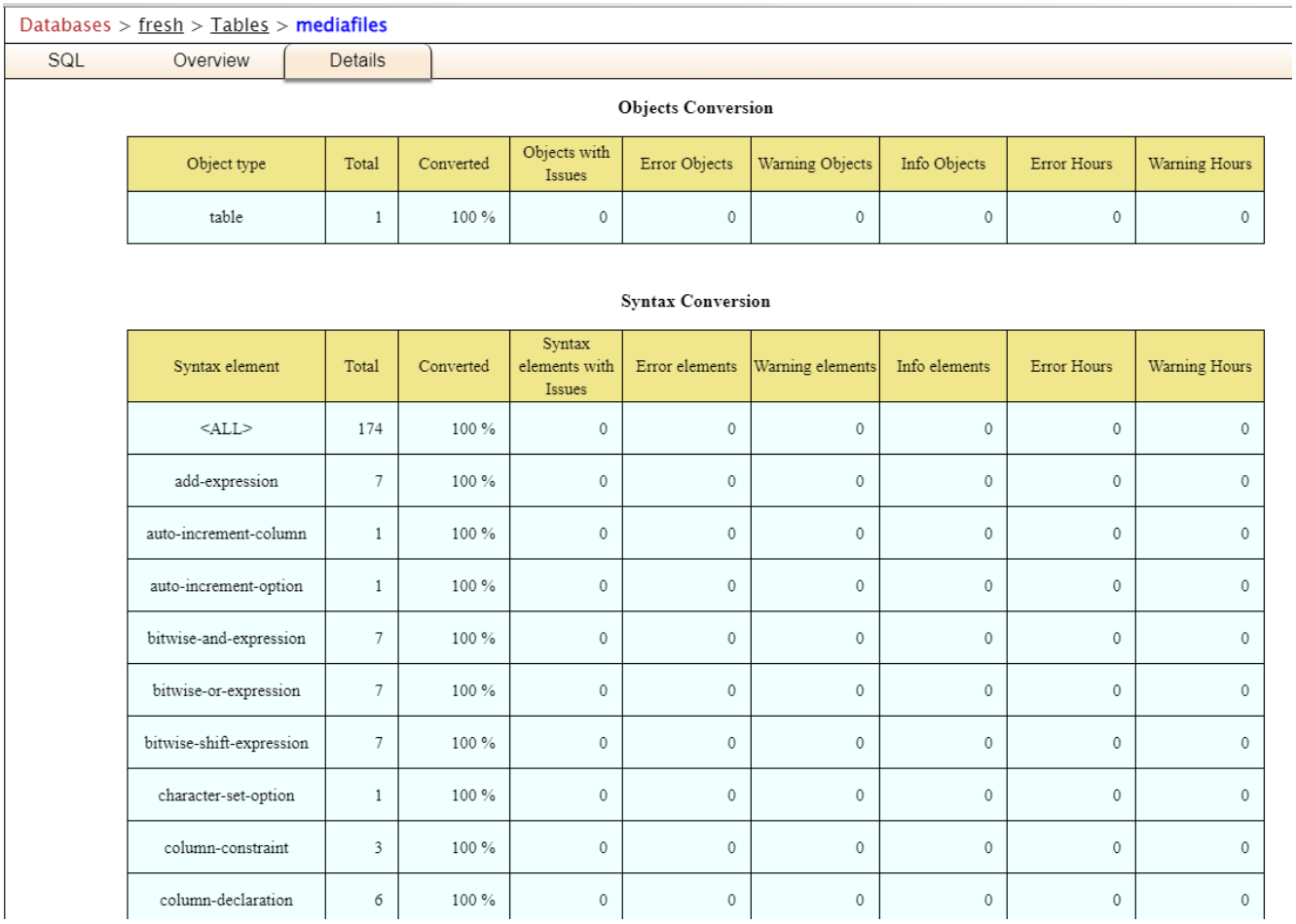


Рис. 26 Відтворення структури таблиці mediafiles

На рисунку 26 зображено запит для відтворення структури таблиці mediafiles. Запит містить відповідну назву таблиці, набір полів з типами даних, що відповідатимуть вихідним у цільовій системі, та набір обмежень що застосовуються в таблиці (в даному випадку обмеження – унікальний первинний ключ таблиці mediafiles id). Запит призначений для попередньої оцінки відповідності типів даних і їх модифікації, якщо визначені автоматично не задовольняють умови, оскільки типи даних в вихідній та цільовій СКБД відрізняються. Наприклад, типу даних varchar в MySQL Server відповідає nvarchar в Microsoft SQL server. Типи даних, що будуть застосовані, відповідають

очікуваним.

Звіт також містить детальну таблицю зі статистикою можливості перетворення структури таблиці, її об’єктів та даних, що можна побачити з рисунку 27.



Databases > fresh > Tables > mediafiles

SQL | Overview | Details

Objects Conversion

Object type	Total	Converted	Objects with Issues	Error Objects	Warning Objects	Info Objects	Error Hours	Warning Hours
table	1	100 %	0	0	0	0	0	0

Syntax Conversion

Syntax element	Total	Converted	Syntax elements with Issues	Error elements	Warning elements	Info elements	Error Hours	Warning Hours
<ALL>	174	100 %	0	0	0	0	0	0
add-expression	7	100 %	0	0	0	0	0	0
auto-increment-column	1	100 %	0	0	0	0	0	0
auto-increment-option	1	100 %	0	0	0	0	0	0
bitwise-and-expression	7	100 %	0	0	0	0	0	0
bitwise-or-expression	7	100 %	0	0	0	0	0	0
bitwise-shift-expression	7	100 %	0	0	0	0	0	0
character-set-option	1	100 %	0	0	0	0	0	0
column-constraint	3	100 %	0	0	0	0	0	0
column-declaration	6	100 %	0	0	0	0	0	0

Рис. 27 Таблиця перетворення об’єктів таблиці mediafiles

Як видно з рисунку 27, структура таблиці mediafiles може бути мігрована на 100%.

3.3 Перетворення структури бази даних

Наступним кроком підготовки до міграції даних є відтворення структури бази даних в цільовій системі. Процес перетворення об’єктів: MySQL визначення об’єктів трансформуються у відповідні об’єкти SQL Server. Для контролю за даним процесом зберігаються і завантажуються метадані, які можна переглянути в користувацькому інтерфейсі SSMA на рисунку 28.

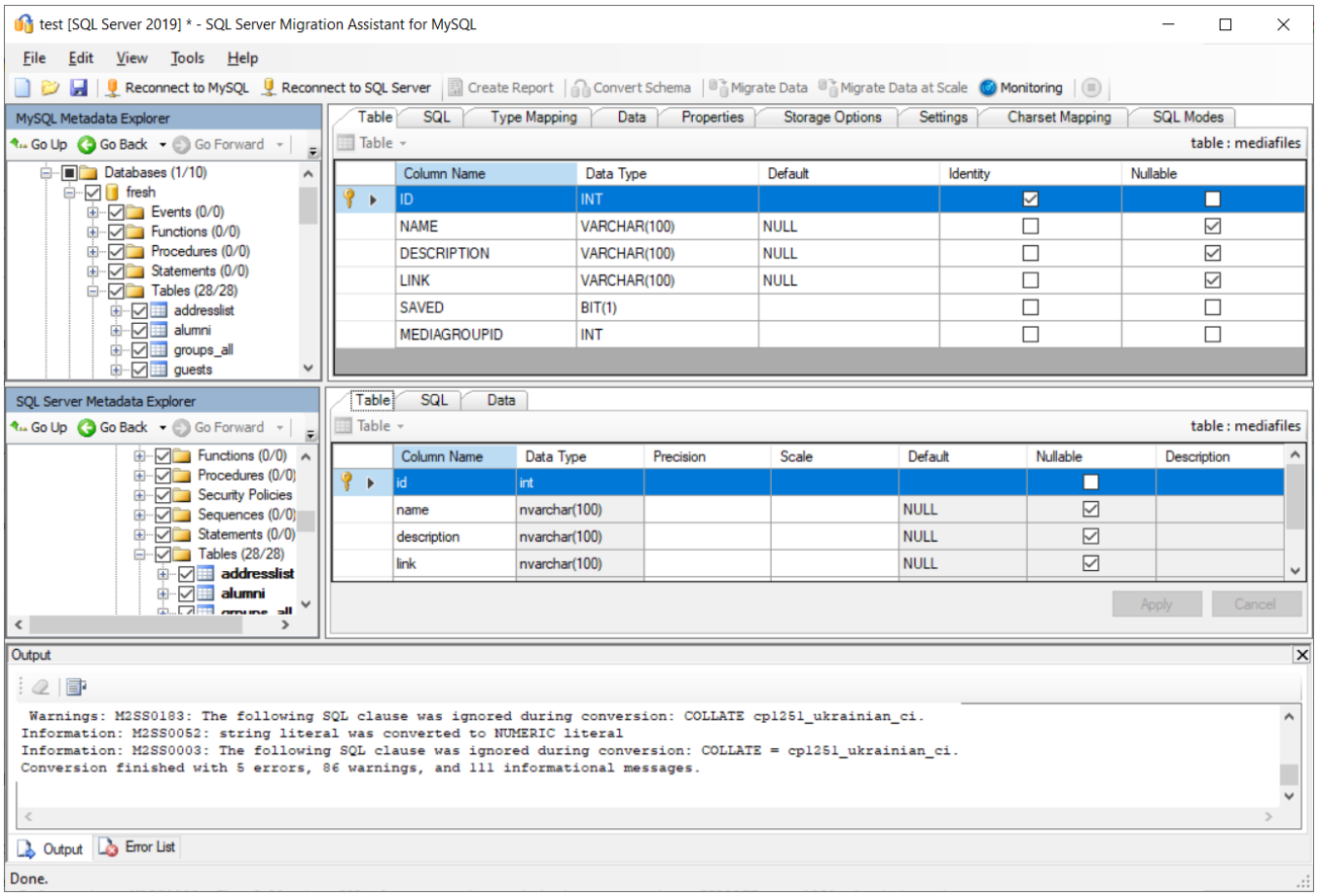


Рис. 28 Користувацький інтерфейсі SSMA

На рисунку 28 можна побачити повідомлення у області Output, яке містить перелік попереджень, помилок та рекомендацій, які дозволять визначити, чи потребуються зміни в процесі перетворення структури даних. Також можна побачити відповідність структур вихідної та цільової баз даних з накладеними обмеженнями. У випадку невідповідності необхідно провести додаткові налаштування та конвертувати структуру БД знову [15]. На рисунку зображено приклад таблиці mediafiles. Як можна побачити на прикладі, запит у звіті та конвертована схема ідентичні. Детальні SQL-запити, які виконувались для конвертації схеми бази даних можна побачити в додатку 2.

3.4 Виконання міграції даних

Основним етапом процесу міграції даних є власне виконання міграції після детального аналізу та конвертування структури баз даних.

Після конвертації схеми бази даних перед безпосередньою міграцією даних необхідно синхронізувати схему з базою даних. Перед виконанням операції SSMA пропонує верифікувати об’єкти, які будуть змінені, що зображено на рисунку 29.

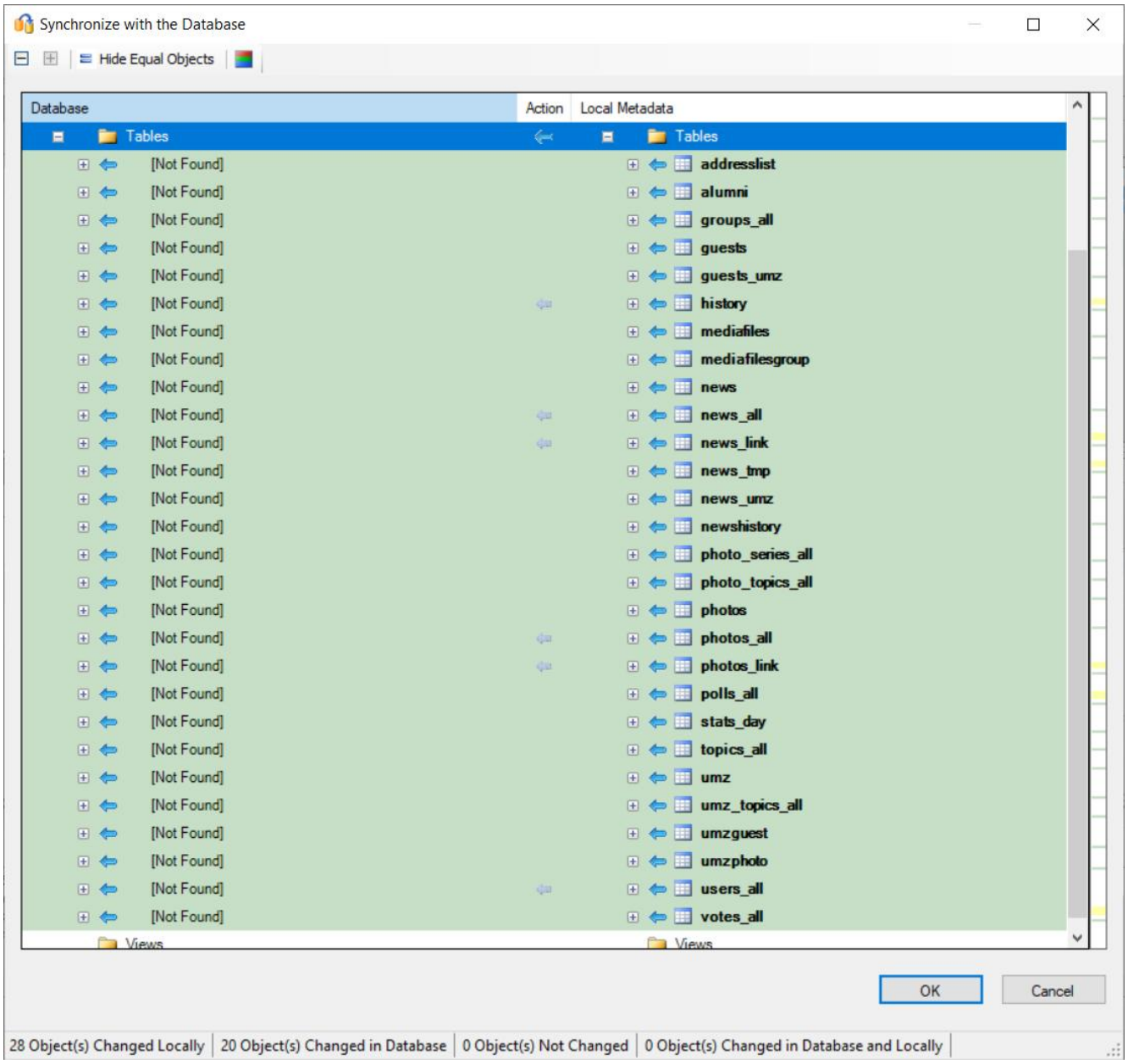
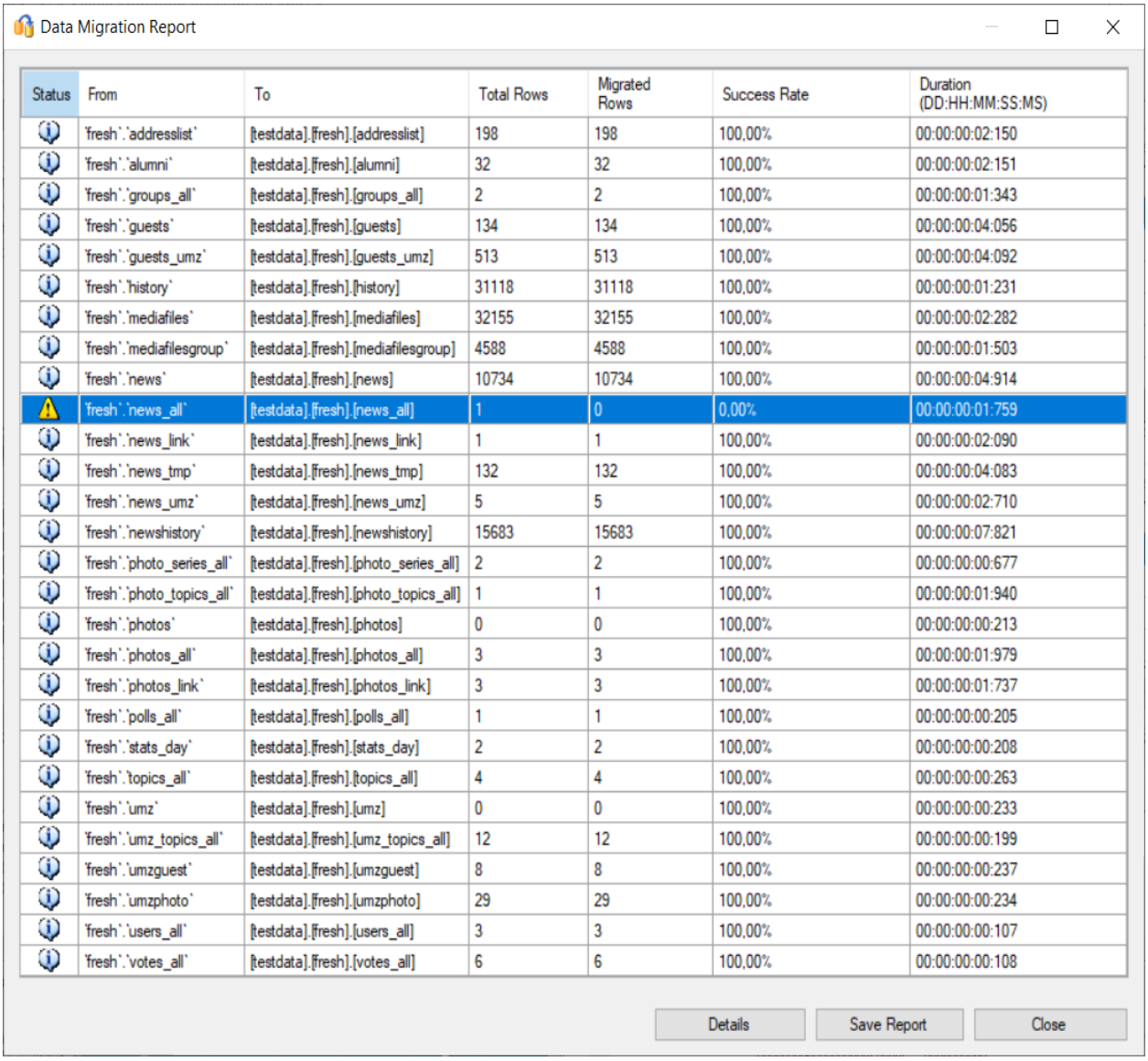


Рис. 29 Список об’єктів що підлягають змінам в базі даних

На рисунку 29 наявний список об’єктів, які будуть змінені в базі даних. Зеленим кольором виділені об’єкти, які будуть створені в базі даних. Всього 28 об’єктів.

Таблицю зі звітом за результатами виконаної міграції бази даних, що

містить сутності керування новинами, можна побачити на рисунку 30.



Status	From	To	Total Rows	Migrated Rows	Success Rate	Duration (DD:HH:MM:SS:MS)
	`fresh`.`addresslist`	[testdata].[fresh].[addresslist]	198	198	100,00%	00:00:00:02:150
	`fresh`.`alumni`	[testdata].[fresh].[alumni]	32	32	100,00%	00:00:00:02:151
	`fresh`.`groups_all`	[testdata].[fresh].[groups_all]	2	2	100,00%	00:00:00:01:343
	`fresh`.`guests`	[testdata].[fresh].[guests]	134	134	100,00%	00:00:00:04:056
	`fresh`.`guests_umz`	[testdata].[fresh].[guests_umz]	513	513	100,00%	00:00:00:04:092
	`fresh`.`history`	[testdata].[fresh].[history]	31118	31118	100,00%	00:00:00:01:231
	`fresh`.`mediafiles`	[testdata].[fresh].[mediafiles]	32155	32155	100,00%	00:00:00:02:282
	`fresh`.`mediafilesgroup`	[testdata].[fresh].[mediafilesgroup]	4588	4588	100,00%	00:00:00:01:503
	`fresh`.`news`	[testdata].[fresh].[news]	10734	10734	100,00%	00:00:00:04:914
⚠	`fresh`.`news_all`	[testdata].[fresh].[news_all]	1	0	0,00%	00:00:00:01:759
	`fresh`.`news_link`	[testdata].[fresh].[news_link]	1	1	100,00%	00:00:00:02:090
	`fresh`.`news_tmp`	[testdata].[fresh].[news_tmp]	132	132	100,00%	00:00:00:04:083
	`fresh`.`news_umz`	[testdata].[fresh].[news_umz]	5	5	100,00%	00:00:00:02:710
	`fresh`.`newshistory`	[testdata].[fresh].[newshistory]	15683	15683	100,00%	00:00:00:07:821
	`fresh`.`photo_series_all`	[testdata].[fresh].[photo_series_all]	2	2	100,00%	00:00:00:00:677
	`fresh`.`photo_topics_all`	[testdata].[fresh].[photo_topics_all]	1	1	100,00%	00:00:00:01:940
	`fresh`.`photos`	[testdata].[fresh].[photos]	0	0	100,00%	00:00:00:00:213
	`fresh`.`photos_all`	[testdata].[fresh].[photos_all]	3	3	100,00%	00:00:00:01:979
	`fresh`.`photos_link`	[testdata].[fresh].[photos_link]	3	3	100,00%	00:00:00:01:737
	`fresh`.`polls_all`	[testdata].[fresh].[polls_all]	1	1	100,00%	00:00:00:00:205
	`fresh`.`stats_day`	[testdata].[fresh].[stats_day]	2	2	100,00%	00:00:00:00:208
	`fresh`.`topics_all`	[testdata].[fresh].[topics_all]	4	4	100,00%	00:00:00:00:263
	`fresh`.`umz`	[testdata].[fresh].[umz]	0	0	100,00%	00:00:00:00:233
	`fresh`.`umz_topics_all`	[testdata].[fresh].[umz_topics_all]	12	12	100,00%	00:00:00:00:199
	`fresh`.`umzguest`	[testdata].[fresh].[umzguest]	8	8	100,00%	00:00:00:00:237
	`fresh`.`umzphoto`	[testdata].[fresh].[umzphoto]	29	29	100,00%	00:00:00:00:234
	`fresh`.`users_all`	[testdata].[fresh].[users_all]	3	3	100,00%	00:00:00:00:107
	`fresh`.`votes_all`	[testdata].[fresh].[votes_all]	6	6	100,00%	00:00:00:00:108

Рис. 30 Звіт виконання міграції даних бази даних fresh

Звіт містить статус, назву таблиці, цільову таблицю, загальну кількість рядків у вихідній таблиці, кількість мігрованих рядків, відсоток успішної міграції та час, витрачений на міграцію таблиці. З рисунку 30 видно, що дані зі всіх таблиць було мігровано без помилок з відсотком виконання 100%, окрім таблиці news_all, яка містить 1 рядок. Для перевірки вмісту таблиці було виконано наступний запит: SELECT * FROM fresh.news_all; Результат зображено на рисунку 31.

	news_id	news_name	news_text	news_date	news_edit	news_view
▶	1	Перша новина	BLOB	1058273940	0000-00-00 00:00:00	0
*	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 31 Результат виконання запиту

З рисунку 31 можна побачити, що таблиця містить 1 рядок. Таблиця не використовується в застосунку панелі адміністрування і є історичними даними, що не містять корисної інформації, тому вирішено не зберігати дані у цільовій системі.

Результат міграції бази даних, що зберігає дані користувачів панелі адміністрування, можна побачити на рисунку 32.

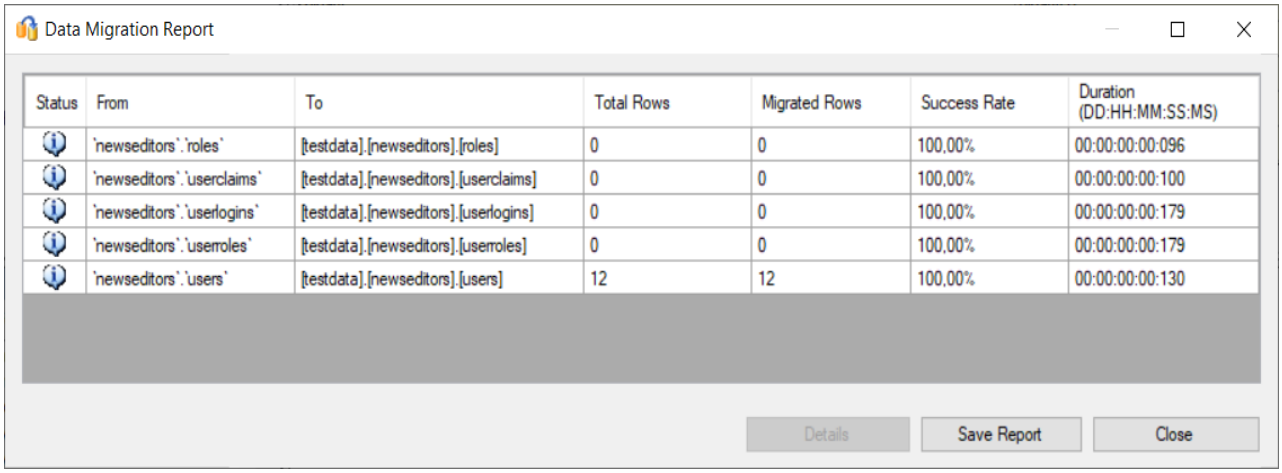


Рис. 32 Звіт міграції даних користувачів

З рисунку 32 можна побачити що всі дані у всіх таблицях успішно мігровані. Проте, оскільки технологією ASP.NET Core Identity, що використовується у веб-застосунку для адміністрування користувачів, було оновлено структуру зберігання, розроблено сценарій, що забезпечить міграцію даних з історичної структури на оновлену. Скрипт наведено на рисунку 33.

```
INSERT INTO testdata.dbo.AspNetUsers
    (Id, Email, NormalizedEmail, EmailConfirmed,
     PhoneNumberConfirmed, TwoFactorEnabled, LockoutEnabled,
     AccessFailedCount, PasswordHash, SecurityStamp,
     ConcurrencyStamp, Username, NormalizedUsername, Fullname)
SELECT Id, Email, UPPER(Email) AS NormalizedEmail, 'False' AS EmailConfirmed,
    'False' AS PhoneNumberConfirmed,
    'False' as TwoFactorEnabled,
    'True' as LockoutEnabled,
    0 AS AccessFailedCount, PasswordHash, SecurityStamp,
    NEWID(), REPLACE(Username,'@newsadmin.univ',''),
    UPPER(REPLACE(Username,'@newsadmin.univ','')) AS NormalizedUsername,
    CONCAT(UserFirsName, ' ', UserMiddleName, ' ', UserLastName) AS Fullname
  FROM testdata.newseditors.users;
```

Рис. 33 Лістинг коду. Сценарій міграції даних користувачів до нової структури зберігання

На рисунку 33 можна побачити, що SQL-скрипт заповнює відповідні присутні дані, такі як логін, пошта, ім’я користувача, хеш паролю, та інші поля таблиці відповідно до нової структури. Сценарій забезпечить непомітний перехід користувачів на модернізовану систему, оскільки повторна реєстрація не буде потрібною, дані для авторизації та автентифікації залишаються незмінними.

3.5 Валідація даних в цільовій системі

Кінцевим етапом міграції даних є перевірка структури та даних, тестування та коректування застосунку, який використовує базу даних. Процес валідації забезпечує повноту та цілісність даних за рахунок виявлення, вилучення та виправлення помилок, якщо такі стались.

У випадку, коли таблиці бази даних містять велику кількість даних, мануальної перевірки недостатньо і потрібно застосовувати програмні засоби для забезпечення точного порівняння даних після міграції. Для автоматизованої перевірки використано автоматизований консольний застосунок для порівняння баз даних Database Compare Utility. Застосунок призначений для порівняння SQL Server баз даних з SQL Server, MySQL, Oracle або Postgres базами даних і використовує пропрієтарні .NET драйвери для підключення до них. Принцип роботи застосунку: визначається вихідна та цільова бази даних, за замовчуванням

порівнюються всі таблиці баз даних, які повинні мати однакову структуру та назви. Для коректного запуску застосунку необхідно вказати всі параметри у файлі конфігурації, наведеному на рисунку 34.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>
  <appSettings>
     <add key="TargetDBType" value="SQLSERVER" />
    <add key="TargetConnectionString" value="Server=KIF\SQLEXPRESS02; Database=testdata;Integrated Security=SSPI;Encrypt=False" />

    <add key="SourceDBType" value="MYSQL" />
    <add key="SourceConnectionString" value="server=127.0.0.1;uid=root;pwd=****;database=newseditors" />
    <add key="OutputFile" value="testdata.xlsx" />
    <add key="OverwriteFile" value="N" />
    <add key="LogFilePath" value="DataCompare-4" />
    <add key="LimitMismatches" value="N" />
    <add key="MaxMismatchPerTable" value="10000" />
    <add key="TableFilter" value="" />
    <add key="SchemaFilter" value="" />
    <add key="SrcSchema" value="" />
    <add key="SrcUseSchema" value="N" />
    <add key="TargUseSchema" value="Y" />
    <add key="UpcaseIdentifiers" value="F" />
    <add key="IgnoreTimezone" value="T"/>
    <add key="SourceTimezoneOffset" value="0"/>
    <add key="RemoveCR" value="T" />
    <add key="RemoveLF" value="F" />
    <add key="TrimSpace" value="T" />
    <add key="CastUIDToRaw" value="T" />
    <add key="RemoveXMLLFSp" value="T" />
    <add key="CastMD5ToUnicode" value="T"/>
    <add key="CastMD5ToVarchar" value="F" />
    <add key="CastBinToHex" value="F" />
    <add key="OptimizeSQL" value="F"/>
    <add key="TrimBinZeros" value="F" />
    <add key="FixXMLSymbols" value="F" />
    <add key="ResetConnection" value="F"/>
    <add key="RowCountOnly" value="N" />
```

Рис. 34 Конфігураційний файл застосунку для тестування міграції

На рисунку 34 можна побачити конфігураційний XML-файл, який містить параметри утиліти та дозволяє змінити налаштування за замовчуванням, наприклад:

TargetDBType – тип СКБД цільової бази даних. В даному випадку SQL Server.

TargetConnectionString – рядок підключення до цільової бази даних

SourceDBType – тип СКБД вихідної бази даних. В даному випадку MySQL.

SourceConnectionString – рядок підключення до вихідної бази даних

MaxThreads – максимальна кількість потоків застосунку, що будуть використані для запущеного процесу порівняння.

OutputFile – опціональний вихідний файл Excel з метриками виконання

LogFilePath – шлях для запису журналу виконання перевірки даних (одна таблиця – один файл).

LimitMismatches – параметр, що дозволяє обмежити кількість неспівпадінь для запису в журнал. В даному випадку необхідно порівняти всі таблиці.

MaxMismatchPerTable – максимальна кількість записаних неспівпадінь, якщо попередній параметр встановлено в True.

TableFilter – опціональний фільтр таблиць, що перевіряються

SchemaFilter - опціональний фільтр схем баз даних, що перевіряються

RowCountOnly – параметр, що дозволяє порівнювати лише кількість рядків у таблицях, якщо порівняння даних не потрібне.

Фільтри можуть бути вказані у вигляді Transact-SQL виразів, які містять універсальний символ. Якщо вони вказуються, то вираз фільтру виглядатиме наступним чином: select … where schema_name like '<Schema Filter>' and table_name like '<Table Filter>'.

Алгоритм роботи застосунку полягає у побудові SQL-запитів вибірки даних для вихідної і цільової платформ, збираючи поля рядку таблиці в один рядок і обчислюючи хеш-функцію за алгоритмом MD5, для чого таблиці баз даних повинні мати первинні ключі [16].

Детальний результат перевірки для бази даних fresh можна побачити у Додатку 3. Проаналізувавши повний журнал можна зробити висновок, що всі таблиці після міграції містять однакову кількість рядків, що наведено на рис. 35.

```
Строка 6: 2023.04.09 22:31:14 (0) Row counts (32155) match on both source and target for table: fresh.mediafiles
Строка 8: 2023.04.09 22:31:14 (1) Row counts (31118) match on both source and target for table: fresh.history
Строка 10: 2023.04.09 22:31:14 (2) Row counts (15683) match on both source and target for table: fresh.newshistory
Строка 12: 2023.04.09 22:31:14 (3) Row counts (10734) match on both source and target for table: fresh.news
Строка 14: 2023.04.09 22:31:15 (4) Row counts (4588) match on both source and target for table: fresh.mediafilesgroup
Строка 16: 2023.04.09 22:31:15 (5) Row counts (513) match on both source and target for table: fresh.guests_umz
Строка 20: 2023.04.09 22:31:15 (4) Row counts (198) match on both source and target for table: fresh.addresslist
Строка 23: 2023.04.09 22:31:15 (6) Row counts (134) match on both source and target for table: fresh.guests
Строка 27: 2023.04.09 22:31:15 (1) Row counts (132) match on both source and target for table: fresh.news_tmp
Строка 32: 2023.04.09 22:31:15 (4) Row counts (32) match on both source and target for table: fresh.alumni
Строка 36: 2023.04.09 22:31:15 (7) Row counts (29) match on both source and target for table: fresh.umzphoto
Строка 49: 2023.04.09 22:31:16 (4) Row counts (12) match on both source and target for table: fresh.umz_topics_all
Строка 56: 2023.04.09 22:31:16 (0) Row counts (8) match on both source and target for table: fresh.umzguest
Строка 60: 2023.04.09 22:31:16 (4) Row counts (6) match on both source and target for table: fresh.votes_all
Строка 65: 2023.04.09 22:31:16 (0) Row counts (5) match on both source and target for table: fresh.news_umz
Строка 70: 2023.04.09 22:31:16 (7) Row counts (4) match on both source and target for table: fresh.topics_all
Строка 74: 2023.04.09 22:31:16 (0) Row counts (3) match on both source and target for table: fresh.photos_all
Строка 78: 2023.04.09 22:31:16 (4) Row counts (3) match on both source and target for table: fresh.photos_link
Строка 82: 2023.04.09 22:31:16 (0) Row counts (3) match on both source and target for table: fresh.users_all
Строка 94: 2023.04.09 22:31:16 (0) Row counts (2) match on both source and target for table: fresh.groups_all
Строка 98: 2023.04.09 22:31:16 (4) Row counts (2) match on both source and target for table: fresh.photo_series_all
Строка 102: 2023.04.09 22:31:16 (0) Row counts (2) match on both source and target for table: fresh.stats_day
Строка 106: 2023.04.09 22:31:16 (4) Row counts (1) match on both source and target for table: fresh.photo_topics_all
Строка 111: 2023.04.09 22:31:16 (0) Row counts (1) match on both source and target for table: fresh.news_link
Строка 115: 2023.04.09 22:31:16 (0) Row counts (1) match on both source and target for table: fresh.polls_all
Строка 119: 2023.04.09 22:31:16 (4) Row counts (0) match on both source and target for table: fresh.umz
Строка 124: 2023.04.09 22:31:16 (6) Row counts (0) match on both source and target for table: fresh.photos
```

Рис. 35 Список таблиць для яких співпадає кількість рядків

На рисунку 35 можна побачити список всіх таблиць, для яких проводилось порівняння кількості рядків. За даним параметром можна вважати міграцію успішною.

Проте, як зазначено в останньому рядку журналу, тестування показало розбіжності при обрахунку хеш-функцій для деяких таблиць. Для кінцевого висновку про результат міграції необхідно проаналізувати детальні журнали таблиць. Оскільки таблиця news є основною, що містить головну сутність предметної області, розглянемо її. Приклад запиту, що обраховує вибірку даних для порівняння таблиці news наведено на рисунку 36.

```
Source Select:
    select `nid`, UPPER(MD5(CONCAT_WS(_utf16'|',
    case when `topic` is null then _utf16'~'
        else CAST(UPPER(MD5(CAST(`topic` as char character set utf16le))) as char character set utf16le) end,
    case when `title` is null then _utf16'~'
        else CAST(UPPER(MD5(CAST(`title` as char character set utf16le))) as char character set utf16le) end,
    case when `content` is null then _utf16'~'
        else CAST(UPPER(MD5(CAST(`content` as char character set utf16le))) as char character set utf16le) end,
    CAST(IFNULL(REPLACE(FORMAT(`date`, 0), ',', ''), _utf16'~') as char character set utf16le),
    CAST(IFNULL(REPLACE(FORMAT(`archive`, 0), ',', ''), _utf16'~') as char character set utf16le),
    case when `titleEN` is null then _utf16'~'
        else CAST(UPPER(MD5(CAST(`titleEN` as char character set utf16le))) as char character set utf16le) end,
    case when `titleRU` is null then _utf16'~'
        else CAST(UPPER(MD5(CAST(`titleRU` as char character set utf16le))) as char character set utf16le) end,
    case when `contentEN` is null then _utf16'~'
        else CAST(UPPER(MD5(CAST(`contentEN` as char character set utf16le))) as char character set utf16le) end,
    case when `contentRU` is null then _utf16'~'
        else CAST(UPPER(MD5(CAST(`contentRU` as char character set utf16le))) as char character set utf16le) end,
    case when `tagGroup` is null then _utf16'~'
        else CAST(UPPER(MD5(CAST(`tagGroup` as char character set utf16le))) as char character set utf16le) end,
    case when `teazer` is null then _utf16'~'
        else CAST(UPPER(MD5(CAST(`teazer` as char character set utf16le))) as char character set utf16le) end,
    CAST(IFNULL(REPLACE(FORMAT(`pictGroup`, 0), ',', ''), _utf16'~') as char character set utf16le),
    CAST(IFNULL(REPLACE(FORMAT(`vidGroup`, 0), ',', ''), _utf16'~') as char character set utf16le),
    CAST(IFNULL(REPLACE(FORMAT(`docGroup`, 0), ',', ''), _utf16'~') as char character set utf16le),
    CAST(IFNULL(REPLACE(FORMAT(`hidden`, 0), ',', ''), _utf16'~') as char character set utf16le),
    CAST(IFNULL(REPLACE(FORMAT(`lastChangeID`, 0), ',', ''), _utf16'~') as char character set utf16le),
    CAST(IFNULL(REPLACE(FORMAT(`hiddenFilesGroup`, 0), ',', ''), _utf16'~') as char character set utf16le))) as `hash`
    from `news`
    ORDER BY `nid`
Target Select: select [nid],
        HASHBYTES('MD5', coalesce(convert(nvarchar(32),
        HASHBYTES('MD5', rtrim(replace([topic], char(13), ''))), 2), N'~')
        +N'|'+coalesce(convert(nvarchar(32),HASHBYTES('MD5', rtrim(replace([title], char(13), ''))), 2), N'~')
        +N'|'+coalesce(convert(nvarchar(32),HASHBYTES('MD5', rtrim(replace([content], char(13), ''))), 2), N'~')
        +N'|'+ISNULL(cast([date] as nvarchar(50)), N'~')+N'|'+ISNULL(cast([archive] as nvarchar(50)), N'~')
        +N'|'+coalesce(convert(nvarchar(32),HASHBYTES('MD5', rtrim(replace([titleEN], char(13), ''))), 2), N'~')
        +N'|'+coalesce(convert(nvarchar(32),HASHBYTES('MD5', rtrim(replace([titleRU], char(13), ''))), 2), N'~')
        +N'|'+coalesce(convert(nvarchar(32),HASHBYTES('MD5', rtrim(replace([contentEN], char(13), ''))), 2), N'~')
        +N'|'+coalesce(convert(nvarchar(32),HASHBYTES('MD5', rtrim(replace([contentRU], char(13), ''))), 2), N'~')
        +N'|'+coalesce(convert(nvarchar(32),HASHBYTES('MD5', rtrim(replace([tagGroup], char(13), ''))), 2), N'~')
        +N'|'+coalesce(convert(nvarchar(32),HASHBYTES('MD5', rtrim(replace([teazer], char(13), ''))), 2), N'~')
        +N'|'+ISNULL(cast([pictGroup] as nvarchar(50)), N'~')
        +N'|'+ISNULL(cast([vidGroup] as nvarchar(50)), N'~')
        +N'|'+ISNULL(cast([docGroup] as nvarchar(50)), N'~')
        +N'|'+ISNULL(cast([hidden] as nvarchar(50)), N'~')
        +N'|'+ISNULL(cast([lastChangeID] as nvarchar(50)), N'~')
        +N'|'+ISNULL(cast([hiddenFilesGroup] as nvarchar(50)), N'~'))
        FROM [fresh].[news]
        ORDER BY [nid]
```

Рис. 36 Запит для порівняння вихідної та цільової таблиць

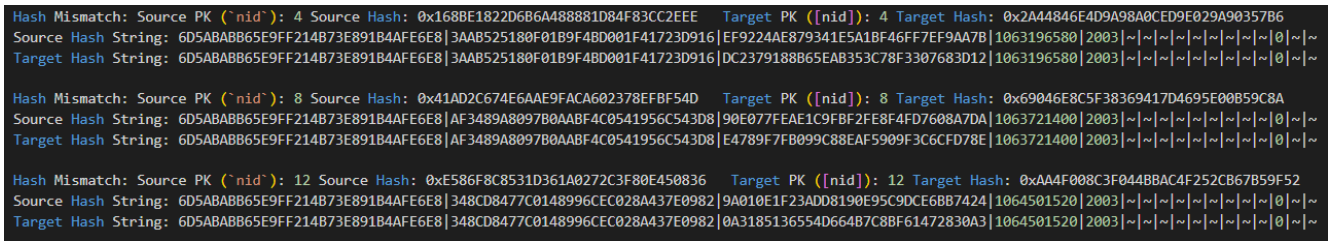


Рис. 37 Приклад журналу порівняння для таблиці news

З рисунку 37 можна побачити запити, що формують рядок з кожного поля таблиці бази даних і потім обраховують хеш-функцію. З результатів порівняння, що зображені на рис. бачимо, що хеш-функція відрізняється для поля content. Поле містить деталі новини у вигляді html-розмітки. Перевіримо детальніше, які саме дані втратились під час міграції шляхом порівняння даних по символах, адже тип даних колонки content - текст. Приклад перевірки зображено на рисунку 38.

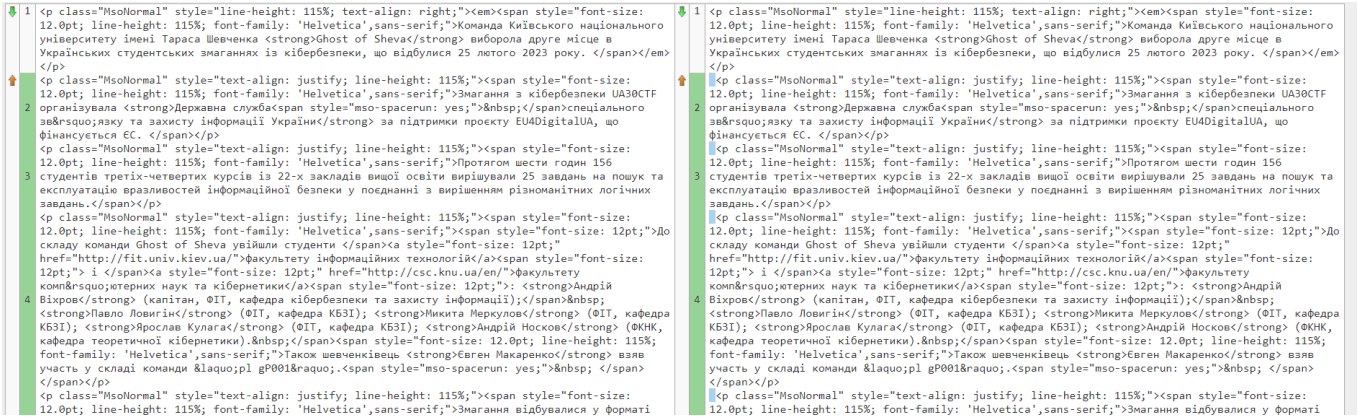


Рис. 38 Приклад порівняння вмісту поля content

Як можна побачити з прикладу на рисунку 38, було втрачено пробіли, проте, оскільки дані зберігаються і відтворюються у HTML-форматі, порожні символи не несуть корисної інформації. Можна вважати, що дані не втратили повноти.

3.6 Оновлення застосунку після міграції

Оскільки маємо створену базу даних, необхідно надати можливість панелі адміністрування працювати з нею. Перш за все, створюється міграція, що дозволить працювати з моделям з коду за допомогою інструменту об’єктно-

реляційного відображення Entity Framework Core. Технологія надає інструменти роботи з об’єктами та їх колекціями замість таблиць. Інструмент підтримує кілька механізмів: code first – спочатку створюються об’єкти застосунку, а потім відповідні таблиці в базі даних, database first – спочатку створюються таблиці бази даних, а потім вносяться відповідні зміни до об’єктів застосунку. В даному випадку необхідно перейти від використання database first підходу до code first.

Для цього створено міграцію, що надасть змогу використовувати базу даних після проведеної міграції. Приклад створення таблиці наведено на рисунку 39.

```
migrationBuilder.CreateTable(
    name: "news",
    schema: "fresh",
    columns: table => new
    {
        nid = table.Column<int>(nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        topic = table.Column<string>(maxLength: 255, nullable: true),
        title = table.Column<string>(type: "text", maxLength: 65535, nullable: true),
        content = table.Column<string>(type: "text", maxLength: 65535, nullable: true),
        date = table.Column<int>(nullable: true),
        archive = table.Column<int>(nullable: true),
        titleEN = table.Column<string>(type: "text", maxLength: 65535, nullable: true),
        titleRU = table.Column<string>(type: "text", maxLength: 65535, nullable: true),
        contentEN = table.Column<string>(type: "text", maxLength: 65535, nullable: true),
        contentRU = table.Column<string>(type: "text", maxLength: 65535, nullable: true),
        tagGroup = table.Column<string>(maxLength: 255, nullable: true),
        teazer = table.Column<string>(maxLength: 255, nullable: true),
        pictGroup = table.Column<int>(nullable: true),
        vidGroup = table.Column<int>(nullable: true),
        docGroup = table.Column<int>(nullable: true),
        hiddenFilesGroup = table.Column<int>(nullable: true),
        hidden = table.Column<bool>(nullable: false),
        lastChangeID = table.Column<int>(nullable: true)
    },
    constraints: table =>{
        table.PrimaryKey("PK_news_nid", x => x.nid);
        table.ForeignKey(
            name: "FK_news_mediafilesgroup_docGroup",
            column: x => x.docGroup,
            principalSchema: "fresh",
            principalTable: "mediafilesgroup",
            principalColumn: "id",
            onDelete: ReferentialAction.Restrict);
        table.ForeignKey(
            name: "FK_news_mediafilesgroup_hiddenFilesGroup",
            column: x => x.hiddenFilesGroup,
            principalSchema: "fresh",
            principalTable: "mediafilesgroup",
            principalColumn: "id",
            onDelete: ReferentialAction.Restrict);
        table.ForeignKey(
            name: "FK_news_mediafilesgroup_pictGroup",
            column: x => x.pictGroup,
            principalSchema: "fresh",
            principalTable: "mediafilesgroup",
            principalColumn: "id",
            onDelete: ReferentialAction.Restrict);
        table.ForeignKey(
            name: "FK_news_mediafilesgroup_vidGroup",
            column: x => x.vidGroup,
            principalSchema: "fresh",
            principalTable: "mediafilesgroup",
            principalColumn: "id",
            onDelete: ReferentialAction.Restrict);
    });
```

Рис. 39 Лістинг коду. Створення таблиці за допомогою EF Core

На рисунку 39 зображено частину програмної міграції для відтворення таблиці news з коду застосунку. Можна побачити, що застосовуються всі необхідні обмеження (наприклад, зовнішні ключі), типи даних та набір полів, що і в базі даних. Це дозволяє повторно відтворювати структуру даних за необхідності і коректно працювати з базою даних. Відповідне створення стосується всіх таблиць бази даних.

3.7 Розробка спеціалізованої інтеграції

3.7.1 Graph API

Graph API є основним способом обміну інформації на платформі Facebook. Це API на основі HTTP, який програми можуть використовувати для програмного запиту даних, публікації нових історій, керування рекламою, завантаження фотографій і виконання багатьох інших завдань.

«Соціальний граф» є основоположною ідеєю Graph API і представленням даних на платформі Facebook. Він складається з вузлів, ребер і полів. Зазвичай вузли використовуються, щоб отримати дані про конкретний об’єкт, ребра використовуються, щоб отримати колекції об’єктів на одному об’єкті, і поля використовуються, щоб отримати дані про один об’єкт або кожен об’єкт у колекції. У документації Facebook називає вузол і ребро «endpoint».

Передача даних відповідає стандарту HTTP/1.1, і всі кінцеві точки потребують HTTPS. Оскільки Graph API базується на HTTP, він працює з будь-якою мовою, яка має бібліотеку HTTP, наприклад cURL.

Майже всі запити передаються на URL-адресу хосту graph.facebook.com. Єдиним винятком є завантаження відео, яке використовує graph-video.facebook.com.

Токени доступу дозволяють застосунку отримувати доступ до Graph API. Майже для всіх запитів до Graph API потрібен певний токен доступу, тому кожен раз, при отриманні доступу до HTTP-методу, запит може потребувати його.

Зазвичай вони виконують дві функції:

отримання доступу до інформації користувача без запиту пароля користувача;

ідентифікація програми, користувача, який використовує програму, і тип даних, до яких користувач надав доступ програмі.

Вузол — це окремий об’єкт з унікальним ідентифікатором. Наприклад, існує багато об’єктів вузла «Користувач», кожен з яких має унікальний ідентифікатор, що представляє людину на Facebook. Сторінки, групи, публікації, фотографії та коментарі – це лише деякі з вузлів Facebook Social Graph.

Ребро – являє собою з'єднання між двома вузлами. Наприклад, до вузла «Користувач» можуть бути підключені фотографії, а до вузла «Фото» — коментарі.

Поля є властивостями вузла. При запиті вузла або ребра, він за замовчуванням повертає набір полів. Однак існує можливість вказати, які необхідно повернути, використовуючи параметр fields і перераховуючи кожне поле. Це замінює значення за замовчуванням і повертає лише вказані поля та ідентифікатор об’єкта, який завжди повертається [17].

3.7.2 Зберігання даних

Оскільки формат даних для публікації у соціальній мережі вимагає текстового відображення, тоді як публікація на веб-сайт вимагає HTML-розмітки, для його зберігання було створено окрему сутність FacebookNews, що зображено на рисунку 40.

```
namespace core_news_admin.Models.DataModels
{
    8 references
    public class FacebookNews
    {
        4 references
        public int Id { get; set; }
        1 reference
        public string OriginalNewsId { get; set; }
        0 references
        public News OriginalNews { get; set; }
        1 reference
        public string TextContext { get; set; }
        0 references
        public bool Saved { get; set; }
        0 references
        public bool Hidden { get; set; }
        0 references
        public string PagePostId { get; set; }
    }
}
```

Рис. 40 Лістинг коду. Модель новини Фейсбук

З наступних рядків на рисунку 41 можна побачити що дана сутність зберігає зв'язок з сутністю основної новини з використанням EF Core задля забезпечення відповідності:

```
0 references
public string OriginalNewsId { get; set; }
0 references
public News OriginalNews { get; set; }
```

Рис. 41 Лістинг коду. Зв’язок двох сутностей

Набори сутностей поєднуються у єдину базу даних за допомогою контексту бази даних, зображеного на рисунку 42 який складається з їх опису:

```
using core_news_admin.Models.DataModels;

namespace core_news_admin
{
    10 references
    public partial class DatabaseContext : IdentityDbContext<ApplicationUser>
    {
        0 references
        public DatabaseContext(DbContextOptions<DatabaseContext> options)
            : base(options)
        {
            Database.EnsureCreated();
        }

        3 references
        public virtual DbSet<History> History { get; set; }
        4 references
        public virtual DbSet<MediaFiles> MediaFiles { get; set; }
        3 references
        public virtual DbSet<MediaFilesGroup> MediaFilesGroup { get; set; }
        10 references
        public virtual DbSet<News> News { get; set; }

        2 references
        public virtual DbSet<NewsHistory> NewsHistory { get; set; }
        0 references
        public virtual DbSet<FacebookNews> FacebookNews { get; set; }
```

Рис. 42 Лістинг коду. Контекст бази даних

Відповідно до рисунку 42 можна побачити що додаткова сутність новини соціальної мережі також введена до контексту бази даних, відповідно до чого створено міграцію для оновлення БД, яку можна побачити на рисунку 43.

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "FacebookNews",
        columns: table => new
        {
            Id = table.Column<int>(nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            OriginalNewsId = table.Column<string>(nullable: true),
            OriginalNewsNid = table.Column<int>(nullable: true),
            TextContext = table.Column<string>(nullable: true),
            Saved = table.Column<bool>(nullable: false),
            Hidden = table.Column<bool>(nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_FacebookNews", x => x.Id);
            table.ForeignKey(
                name: "FK_FacebookNews_news_OriginalNewsNid",
                column: x => x.OriginalNewsNid,
                principalSchema: "fresh",
                principalTable: "news",
                principalColumn: "nid",
                onDelete: ReferentialAction.Restrict);
        });

    migrationBuilder.CreateIndex(
        name: "IX_FacebookNews_OriginalNewsNid",
        table: "FacebookNews",
        column: "OriginalNewsNid");
}
```

Рис. 43 Створення таблиці для зберігання новин

З рисунку 43 можемо побачити створення нової таблиці для зберігання новин. Оскільки платформа Facebook потребує публікацій у текстовому форматі та часто зі зміненим змістом відносно основної новини, необхідно зберігати ці дані окремо.

3.7.3 Обробка даних

Для реалізації алгоритму обміну даними з Facebook підготовано відповідний сервіс-клієнт FacebookService, який за рахунок HTTP-клієнта оброблюватиме дані відповідно до формату, що надає Graph API. Реалізацію сервісу можна побачити в додатку 4. Сервіс дозволяє створювати новину в адміністративній панелі, публікувати її на Facebook, приховувати допис від читачів, видаляти його повністю. При видаленні збережена можливість повторної публікації та

редагування.

Для відповідної роботи з користувачем в рамках Graph API реалізовано сутність User, зображену на рисунку 44. Сутність містить в собі токен доступу та логін, що необхідно для авторизації та автентифікації.

```
public class User
{
    0 references
    public string Id { get; set; }
    0 references
    public string LoginUrl { get; set; }
    0 references
    public string AccessToken { get; set; }
}
```

Рис. 44 Лістинг коду. Сутність User

Graph API повертає інформацію одразу у форматі пагінованих сторінок, відповідно для роботи з ними впроваджено клас вказівник на сторінку FacebookPaginator, зображений на рисунку 45.

```
namespace core_news_admin.Services.FacebookCore
{
    3 references
    public class FacebookPaginator
    {
        0 references
        public FacebookPaginator() {}
        0 references
        public FacebookPaginator(string before, string after)
        {
            Before = before;
            After = after;
        }
        3 references
        public string Before { get; internal set; }
        3 references
        public string After { get; internal set; }
    }
    5 references
    public enum Direction
    {
        None,
        Next,
        Previous,
        Before,
        After
    }
}
```

Рис. 45 Лістинг коду. Клас-вказівник FacebookPaginator.

Використовуючи архітектурний шаблон Модель-Представлення-Контролер, передбачений технологією ASP.NET Core MVC, реалізовано набір контролерів, що керують доступом до даних та їх обробкою, представлень, що відображаються користувачам та моделей, що описують сутності, призначені для зберігання даних, відображення необхідної інформації у представленнях чи описі даних, що має віддати представлення у контролер.

Контролери реалізовані з додержанням принципів REST.

REST – передача стану представлення – архітектурний підхід до побудови API, який дає можливість найефективнішим чином використовувати протокол HTTP, надаючи доступ до способів обробки і передачі стану ресурсів.

Для виконання маніпуляцій з даними HTTP надає перелік методів, основні з яких:

GET – запит представлення ресурсу, використовуючи даний метод можна лише отримувати дані.

PUT – заміна всіх поточних представлень даними запиту.

POST – відправка сутностей до певного ресурсу, викликає зміни на сервері.

DELETE – видалення вказаного ресурсу.

Наприклад, видалення новини на Facebook реалізовано за допомогою методу GET та передачею унікального ідентифікатору в якості параметру, що можна побачити на рисунку 46.

```
// GET: FacebookController/Delete/5
0 references
public async Task<ActionResult> Delete(int id)
{
    FacebookNews deletedNews = await facebookService.DeleteFromFacebook(id);
    return View(deletedNews);
}
```

Рис. 46 Лістинг коду. Метод видалення новини

Також у панелі управління реалізовано створення новини у панелі адміністрування за допомогою методу POST, що можна побачити нижче на

рисунку 47.

```
// Post: FacebookController/Create
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<ActionResult> Create(FacebookNews news)
{
    FacebookNews newNews = await facebookService.Save(news);
    return View(newNews);
}
```

Рис. 47 Лістинг коду. Метод завантаження файлів

За замовчуванням методи контролеру є методами GET, тому для вирішення цієї задачу використовуються атрибути – наприклад [HttpPost], як зазначено в прикладі вище. Атрибути вказуються для засобів маршрутизації для вибору дії і контролеру на основі методу HTTP запиту.

Вважається гарною практикою використання методу POST і передачі в його тіло моделі-параметру замість аналогічного запиту, але з використанням методу GET і передачі параметру в рядку запиту.

Крім того, ASP.NET Core MVC має вбудовану можливість впровадження залежностей (DI). Контролери можуть запитувати необхідні служби наприклад через свої конструктори.

Наприклад, контролер для роботи з Facebook має в собі містити сервіс що дає доступ до роботи з Graph API і не може працювати без даної залежності. Про це вказано у сигнатурі конструктору, як можна побачити нижче, на рисунку 48:

```
namespace core_news_admin.Controllers
{
    [Authorize]
    1 reference
    public class FacebookController : Controller
    {
        private FacebookService facebookService;

        0 references
        public FacebookController(FacebookService facebookService)
        {
            this.facebookService = facebookService;
        }

        // GET: FacebookController
        0 references
        public async Task<ActionResult> Index(int? page)
        {
            List<FacebookNews> allNews = await facebookService.GetAll();

            int pageSize = 40;
            int pageNumber = (page ?? 1);

            return View(allNews.ToPagedList(pageNumber, pageSize));
        }
```

Рис. 48 Лістинг коду. Контролер роботи з новинами

На рисунку 49 зображено приклад опублікованого допису в соціальній мережі Facebook.

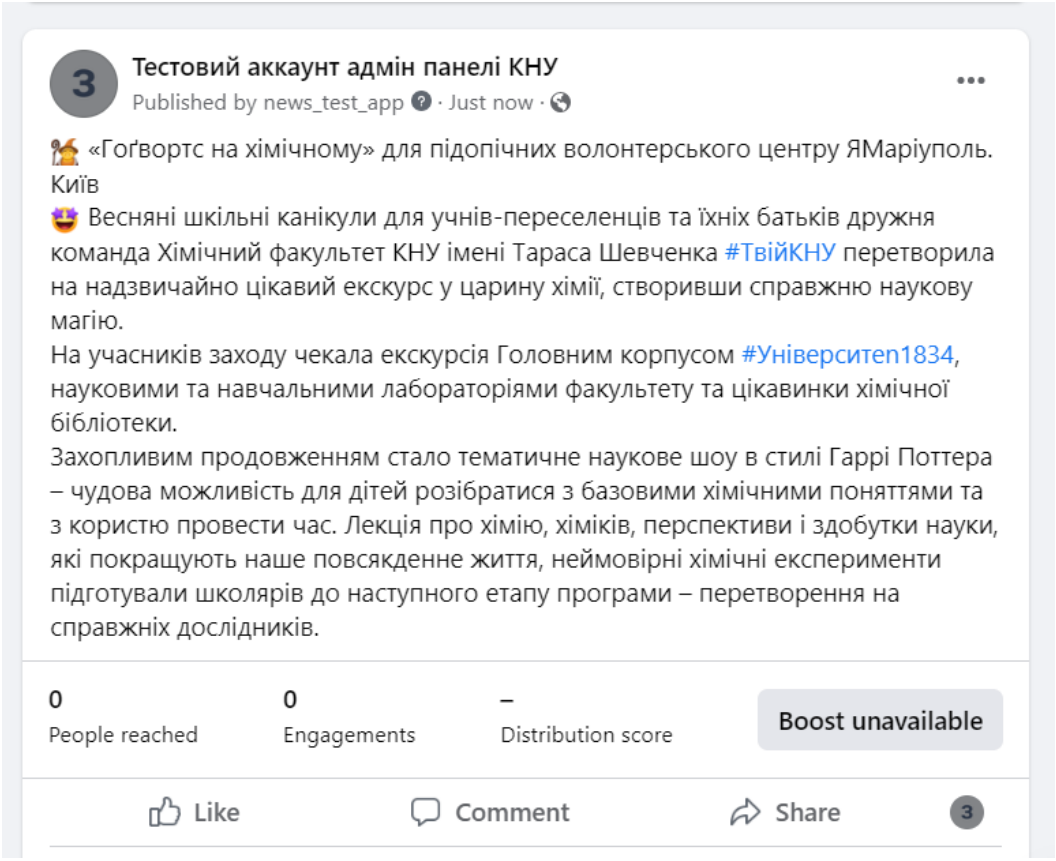


Рис. 49 Приклад опублікованого допису на Facebook

Як можна побачити з рисунку 49, допис містить текст, зазначену сторінку публікації та застосунок, що його опублікував. Кожен з цих параметрів можна налаштувати в адміністративній панелі соціальної мережі сторінки Facebook.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено аналітичний огляд способів і методів переносу даних, а також проаналізовано технології та підходи інтеграції комплексів програмних систем, що дозволило переконатись у значному впливі необхідності взаємодії інформаційних систем шляхом обміну даних та стало підґрунтям для розробки методу міграції даних та модернізації системи управління публікацією новин засобами веб-інтерфейсу.

За результатами аналізу розроблено та реалізовано метод міграції бази даних зі зміною системи керування базами даних MySQL на Microsoft SQL Server. Міграція бази даних дозволяє виправляти помилки та адаптувати дані до змін вимог.

Враховуючи наявні архітектурні підходи будо спроєктовано та реалізовано інтеграцію веб-додатку з соціальною мережею Facebook. Інтеграція вирішує питання диверсифікації засобів зовнішньої комунікації за рахунок поширення інформації у різних незалежних одне від одного засобах масової інформації.

Потенційними користувачами системи є адміністратори сайту закладу вищої освіти, а також студенти. Застосунок може бути розширений додатковими спеціалізованими модулями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statista [Електронний ресурс] — Режим доступу: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/
2. Gorman K., Hirt A., Noderer D., Rowland-Jones J., Sirpal A., Ryan D., Woody B. Introducing Microsoft SQL Server – UK, Birmingham: Packt Publishing Ltd., 2019. – 489 с.
3. Russom P. Best Practices in Data Migration. - TDWI Monograph, 2006 – 13 с.
4. Kimball R., Caserta J., The Data Warehouse ETL Toolkit Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data., - Wiley Publishing, Inc., 2004.- 526 с.
5. Gu L., Budd L., Cayci A., Hendricks C., Purnell M., Rigdon C. A Practical Guide to DB2 UDB Data Replication V8, - IBM Corp. 2002, - 576 с.
6. Reasons to Automate ETL Processes. [Електронний ресурс] — Режим доступу: https://www.wherescape.com/blog/8-reasons-why-you-need-to-automate-your-etl-processes/
7. Migration consistency [Електронний ресурс] — Режим доступу: https://cloud.google.com/architecture/database-migration-concepts-principles-part-1#migration_consistency
8. Summers B. L. Effective Methods for Software and Systems Integration., Taylor & Francis Group, LLC 2013, - 196 с.
9. Kleppman M., (2017) Designing DataIntensive Applications: The big ideas Behind Reliable, Scalable, and Maintainable Systems, USA, O'Reilly Media, Inc.
10. Effects of Database Size on Rule System Performance: Five Case Studies [Електронний ресурс] — Режим доступу: https://www.vldb.org/conf/1991/P287.PDF
11. Hohpe G., Woolf B. Enterprise Integration Patterns, The Addison Wesley Signature Series., - 574 с.

12. Masse M. REST Api. Design Rulebook. O’Reilly 2012 , 114 с.
13. mysqldump — A Database Backup Program [Електронний ресурс] — Режим доступу:https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#mysqldump-syntax
14. MySQL Workbench [Електронний ресурс] — Режим доступу: https://dev.mysql.com/doc/refman/8.0/en/workbench.html
15. Migration guide: MySQL to SQL Server [Електронний ресурс] — Режим доступу:https://learn.microsoft.com/ru-ru/sql/sql-server/migrate/guides/mysql-to-sql-server?view=sql-server-ver16
16. Database Compare Utility [Електронний ресурс] — Режим доступу:https://www.microsoft.com/en-us/download/details.aspx?id=103016
17. Graph API [Електронний ресурс] — Режим доступу: https://developers.facebook.com/docs/graph-api/

ДОДАТОК 1

Запити на створення таблиць при міграції даних для верифікації

Таблиця History:

```
CREATE TABLE `history` (
  `historyID` int NOT NULL AUTO_INCREMENT,
  `userID` varchar(128) CHARACTER SET cp1251 COLLATE cp1251_ukrainian_ci NOT NULL,
  `newsID` int NOT NULL,
  `newsHistoryID` int DEFAULT NULL,
  `date` int NOT NULL,
  `userActionID` int NOT NULL,
  PRIMARY KEY (`historyID`),
  KEY `userID_idx` (`userID`),
  KEY `newsID_idx` (`newsID`),
  KEY `news_oldID_idx` (`newsHistoryID`)
) ENGINE=InnoDB AUTO_INCREMENT=31478 DEFAULT CHARSET=cp1251 COLLATE=cp1251_ukrainian_ci
```

Таблиця NewsHistory:

```
CREATE TABLE `newshistory` (
  `nid` int NOT NULL AUTO_INCREMENT,
  `topic` varchar(255) CHARACTER SET cp1251 COLLATE cp1251_ukrainian_ci DEFAULT NULL,
  `title` text CHARACTER SET cp1251 COLLATE cp1251_ukrainian_ci,
  `content` text CHARACTER SET cp1251 COLLATE cp1251_ukrainian_ci,
  `date` int DEFAULT NULL,
  `titleEN` text CHARACTER SET cp1251 COLLATE cp1251_general_ci,
  `titleRU` text CHARACTER SET cp1251 COLLATE cp1251_general_ci,
  `contentEN` text CHARACTER SET cp1251 COLLATE cp1251_general_ci,
  `contentRU` text CHARACTER SET cp1251 COLLATE cp1251_general_ci,
  `tagGroup` varchar(255) CHARACTER SET cp1251 COLLATE cp1251_ukrainian_ci DEFAULT NULL,
  `teazer` varchar(255) CHARACTER SET cp1251 COLLATE cp1251_ukrainian_ci DEFAULT NULL,
  PRIMARY KEY (`nid`),
  FULLTEXT KEY `title` (`title`,`content`)
) ENGINE=MyISAM AUTO_INCREMENT=23287 DEFAULT CHARSET=cp1251 COLLATE=cp1251_ukrainian_ci
```

Таблиця Mediafiles:

```
CREATE TABLE `mediafiles` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(100) DEFAULT NULL,
  `description` varchar(100) DEFAULT NULL,
  `link` varchar(100) DEFAULT NULL,
  `saved` bit(1) NOT NULL,
  `mediaGroupID` int NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=32244 DEFAULT CHARSET=latin1
```

Таблиця Mediafilesgroup:

```
CREATE TABLE `mediafilesgroup` (
  `id` int NOT NULL AUTO_INCREMENT,
  `designType` int NOT NULL,
  `positionType` int NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4620 DEFAULT CHARSET=latin1
```

Таблиця News:

```
CREATE TABLE `news` (
  `nid` int NOT NULL AUTO_INCREMENT,
  `topic` varchar(255) CHARACTER SET cp1251 COLLATE cp1251_ukrainian_ci DEFAULT NULL,
  `title` text CHARACTER SET cp1251 COLLATE cp1251_ukrainian_ci,
  `content` text CHARACTER SET cp1251 COLLATE cp1251_ukrainian_ci,
  `date` int DEFAULT NULL,
  `archive` int DEFAULT NULL,
  `titleEN` text CHARACTER SET cp1251 COLLATE cp1251_general_ci,
  `titleRU` text CHARACTER SET cp1251 COLLATE cp1251_general_ci,
  `contentEN` text CHARACTER SET cp1251 COLLATE cp1251_general_ci,
  `contentRU` text CHARACTER SET cp1251 COLLATE cp1251_general_ci,
  `tagGroup` varchar(255) CHARACTER SET cp1251 COLLATE cp1251_general_ci DEFAULT NULL,
  `teazer` varchar(255) CHARACTER SET cp1251 COLLATE cp1251_general_ci DEFAULT NULL,
  `pictGroup` int DEFAULT NULL,
  `vidGroup` int DEFAULT NULL,
  `docGroup` int DEFAULT NULL,
  `hidden` tinyint(1) NOT NULL,
  `lastChangeID` int DEFAULT NULL,
  `hiddenFilesGroup` int DEFAULT NULL,
  PRIMARY KEY (`nid`),
  FULLTEXT KEY `title` (`title`,`content`)
) ENGINE=MyISAM AUTO_INCREMENT=12639 DEFAULT CHARSET=cp1251 COLLATE=cp1251_ukrainian_ci
```

ДОДАТОК 2

Виконані запити для конвертації структури бази даних

Таблиця History:

```
IF EXISTS (SELECT * FROM sys.objects so JOIN sys.schemas sc ON so.schema_id =
sc.schema_id WHERE so.name = N'history'  AND sc.name = N'fresh'  AND type in
(N'U'))
BEGIN

  DECLARE @drop_statement nvarchar(500)

  DECLARE drop_cursor CURSOR FOR
      SELECT 'alter table '+quotename(schema_name(ob.schema_id))+
      '.'+quotename(object_name(ob.object_id))+ ' drop constraint ' +
quotename(fk.name)
      FROM sys.objects ob INNER JOIN sys.foreign_keys fk ON fk.parent_object_id =
ob.object_id
      WHERE fk.referenced_object_id =
          (
             SELECT so.object_id
             FROM sys.objects so JOIN sys.schemas sc
             ON so.schema_id = sc.schema_id
             WHERE so.name = N'history'  AND sc.name = N'fresh'  AND type in
(N'U')
           )

  OPEN drop_cursor

  FETCH NEXT FROM drop_cursor
  INTO @drop_statement

  WHILE @@FETCH_STATUS = 0
  BEGIN
     EXEC (@drop_statement)

     FETCH NEXT FROM drop_cursor
     INTO @drop_statement
  END

  CLOSE drop_cursor
  DEALLOCATE drop_cursor

  DROP TABLE [fresh].[history]
END
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE
[fresh].[history]
(
   [historyID] int IDENTITY(31478, 1)  NOT NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_ukrainian_ci.
```

```
   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR according to character set
mapping for cp1251 character set
   */

   [userID] varchar(128)  NOT NULL,
   [newsID] int  NOT NULL,
   [newsHistoryID] int  NULL,
   [date] int  NOT NULL,
   [userActionID] int  NOT NULL
)
WITH (DATA_COMPRESSION = NONE)
GO
IF EXISTS (SELECT * FROM sys.objects so JOIN sys.schemas sc ON so.schema_id =
sc.schema_id WHERE so.name = N'PK_history_historyID'  AND sc.name = N'fresh'  AND
type in (N'PK'))
ALTER TABLE [fresh].[history] DROP CONSTRAINT [PK_history_historyID]
 GO



ALTER TABLE [fresh].[history]
 ADD CONSTRAINT [PK_history_historyID]
   PRIMARY KEY
   CLUSTERED ([historyID] ASC)

GO

ALTER TABLE  [fresh].[history]
 ADD DEFAULT NULL FOR [newsHistoryID]
GO
```

Таблиця NewsHistory:

```
IF EXISTS (SELECT * FROM sys.objects so JOIN sys.schemas sc ON so.schema_id =
sc.schema_id WHERE so.name = N'newshistory'  AND sc.name = N'fresh'  AND type in
(N'U'))
BEGIN

  DECLARE @drop_statement nvarchar(500)

  DECLARE drop_cursor CURSOR FOR
      SELECT 'alter table '+quotename(schema_name(ob.schema_id))+
      '.'+quotename(object_name(ob.object_id))+ ' drop constraint ' +
quotename(fk.name)
      FROM sys.objects ob INNER JOIN sys.foreign_keys fk ON fk.parent_object_id =
ob.object_id
      WHERE fk.referenced_object_id =
          (
             SELECT so.object_id
             FROM sys.objects so JOIN sys.schemas sc
             ON so.schema_id = sc.schema_id
             WHERE so.name = N'newshistory'  AND sc.name = N'fresh'  AND type in
(N'U')
           )

  OPEN drop_cursor

  FETCH NEXT FROM drop_cursor
  INTO @drop_statement

  WHILE @@FETCH_STATUS = 0
```

```
  BEGIN
     EXEC (@drop_statement)

     FETCH NEXT FROM drop_cursor
     INTO @drop_statement
  END

  CLOSE drop_cursor
  DEALLOCATE drop_cursor

  DROP TABLE [fresh].[newshistory]
END
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE
[fresh].[newshistory]
(
   [nid] int IDENTITY(23287, 1)  NOT NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_ukrainian_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR according to character set
mapping for cp1251 character set
   */

   [topic] varchar(255)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_ukrainian_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [title] varchar(max)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_ukrainian_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [content] varchar(max)  NULL,
   [date] int  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
```

```
cp1251_general_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [titleEN] varchar(max)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_general_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [titleRU] varchar(max)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_general_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [contentEN] varchar(max)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_general_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [contentRU] varchar(max)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_ukrainian_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR according to character set
mapping for cp1251 character set
   */

   [tagGroup] varchar(255)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_ukrainian_ci.

   *   SSMA informational messages:
```

```
   *   M2SS0055: Data type was converted to VARCHAR according to character set
mapping for cp1251 character set
   */

   [teazer] varchar(255)  NULL
)
WITH (DATA_COMPRESSION = NONE)
GO
IF EXISTS (SELECT * FROM sys.objects so JOIN sys.schemas sc ON so.schema_id =
sc.schema_id WHERE so.name = N'PK_newshistory_nid'  AND sc.name = N'fresh'  AND
type in (N'PK'))
ALTER TABLE [fresh].[newshistory] DROP CONSTRAINT [PK_newshistory_nid]
 GO



ALTER TABLE [fresh].[newshistory]
 ADD CONSTRAINT [PK_newshistory_nid]
   PRIMARY KEY
   CLUSTERED ([nid] ASC)

GO

ALTER TABLE  [fresh].[newshistory]
 ADD DEFAULT NULL FOR [topic]
GO

ALTER TABLE  [fresh].[newshistory]
 ADD DEFAULT NULL FOR [date]
GO

ALTER TABLE  [fresh].[newshistory]
 ADD DEFAULT NULL FOR [tagGroup]
GO

ALTER TABLE  [fresh].[newshistory]
 ADD DEFAULT NULL FOR [teazer]
GO
```

Таблиця mediafiles:

```
IF EXISTS (SELECT * FROM sys.objects so JOIN sys.schemas sc ON so.schema_id =
sc.schema_id WHERE so.name = N'mediafiles'  AND sc.name = N'fresh'  AND type in
(N'U'))
BEGIN

  DECLARE @drop_statement nvarchar(500)

  DECLARE drop_cursor CURSOR FOR
      SELECT 'alter table '+quotename(schema_name(ob.schema_id))+
      '.'+quotename(object_name(ob.object_id))+ ' drop constraint ' +
quotename(fk.name)
      FROM sys.objects ob INNER JOIN sys.foreign_keys fk ON fk.parent_object_id =
ob.object_id
      WHERE fk.referenced_object_id =
          (
             SELECT so.object_id
             FROM sys.objects so JOIN sys.schemas sc
             ON so.schema_id = sc.schema_id
             WHERE so.name = N'mediafiles'  AND sc.name = N'fresh'  AND type in
(N'U')
           )
```

```
  OPEN drop_cursor

  FETCH NEXT FROM drop_cursor
  INTO @drop_statement

  WHILE @@FETCH_STATUS = 0
  BEGIN
     EXEC (@drop_statement)

     FETCH NEXT FROM drop_cursor
     INTO @drop_statement
  END

  CLOSE drop_cursor
  DEALLOCATE drop_cursor

  DROP TABLE [fresh].[mediafiles]
END
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE
[fresh].[mediafiles]
(
   [id] int IDENTITY(32244, 1)  NOT NULL,

   /*
   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR according to character set
mapping for latin1 character set
   */

   [name] varchar(100)  NULL,

   /*
   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR according to character set
mapping for latin1 character set
   */

   [description] varchar(100)  NULL,

   /*
   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR according to character set
mapping for latin1 character set
   */

   [link] varchar(100)  NULL,
   [saved] binary(1)  NOT NULL,
   [mediaGroupID] int  NOT NULL
)
WITH (DATA_COMPRESSION = NONE)
GO
IF EXISTS (SELECT * FROM sys.objects so JOIN sys.schemas sc ON so.schema_id =
sc.schema_id WHERE so.name = N'PK_mediafiles_id'  AND sc.name = N'fresh'  AND type
in (N'PK'))
ALTER TABLE [fresh].[mediafiles] DROP CONSTRAINT [PK_mediafiles_id]
 GO
```

```
ALTER TABLE [fresh].[mediafiles]
 ADD CONSTRAINT [PK_mediafiles_id]
   PRIMARY KEY
   CLUSTERED ([id] ASC)

GO

ALTER TABLE  [fresh].[mediafiles]
 ADD DEFAULT NULL FOR [name]
GO

ALTER TABLE  [fresh].[mediafiles]
 ADD DEFAULT NULL FOR [description]
GO

ALTER TABLE  [fresh].[mediafiles]
 ADD DEFAULT NULL FOR [link]
GO
```

Таблиця Mediafilesgroup:

```
IF EXISTS (SELECT * FROM sys.objects so JOIN sys.schemas sc ON so.schema_id =
sc.schema_id WHERE so.name = N'mediafilesgroup'  AND sc.name = N'fresh'  AND type
in (N'U'))
BEGIN

  DECLARE @drop_statement nvarchar(500)

  DECLARE drop_cursor CURSOR FOR
      SELECT 'alter table '+quotename(schema_name(ob.schema_id))+
      '.'+quotename(object_name(ob.object_id))+ ' drop constraint ' +
quotename(fk.name)
      FROM sys.objects ob INNER JOIN sys.foreign_keys fk ON fk.parent_object_id =
ob.object_id
      WHERE fk.referenced_object_id =
          (
             SELECT so.object_id
             FROM sys.objects so JOIN sys.schemas sc
             ON so.schema_id = sc.schema_id
             WHERE so.name = N'mediafilesgroup'  AND sc.name = N'fresh'  AND type
in (N'U'))

  OPEN drop_cursor

  FETCH NEXT FROM drop_cursor
  INTO @drop_statement

  WHILE @@FETCH_STATUS = 0
  BEGIN
     EXEC (@drop_statement)

     FETCH NEXT FROM drop_cursor
     INTO @drop_statement
  END

  CLOSE drop_cursor
  DEALLOCATE drop_cursor

  DROP TABLE [fresh].[mediafilesgroup]
END
```

```
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE
[fresh].[mediafilesgroup]
(
   [id] int IDENTITY(4620, 1)  NOT NULL,
   [designType] int  NOT NULL,
   [positionType] int  NOT NULL
)
WITH (DATA_COMPRESSION = NONE)
GO
IF EXISTS (SELECT * FROM sys.objects so JOIN sys.schemas sc ON so.schema_id =
sc.schema_id WHERE so.name = N'PK_mediafilesgroup_id'  AND sc.name = N'fresh'  AND
type in (N'PK'))
ALTER TABLE [fresh].[mediafilesgroup] DROP CONSTRAINT [PK_mediafilesgroup_id]
 GO



ALTER TABLE [fresh].[mediafilesgroup]
 ADD CONSTRAINT [PK_mediafilesgroup_id]
   PRIMARY KEY
   CLUSTERED ([id] ASC)

GO
```

Таблиця News:

```
IF EXISTS (SELECT * FROM sys.objects so JOIN sys.schemas sc ON so.schema_id =
sc.schema_id WHERE so.name = N'news'  AND sc.name = N'fresh'  AND type in (N'U'))
BEGIN

  DECLARE @drop_statement nvarchar(500)

  DECLARE drop_cursor CURSOR FOR
      SELECT 'alter table '+quotename(schema_name(ob.schema_id))+
      '.'+quotename(object_name(ob.object_id))+ ' drop constraint ' +
quotename(fk.name)
      FROM sys.objects ob INNER JOIN sys.foreign_keys fk ON fk.parent_object_id =
ob.object_id
      WHERE fk.referenced_object_id =
          (
             SELECT so.object_id
             FROM sys.objects so JOIN sys.schemas sc
             ON so.schema_id = sc.schema_id
             WHERE so.name = N'news'  AND sc.name = N'fresh'  AND type in (N'U')
           )

  OPEN drop_cursor

  FETCH NEXT FROM drop_cursor
  INTO @drop_statement

  WHILE @@FETCH_STATUS = 0
  BEGIN
     EXEC (@drop_statement)

     FETCH NEXT FROM drop_cursor
     INTO @drop_statement
```

```
  END

  CLOSE drop_cursor
  DEALLOCATE drop_cursor

  DROP TABLE [fresh].[news]
END
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE
[fresh].[news]
(
   [nid] int IDENTITY(12639, 1)  NOT NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_ukrainian_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR according to character set
mapping for cp1251 character set
   */

   [topic] varchar(255)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_ukrainian_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [title] varchar(max)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_ukrainian_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [content] varchar(max)  NULL,
   [date] int  NULL,
   [archive] int  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_general_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
```

```
set mapping for cp1251 character set
   */

   [titleEN] varchar(max)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_general_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [titleRU] varchar(max)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_general_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [contentEN] varchar(max)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_general_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR(MAX) according to character
set mapping for cp1251 character set
   */

   [contentRU] varchar(max)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_general_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR according to character set
mapping for cp1251 character set
   */

   [tagGroup] varchar(255)  NULL,

   /*
   *   SSMA warning messages:
   *   M2SS0183: The following SQL clause was ignored during conversion: COLLATE
cp1251_general_ci.

   *   SSMA informational messages:
   *   M2SS0055: Data type was converted to VARCHAR according to character set
mapping for cp1251 character set
   */
```

```
   [teazer] varchar(255)  NULL,
   [pictGroup] int  NULL,
   [vidGroup] int  NULL,
   [docGroup] int  NULL,
   [hidden] smallint  NOT NULL,
   [lastChangeID] int  NULL,
   [hiddenFilesGroup] int  NULL)
WITH (DATA_COMPRESSION = NONE)
GO
IF EXISTS (SELECT * FROM sys.objects so JOIN sys.schemas sc ON so.schema_id =
sc.schema_id WHERE so.name = N'PK_news_nid'  AND sc.name = N'fresh'  AND type in
(N'PK'))
ALTER TABLE [fresh].[news] DROP CONSTRAINT [PK_news_nid]
 GO



ALTER TABLE [fresh].[news]
 ADD CONSTRAINT [PK_news_nid]
   PRIMARY KEY
   CLUSTERED ([nid] ASC)

GO

ALTER TABLE  [fresh].[news]
 ADD DEFAULT NULL FOR [topic]
GO

ALTER TABLE  [fresh].[news]
 ADD DEFAULT NULL FOR [date]
GO

ALTER TABLE  [fresh].[news]
 ADD DEFAULT NULL FOR [archive]
GO

ALTER TABLE  [fresh].[news]
 ADD DEFAULT NULL FOR [tagGroup]
GO

ALTER TABLE  [fresh].[news]
 ADD DEFAULT NULL FOR [teazer]
GO

ALTER TABLE  [fresh].[news]
 ADD DEFAULT NULL FOR [pictGroup]
GO

ALTER TABLE  [fresh].[news]
 ADD DEFAULT NULL FOR [vidGroup]
GO

ALTER TABLE  [fresh].[news]
 ADD DEFAULT NULL FOR [docGroup]
GO

ALTER TABLE  [fresh].[news]
 ADD DEFAULT NULL FOR [lastChangeID]
GO

ALTER TABLE  [fresh].[news]
 ADD DEFAULT NULL FOR [hiddenFilesGroup]
GO
```

ДОДАТОК 3

Детальний журнал порівняння вихідної та цільової баз даних

```
2023.04.09 22:31:13 Compare from: server=127.0.0.1;uid=root;pwd=****;database=fresh
2023.04.09 22:31:13 Compare to: Server=KIF\SQLEXPRESS02; Database=testdata;Integrated Security=SSPI;Encrypt=False
2023.04.09 22:31:13 Using a maximum of 16 threads
2023.04.09 22:31:14 Total Tables to Process (from target): 41
2023.04.09 22:31:14 (0) Processing table (1): fresh.mediafiles
2023.04.09 22:31:14 (0) Row counts (32155) match on both source and target for table: fresh.mediafiles
2023.04.09 22:31:14 (1) Processing table (2): fresh.history
2023.04.09 22:31:14 (1) Row counts (31118) match on both source and target for table: fresh.history
2023.04.09 22:31:14 (2) Processing table (3): fresh.newshistory
2023.04.09 22:31:14 (2) Row counts (15683) match on both source and target for table: fresh.newshistory
2023.04.09 22:31:14 (3) Processing table (4): fresh.news
2023.04.09 22:31:14 (3) Row counts (10734) match on both source and target for table: fresh.news
2023.04.09 22:31:14 (4) Processing table (5): fresh.mediafilesgroup
2023.04.09 22:31:15 (4) Row counts (4588) match on both source and target for table: fresh.mediafilesgroup
2023.04.09 22:31:15 (5) Processing table (6): fresh.guests_umz
2023.04.09 22:31:15 (5) Row counts (513) match on both source and target for table: fresh.guests_umz
2023.04.09 22:31:15 (4) Processing of fresh.mediafilesgroup complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 4я588 Total Time: 0,3 Row Count Time: 0,1 Compare Time: 0,1
2023.04.09 22:31:15 (4) Processing table (7): fresh.addresslist
2023.04.09 22:31:15 (4) Row counts (198) match on both source and target for table: fresh.addresslist
2023.04.09 22:31:15 (5) Error - Hash Mismatch(es) on table: fresh.guests_umz
2023.04.09 22:31:15 (6) Processing table (8): fresh.guests
2023.04.09 22:31:15 (6) Row counts (134) match on both source and target for table: fresh.guests
2023.04.09 22:31:15 (1) Processing of fresh.history complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 31я118 Total Time: 0,9 Row Count Time: 0,0 Compare Time: 0,9
2023.04.09 22:31:15 (1) Processing table (9): fresh.news_tmp
2023.04.09 22:31:15 (1) Row counts (132) match on both source and target for table: fresh.news_tmp
2023.04.09 22:31:15 (6) Error - Hash Mismatch(es) on table: fresh.guests
2023.04.09 22:31:15 (4) Processing of fresh.addresslist complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 198 Total Time: 0,1 Row Count Time: 0,0 Compare Time: 0,1
2023.04.09 22:31:15 (4) Processing table (10): fresh.alumni
2023.04.09 22:31:15 (4) Row counts (32) match on both source and target for table: fresh.alumni
```

```
2023.04.09 22:31:15 (4) Error - Hash Mismatch(es) on table: fresh.alumni
2023.04.09 22:31:15 (3) Error - Hash Mismatch(es) on table: fresh.news
2023.04.09 22:31:15 (7) Row counts (29) match on both source and target for table: fresh.umzphoto
2023.04.09 22:31:15 (1) Error - Hash Mismatch(es) on table: fresh.news_tmp
2023.04.09 22:31:16 (4) Processing of fresh.alumni complete - Errors/Warnings Detected: 1/0 -
    Compared Rows: 32 Total Time: 0,2 Row Count Time: 0,0 Compare Time: 0,2
2023.04.09 22:31:16 (4) Processing table (12): dbo.AspNetUserRoles
2023.04.09 22:31:16 (4) Exception on getting source row count: Table 'fresh.aspnetuserroles' doesn't exist
2023.04.09 22:31:16 (4) Error on processing of dbo.AspNetUserRoles - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
2023.04.09 22:31:16 (4) Processing table (13): dbo.AspNetUsers
2023.04.09 22:31:16 (4) Exception on getting source row count: Table 'fresh.aspnetusers' doesn't exist
2023.04.09 22:31:16 (4) Error on processing of dbo.AspNetUsers - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
2023.04.09 22:31:16 (4) Processing table (14): fresh.umz_topics_all
2023.04.09 22:31:16 (4) Row counts (12) match on both source and target for table: fresh.umz_topics_all
2023.04.09 22:31:16 (0) Processing of fresh.mediafiles complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 32я155 Total Time: 1,5 Row Count Time: 0,0 Compare Time: 1,4
2023.04.09 22:31:16 (0) Processing table (15): newseditors.users
2023.04.09 22:31:16 (0) Exception on getting source row count: Table 'fresh.users' doesn't exist
2023.04.09 22:31:16 (0) Error on processing of newseditors.users - Errors/Warnings Detected: 1/0 - Error: Error getting row count from source.
2023.04.09 22:31:16 (0) Processing table (16): fresh.umzguest
2023.04.09 22:31:16 (0) Row counts (8) match on both source and target for table: fresh.umzguest
2023.04.09 22:31:16 (4) Processing of fresh.umz_topics_all complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 12 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (4) Processing table (17): fresh.votes_all
2023.04.09 22:31:16 (4) Row counts (6) match on both source and target for table: fresh.votes_all
2023.04.09 22:31:16 (0) Processing of fresh.umzguest complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 8 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (0) Processing table (18): fresh.news_umz
2023.04.09 22:31:16 (4) Error - Hash Mismatch(es) on table: fresh.votes_all
2023.04.09 22:31:16 (0) Row counts (5) match on both source and target for table: fresh.news_umz
```

```
2023.04.09 22:31:16 (7) Processing of fresh.umzphoto complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 29 Total Time: 0,1 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (7) Processing table (19): fresh.topics_all
2023.04.09 22:31:16 (0) Error - Hash Mismatch(es) on table: fresh.news_umz
2023.04.09 22:31:16 (7) Row counts (4) match on both source and target for table: fresh.topics_all
2023.04.09 22:31:16 (0) Processing of fresh.news_umz complete - Errors/Warnings Detected: 1/0 -
    Compared Rows: 5 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (0) Processing table (20): fresh.photos_all
2023.04.09 22:31:16 (0) Row counts (3) match on both source and target for table: fresh.photos_all
2023.04.09 22:31:16 (4) Processing of fresh.votes_all complete - Errors/Warnings Detected: 5/0 -
    Compared Rows: 6 Total Time: 0,1 Row Count Time: 0,0 Compare Time: 0,1
2023.04.09 22:31:16 (4) Processing table (21): fresh.photos_link
2023.04.09 22:31:16 (4) Row counts (3) match on both source and target for table: fresh.photos_link
2023.04.09 22:31:16 (0) Processing of fresh.photos_all complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 3 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (0) Processing table (22): fresh.users_all
2023.04.09 22:31:16 (0) Row counts (3) match on both source and target for table: fresh.users_all
2023.04.09 22:31:16 (4) Warning on processing of fresh.photos_link - Errors/Warnings Detected: 0/1 -
    Warning: Tables have the same rowcount, but do not have a PK/rowguid column(s). No additional comparison is possible.
2023.04.09 22:31:16 (4) Processing table (23): dbo.__EFMigrationsHistory
2023.04.09 22:31:16 (4) Exception on getting source row count: Table 'fresh.__efmigrationshistory' doesn't exist
2023.04.09 22:31:16 (4) Error on processing of dbo.__EFMigrationsHistory - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
2023.04.09 22:31:16 (4) Processing table (24): dbo.AspNetRoles
2023.04.09 22:31:16 (4) Exception on getting source row count: Table 'fresh.aspnetroles' doesn't exist
2023.04.09 22:31:16 (0) Processing of fresh.users_all complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 3 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (0) Processing table (25): fresh.groups_all
2023.04.09 22:31:16 (0) Row counts (2) match on both source and target for table: fresh.groups_all
2023.04.09 22:31:16 (4) Error on processing of dbo.AspNetRoles - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
2023.04.09 22:31:16 (4) Processing table (26): fresh.photo_series_all
```

```
2023.04.09 22:31:16 (4) Row counts (2) match on both source and target for table: fresh.photo_series_all
2023.04.09 22:31:16 (0) Processing of fresh.groups_all complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 2 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (0) Processing table (27): fresh.stats_day
2023.04.09 22:31:16 (0) Row counts (2) match on both source and target for table: fresh.stats_day
2023.04.09 22:31:16 (4) Processing of fresh.photo_series_all complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 2 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (4) Processing table (28): fresh.photo_topics_all
2023.04.09 22:31:16 (4) Row counts (1) match on both source and target for table: fresh.photo_topics_all
2023.04.09 22:31:16 (2) Error - Hash Mismatch(es) on table: fresh.newshistory
2023.04.09 22:31:16 (0) Processing of fresh.stats_day complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 2 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (0) Processing table (29): fresh.news_link
2023.04.09 22:31:16 (0) Row counts (1) match on both source and target for table: fresh.news_link
2023.04.09 22:31:16 (0) Warning on processing of fresh.news_link - Errors/Warnings Detected: 0/1 -
    Warning: Tables have the same rowcount, but do not have a PK/rowguid column(s). No additional comparison is possible.
2023.04.09 22:31:16 (0) Processing table (30): fresh.polls_all
2023.04.09 22:31:16 (0) Row counts (1) match on both source and target for table: fresh.polls_all
2023.04.09 22:31:16 (4) Processing of fresh.photo_topics_all complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 1 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (4) Processing table (31): fresh.umz
2023.04.09 22:31:16 (4) Row counts (0) match on both source and target for table: fresh.umz
2023.04.09 22:31:16 (0) Error - Hash Mismatch(es) on table: fresh.polls_all
2023.04.09 22:31:16 (6) Processing of fresh.guests complete - Errors/Warnings Detected: 44/0 -
    Compared Rows: 134 Total Time: 0,8 Row Count Time: 0,0 Compare Time: 0,7
2023.04.09 22:31:16 (6) Processing table (32): fresh.photos
2023.04.09 22:31:16 (6) Row counts (0) match on both source and target for table: fresh.photos
2023.04.09 22:31:16 (0) Processing of fresh.polls_all complete - Errors/Warnings Detected: 1/0 -
    Compared Rows: 1 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (0) Processing table (33): fresh.news_all
2023.04.09 22:31:16 (0) Row counts are different between source (1) and target (0) on table: fresh.news_all
2023.04.09 22:31:16 (0) Error - Matching row(s) missing in table: `news_all`
```

```
2023.04.09 22:31:16 (4) Processing of fresh.umz complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 0 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (4) Processing table (34): dbo.AspNetUserTokens
2023.04.09 22:31:16 (4) Exception on getting source row count: Table 'fresh.aspnetusertokens' doesn't exist
2023.04.09 22:31:16 (0) Processing of fresh.news_all complete - Errors/Warnings Detected: 1/0 -
    Compared Rows: 0 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (0) Processing table (35): dbo.AspNetRoleClaims
2023.04.09 22:31:16 (0) Exception on getting source row count: Table 'fresh.aspnetroleclaims' doesn't exist
2023.04.09 22:31:16 (0) Error on processing of dbo.AspNetRoleClaims - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
2023.04.09 22:31:16 (0) Processing table (36): dbo.AspNetUserClaims
2023.04.09 22:31:16 (0) Exception on getting source row count: Table 'fresh.aspnetuserclaims' doesn't exist
2023.04.09 22:31:16 (0) Error on processing of dbo.AspNetUserClaims - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
2023.04.09 22:31:16 (0) Processing table (37): dbo.AspNetUserLogins
2023.04.09 22:31:16 (0) Exception on getting source row count: Table 'fresh.aspnetuserlogins' doesn't exist
2023.04.09 22:31:16 (0) Error on processing of dbo.AspNetUserLogins - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
```

```
2023.04.09 22:31:16 (0) Processing table (38): newseditors.roles
2023.04.09 22:31:16 (0) Exception on getting source row count: Table 'fresh.roles' doesn't exist
2023.04.09 22:31:16 (0) Error on processing of newseditors.roles - Errors/Warnings Detected: 1/0 - Error:
    Error getting row count from source.
2023.04.09 22:31:16 (0) Processing table (39): newseditors.userclaims
2023.04.09 22:31:16 (0) Exception on getting source row count: Table 'fresh.userclaims' doesn't exist
2023.04.09 22:31:16 (0) Error on processing of newseditors.userclaims - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
2023.04.09 22:31:16 (0) Processing table (40): newseditors.userlogins
2023.04.09 22:31:16 (0) Exception on getting source row count: Table 'fresh.userlogins' doesn't exist
2023.04.09 22:31:16 (0) Error on processing of newseditors.userlogins - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
2023.04.09 22:31:16 (0) Processing table (41): newseditors.userroles
```

```
2023.04.09 22:31:16 (0) Exception on getting source row count: Table 'fresh.userroles' doesn't exist
2023.04.09 22:31:16 (0) Error on processing of newseditors.userroles - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
2023.04.09 22:31:16 (4) Error on processing of dbo.AspNetUserTokens - Errors/Warnings Detected: 1/0 -
    Error: Error getting row count from source.
2023.04.09 22:31:16 (6) Processing of fresh.photos complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 0 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:16 (7) Processing of fresh.topics_all complete - Errors/Warnings Detected: 0/0 -
    Compared Rows: 4 Total Time: 0,0 Row Count Time: 0,0 Compare Time: 0,0
2023.04.09 22:31:17 (1) Processing of fresh.news_tmp complete - Errors/Warnings Detected: 127/0 -
    Compared Rows: 132 Total Time: 1,3 Row Count Time: 0,0 Compare Time: 1,3
2023.04.09 22:31:17 (5) Processing of fresh.guests_umz complete - Errors/Warnings Detected: 151/0 -
    Compared Rows: 513 Total Time: 1,5 Row Count Time: 0,0 Compare Time: 1,5
2023.04.09 22:32:55 (3) Processing of fresh.news complete - Errors/Warnings Detected: 10я351/0 -
    Compared Rows: 10я734 Total Time: 100,4 Row Count Time: 0,0 Compare Time: 100,4
2023.04.09 22:33:30 (2) Processing of fresh.newshistory complete - Errors/Warnings Detected: 11я908/0 -
    Compared Rows: 15я683 Total Time: 135,5 Row Count Time: 0,0 Compare Time: 135,5
2023.04.09 22:33:30 Databases Check Complete - Tables: 41 Errors/Warnings Detected: 22я603/2
    Compared Rows: 95я365 Elapsed Time: 00:02:17 Total Processing Time: 243,1 Row Count Time: 0,4 Compare Time: 242,2 Rows/Second: 695,14
```

ДОДАТОК 4

Програмний код сервісу для роботи з Facebook

```
using core_news_admin.Models.DataModels;
using core_news_admin.Services.FacebookCore;
using Microsoft.EntityFrameworkCore;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace core_news_admin.Services.Facebook
{
    public class FacebookService
    {
        private DatabaseContext db;
        private FacebookClient facebookClient;

        public FacebookService(DatabaseContext db, string clientId, string clientSecret)
        {
            this.db = db;
            this.facebookClient = new FacebookClient(clientId, clientSecret);
        }

        public Task<List<FacebookNews>> GetAll()
        {
            return db.FacebookNews.ToListAsync();
        }

        public Task<FacebookNews> GetById(int nid)
        {
            return db.FacebookNews.FirstOrDefaultAsync(n => n.Id == nid);
        }
        public Task<FacebookNews> GetByOriginalNewsId(string originalNewsId)
        {
            return db.FacebookNews.FirstOrDefaultAsync(n => n.OriginalNewsId ==
originalNewsId);
        }

        public async Task<FacebookNews> Save(FacebookNews news)
        {
            await db.FacebookNews.AddAsync(news);
            await db.SaveChangesAsync();
            return await db.FacebookNews.FirstOrDefaultAsync(n => n.Id == news.Id);
        }

        public async Task<FacebookNews> CreateOnFacebook(int nid)
        {
            FacebookNews newsToPublish = await db.FacebookNews.FirstOrDefaultAsync(n => n.Id
== nid);
            if (newsToPublish == null)
            {
                throw new ArgumentException($"News with id {nid} is not existing.");
            }
            JObject result = await facebookClient.PostAsync(newsToPublish.TextContext);
            string pagePostId = result["id"].ToString();
            newsToPublish.PagePostId = pagePostId;
            newsToPublish.Saved = true;
            await db.SaveChangesAsync();
            return await db.FacebookNews.FirstOrDefaultAsync(n => n.Id == newsToPublish.Id);
```

```
        }

        public async Task<FacebookNews> PublishOnFacebook(int nid)
        {
            FacebookNews newsToPublish = await db.FacebookNews.FirstOrDefaultAsync(n => n.Id
== nid);
            if (newsToPublish == null)
            {
                throw new ArgumentException($"News with id {nid} is not existing.");
            }
            JObject result = await facebookClient.PublishAsync(newsToPublish.TextContext);
            newsToPublish.Hidden = false;
            await db.SaveChangesAsync();
            return await db.FacebookNews.FirstOrDefaultAsync(n => n.Id == newsToPublish.Id);
        }

        public async Task<FacebookNews> HideFromFacebook(int nid)
        {
            FacebookNews newsToPublish = await db.FacebookNews.FirstOrDefaultAsync(n => n.Id
== nid);
            if (newsToPublish == null)
            {
                throw new ArgumentException($"News with id {nid} is not existing.");
            }
            JObject result = await facebookClient.HideAsync(newsToPublish.PagePostId);
            newsToPublish.Hidden = true;
            await db.SaveChangesAsync();
            return await db.FacebookNews.FirstOrDefaultAsync(n => n.Id == newsToPublish.Id);
        }

        public async Task<FacebookNews> DeleteFromFacebook(int nid)
        {
            FacebookNews newsToPublish = await db.FacebookNews.FirstOrDefaultAsync(n => n.Id
== nid);
            if (newsToPublish == null)
            {
                throw new ArgumentException($"News with id {nid} is not existing.");
            }
            JObject result = await facebookClient.DeleteAsync(newsToPublish.PagePostId);
            newsToPublish.Hidden = true;
            newsToPublish.Saved = false;
            await db.SaveChangesAsync();
            return await db.FacebookNews.FirstOrDefaultAsync(n => n.Id == newsToPublish.Id);
        }

    }
}
```