Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Інститут прикладної математики та фундаментальних наук

Кафедра прикладної математики

Курсова робота

з курсу «Надвеликі бази даних»

за темою «Розробка та аналіз надвеликої бази даних з використанням технологій Microsoft SQL Server»

Виконала:

студентка групи ПМ-42

Марія РІЗНИК

Перевірив:

Богдан ЛЮБІНСЬКІЙ

_____

(дата) (підпис викладача)

Львів – 2026 р.

Анотація

Курсова робота присвячена розробці та аналізу надвеликої бази даних для предметної області кабельного телебачення з використанням технологій Microsoft SQL Server. У роботі проведено аналіз предметної області та спроєктовано реляційну базу даних, нормалізовану до третьої нормальної форми. Для моделювання великих обсягів даних згенеровано тестовий набір, що включає понад 100 000 абонентів і більше 1 000 000 замовлень кінофільмів, із застосуванням інструменту Redgate SQL Data Generator. Реалізовано ETL-процеси засобами SQL Server Integration Services та побудовано багатовимірний OLAP-куб за допомогою SQL Server Analysis Services для аналітичної обробки даних і формування звітів.

Annotatiom

This course paper is devoted to the development and analysis of a very large database for the cable television domain using Microsoft SQL Server technologies. The study includes an analysis of the subject area and the design of a relational database normalized to the third normal form. To simulate large data volumes, a test dataset containing more than 100,000 subscribers and over 1,000,000 movie orders was generated using the Redgate SQL Data Generator tool. ETL processes were implemented using SQL Server Integration Services, and a multidimensional OLAP cube was built with SQL Server Analysis Services to support analytical data processing and reporting.

Зміст

Вступ

Тема курсової роботи — «Розробка та аналіз надвеликої бази даних з використанням технологій Microsoft SQL Server» — є актуальною з огляду на те, що дедалі більше організацій переходять від невеликих облікових систем до інформаційних платформ, здатних накопичувати та обробляти великі масиви даних у режимі, близькому до реального часу. У таких умовах від бази даних очікують не лише коректного зберігання інформації, а й забезпечення швидкого доступу для аналітики, побудови звітів, прогнозування попиту, оцінки фінансових показників та підтримки управлінських рішень. Саме тому в межах роботи важливо поєднати класичне проектування реляційної бази даних із практиками підготовки даних до аналітичних задач, включно з ETL-процесами та подальшим використанням OLAP-технологій.

Обрана предметна область відповідає варіанту «Кабельне телебачення» і охоплює типову діяльність провайдера, який надає абонентам доступ до пакетів телеканалів та додаткову послугу замовлення кінофільмів. Для такої сфери характерні стабільні, але великі за обсягом довідники (канали, жанри, категорії, статуси), а також дуже значні за кількістю операційні дані: база абонентів, історія підключень, рахунки, платежі та замовлення. У роботі поставлено вимоги, які одразу задають масштаб системи: не менше 100 000 абонентів, не менше 1 000 000 замовлень фільмів і наявність історичних даних щонайменше за 5 років. Це означає, що при проектуванні необхідно враховувати не лише логіку предметної області, а й ефективність моделі, механізми цілісності, індексацію та підготовку структури до формування аналітичних зрізів.

Практична цінність такої системи полягає в тому, що провайдеру важливо розуміти, які фільми найбільш популярні, як змінюються вподобання абонентів, які пакети каналів дають найбільшу виручку, як формується фінансовий результат у розрізі місяців, і як змінюється абонентська база протягом часу. У технічному завданні це зафіксовано у вигляді обов’язкових звітів: рейтинг популярності кінофільмів, споживча орієнтація абонентів,

фінансовий звіт за місяць, аналіз підключень каналів та динаміка абонентської бази. Відповідно, база даних повинна містити атрибути та зв’язки, що дозволяють будувати такі звіти без «ручних» доробок, а також забезпечувати достатню деталізацію даних для подальшого створення вимірів і мір у кубі.

Постановка задачі

Тема: Розробка та аналіз надвеликої бази даних з використанням технологій Microsoft SQL Server.

Мета курсової роботи: Розробити повнофункціональну систему керування надвеликою базою даних для обраної предметної області з реалізацією ETL-процесів, побудовою багатовимірного куба та створенням аналітичних звітів з використанням технологій Microsoft SQL Server (SSIS, SSAS, SSRS).

Предметна область

Варіант 10: "Кабельне телебачення"

Основні сутності: Абоненти, Канали, Групи каналів, Кінофільми, Замовлення, Рахунки, Платежі

Специфічні вимоги:

• Мінімум 100 000 абонентів

• Мінімум 1 000 000 замовлень фільмів

• Історія за 5 років

Обов'язкові звіти:

1. Рейтинг популярності кінофільмів
2. Споживча орієнтація абонентів
3. Фінансовий звіт за місяць
4. Аналіз підключень каналів
5. Динаміка абонентської бази

Етап 1. Аналіз предметної області та проектування бази даних

1.1 Аналіз предметної області

У межах варіанту «Кабельне телебачення» я розглядала провайдера як організацію, що одночасно виконує роль постачальника контенту та білінгового центру. У такій предметній області є дві ключові лінії послуг. Перша — це підписка на пакети телеканалів, де абонент обирає певну групу каналів і щомісяця сплачує абонентську плату. Друга — це відео за запитом (VOD), коли абонент у будь-який момент може замовити перегляд конкретного фільму, сплачуючи разову вартість. Для провайдера важливо мати історію взаємодії абонента з послугами: коли він підключився, які пакети обирав у різні періоди, які фільми замовляв, як формувалися рахунки та яким чином надходили платежі. Саме ця історичність і є основою для аналітики, оскільки дозволяє аналізувати динаміку попиту та зміни поведінки споживачів у часі.

Опис предметної області я формувала так, щоб дані були одразу придатними для побудови обов'язкових звітів. Для рейтингу популярності фільмів необхідно мати каталог фільмів та факти замовлень із датою/часом і сумою оплати. Для звіту про споживчу орієнтацію абонентів потрібно пов'язати абонента з жанрами або категоріями контенту, що вимагає наявності таблиці жанрів і зв'язку фільму з жанром (включно з випадком, коли фільм належить до кількох жанрів). Для фінансового звіту за місяць потрібні щомісячні рахунки, деталізація їхніх складників і фактичні платежі, щоб бачити нарахування, оплату та залишок боргу. Для аналізу підключень каналів необхідно моделювати канали, їх категорії, пакети каналів та історію підключення пакетів абонентами. Для динаміки абонентської бази потрібно мати дати підключення та відключення абонентів, а також статуси, що дозволяють відрізняти активних, призупинених і закритих.

Після деталізації предметної області я визначила основні бізнес-процеси, які мають відображатися в базі даних. Процес обслуговування абонента починається з реєстрації договору та створення облікового запису абонента, після чого абонент підключає пакет каналів. Упродовж часу абонент

може змінювати пакет, тобто відбувається завершення старого підключення та початок нового, а в кожний момент часу має існувати щонайбільше один активний пакет. Паралельно абонент може здійснювати VOD-замовлення фільмів, що формує записи в журналі замовлень. Наприкінці кожного місяця (або за встановленою логікою) формується рахунок, у який входить абонплата за активний пакет, а також суми за VOD-замовлення у відповідному періоді. Далі абонент здійснює платежі за рахунком одним із доступних способів оплати, і система повинна відображати зміну статусу рахунку, суму сплаченого та поточний баланс.

На підставі бізнес-процесів я сформулювала функціональні вимоги до системи зберігання даних. Система повинна забезпечувати збереження великих обсягів абонентів і подій, дозволяти швидко отримувати дані у розрізі абонента, місяця, фільму, жанру, пакету каналів, а також підтримувати коректне формування рахунків і оплат. Важливим є контроль цілісності, щоб не виникало ситуацій із «висячими» посиланнями: замовлення не може існувати без абонента і фільму, платіж не може існувати без рахунку, а канал повинен належати до певної категорії. Оскільки в роботі акцент зроблено на подальшій аналітиці, я також вимагала, щоб структура дозволяла легко будувати виміри часу та атрибути для сегментації абонентів, а також забезпечувала відтворюваність даних при тестуванні ETL.

Після цього я сформулювала бізнес-правила та обмеження, які мають бути реалізовані на рівні бази даних. Насамперед це правила дат для абонента: дата відключення не може бути раніше дати підключення, і для активних абонентів дата відключення може бути відсутньою. Аналогічне правило потрібне для історії підписок: дата завершення підписки не може бути раніше дати початку, а активна підписка має мати порожню дату завершення. Дуже важливим бізнес-правилом є «лише одна активна підписка на абонента» у будь-який момент часу, що не завжди можна описати лише унікальним ключем, тому потребує тригерної логіки або складнішого обмеження. Для рахунків і платежів важливо, щоб сума платежу була додатною, а баланс

розраховувався коректно як різниця між нарахованим і сплаченим. Для довідників, таких як статуси, методи оплати, категорії та жанри, важливо забезпечити стабільність значень, оскільки вони будуть використовуватися як виміри в кубі та візуалізаціях.

У результаті аналізу предметної області я визначила набір основних сутностей і погодила їхній склад із вимогами варіанту: абоненти, канали, групи каналів, кінофільми, замовлення, рахунки та платежі. Додатково я ввела довідникові сутності для статусів, методів оплати, жанрів і категорій каналів, а також проміжні таблиці для зв'язків «багато-до-багатьох» і для ведення історії підписок.

1.2 Концептуальне проектування

На етапі концептуального проектування я побудувала ER-модель, спираючись на логіку предметної області та майбутні аналітичні вимоги. Я використовувала підхід, близький до нотації Crow's Foot, оскільки вона добре відображає кардинальності та обов'язковість зв'язків. Центральною сутністю концептуальної моделі є «Абонент», від якого відходять ключові бізнес-події: підписки на пакети каналів, замовлення фільмів і білінгові операції. Така структура відповідає реальній роботі провайдера і дозволяє в подальшому будувати звіти як у розрізі одного абонента, так і для всієї бази.

Сутність «Канал» я представила як елемент довідника, що має назву та належить до категорії. Категорія каналів — окрема довідникова сутність, яка потрібна не лише для упорядкування, а й для аналітики підключень, адже звіт про підключення каналів зручніше будувати за категоріями. Далі я ввела сутність «Група каналів» як пакет послуг із щомісячною абонплатою. Оскільки один пакет містить багато каналів, а один канал може входити до кількох пакетів, я передбачила проміжну сутність (асоціативну таблицю) для зв'язку пакетів і каналів. Це рішення відповідає нормалізованому підходу і полегшує зміну складу пакетів у майбутньому.

Для історії підписок я ввела сутність, що пов'язує абонента з пакетом каналів та містить дати початку і завершення. На концептуальному рівні це є

«факт» підключення пакету, який повторюється у часі, і на базі якого можна будувати аналіз зміни пакетів, тривалості користування та динаміки абонплати.

Сутність «Кінофільм» описує контент VOD і містить назву, рік виходу та вартість. Жанри винесені в окрему сутність, а зв'язок «фільм—жанр» реалізовано як «багато-до-багатьох», оскільки фільм може бути віднесений до кількох жанрів, а жанр включає багато фільмів. Це принципово важливо для майбутнього звіту про споживчу орієнтацію, де я зможу сегментувати абонентів за жанровими вподобаннями.

Замовлення фільму я описала як окрему сутність події, що пов'язує абонента та фільм, містить дату/час і фактичну суму оплати. Саме ця сутність стане основою для розрахунку популярності фільмів і для побудови часових трендів. Концептуально замовлення є «фактичною» сутністю, що має чіткі виміри: час, абонент, фільм, жанр (через фільм), а також може бути пов'язане з рахунками через механізм деталізації нарахувань.

Білінг я концептуалізувала через сутність «Рахунок» і сутність «Платіж». Рахунок є щомісячним підсумком нарахувань абонента, а платіж — подією оплати рахунку. Для більшої прозорості я також передбачила деталізацію рахунку через «рядки рахунку», де можна зберігати розкладку суми на абонплату і VOD-замовлення. Це рішення допомагає як для аудитної прозорості, так і для аналітики, оскільки дозволяє формувати звіти не лише за загальною сумою, а й за структурою доходів.

Кардинальності зв'язків я визначила так, щоб вони відповідали реальному часу й одночасно не блокували історичність. Абонент може мати багато підписок у різні періоди, але в один момент часу не більше однієї активної. Абонент може робити багато замовлень фільмів, кожне замовлення належить одному абоненту і одному фільму. Абонент може мати багато рахунків, типово один рахунок на місяць, а кожен рахунок належить одному абоненту. Рахунок може мати багато рядків деталізації, а кожен рядок належить одному рахунку. Рахунок може мати багато платежів (часткова оплата), а кожен

платіж належить одному рахунку. Кожен платіж має метод оплати, що визначається довідником методів. Кожен рахунок має статус з довідника статусів рахунку, а абонент має статус з довідника статусів абонента.

1.3 Логічне проектування

Після концептуальної моделі я перейшла до логічного проектування, де основною ціллю була нормалізація структури мінімум до третьої нормальної форми. Я зосередилася на усуненні надлишковості та забезпеченні того, щоб атрибути залежали від ключа, повного ключа і тільки від ключа. Наприклад, категорії каналів, жанри, статуси та методи оплати винесені у довідникові таблиці, а в основних таблицях зберігаються лише зовнішні ключі. Це прибирає дублювання текстових значень і забезпечує стабільність вимірів для аналітики.

Зв’язки «багато-до-багатьох» я реалізувала через окремі проміжні таблиці. Для пакетів і каналів це таблиця, яка містить лише ідентифікатори пакету та каналу, що відповідає класичному підходу нормалізації. Аналогічно для зв’язку «фільм—жанр» створено окрему таблицю, яка дозволяє фільму мати кілька жанрів без дублювання даних у самій таблиці фільмів. Історія підписок абонента є окремою таблицею з атрибутами періоду дії, що дозволяє коректно відображати зміну пакетів у часі.

У процесі логічного проектування я визначила первинні ключі для всіх сутностей. Для основних довідників і основних таблиць я використовувала сурогатні ключі типу Identity, оскільки вони забезпечують компактні індекси та стабільні посилання. Для проміжних таблиць зв’язків доцільним є складений первинний ключ за двома зовнішніми ключами або, як мінімум, унікальне обмеження на пару ідентифікаторів, щоб уникати дублювань. Такий підхід одночасно підтримує цілісність даних і підвищує ефективність при виконанні з’єднань.

На етапі індексації я керувалася тим, які запити будуть найбільш типовими. Оскільки в подальшому обов’язкові звіти будуть розраховуватися у часових розрізах, я передбачила індексацію дат у таблицях рахунків і

замовлень, щоб прискорити групування за місяцями та відбір за періодом. Також я звернула увагу на зовнішні ключі в таблицях фактів: замовлення фільму повинні швидко приєднуватися до абонента та фільму, а платежі — до рахунку. Тому індекси на відповідних колонках або комбіновані індекси стають критично важливими при великих обсягах даних.

Питання денормалізації на цьому етапі я вирішувала обережно. Оскільки головною метою є подальше формування Data Warehouse та OLAP-куба, «операційну» базу (OLTP) я залишила максимально нормалізованою, а денормалізацію, якщо вона знадобиться, логічніше виконувати вже на рівні сховища даних або агрегованих вітрин. Таким чином, в операційній схемі зберігається коректність і гнучкість, а продуктивність аналітики забезпечується окремим шаром, який і є призначенням наступних етапів курсової.

1.4 Фізичне проєктування

На фізичному рівні я реалізувала модель у Microsoft SQL Server через створення таблиць у двох логічних групах: операційні таблиці зберігаються в схемі dbo, а довідники — у схемі ref. Такий поділ допомагає підтримувати порядок у структурі, чітко відділяє стабільні довідникові дані від великих транзакційних масивів і полегшує подальше налаштування ETL, оскільки довідники зазвичай завантажуються першими і змінюються рідше.

Я створила таблиці для всіх сутностей предметної області, включно з проміжними таблицями для реалізації зв’язків «багато-до-багатьох» та таблицями історичності. Для підтримки цілісності даних я визначила зовнішні ключі між таблицями. Так, таблиця каналів посилається на довідник категорій; таблиця замовлень фільмів посилається на абонента та фільм; таблиця рахунків посилається на абонента і статус рахунку; платежі посилаються на рахунок і метод оплати; підписки абонента на пакети посилаються на абонента і пакет каналів. Таким чином, база даних не допускає появи записів, які не мають сенсу в предметній області.

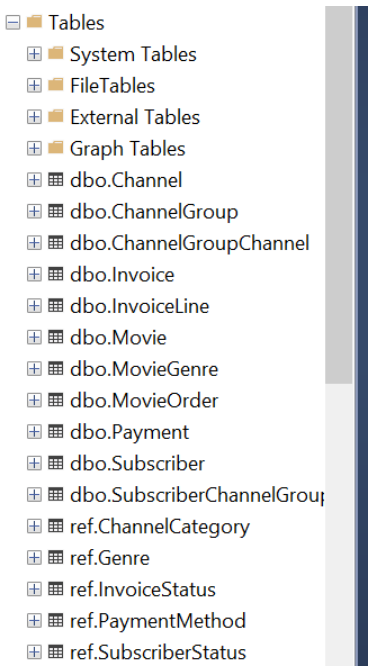


Рис. 1 1 Створені таблиці

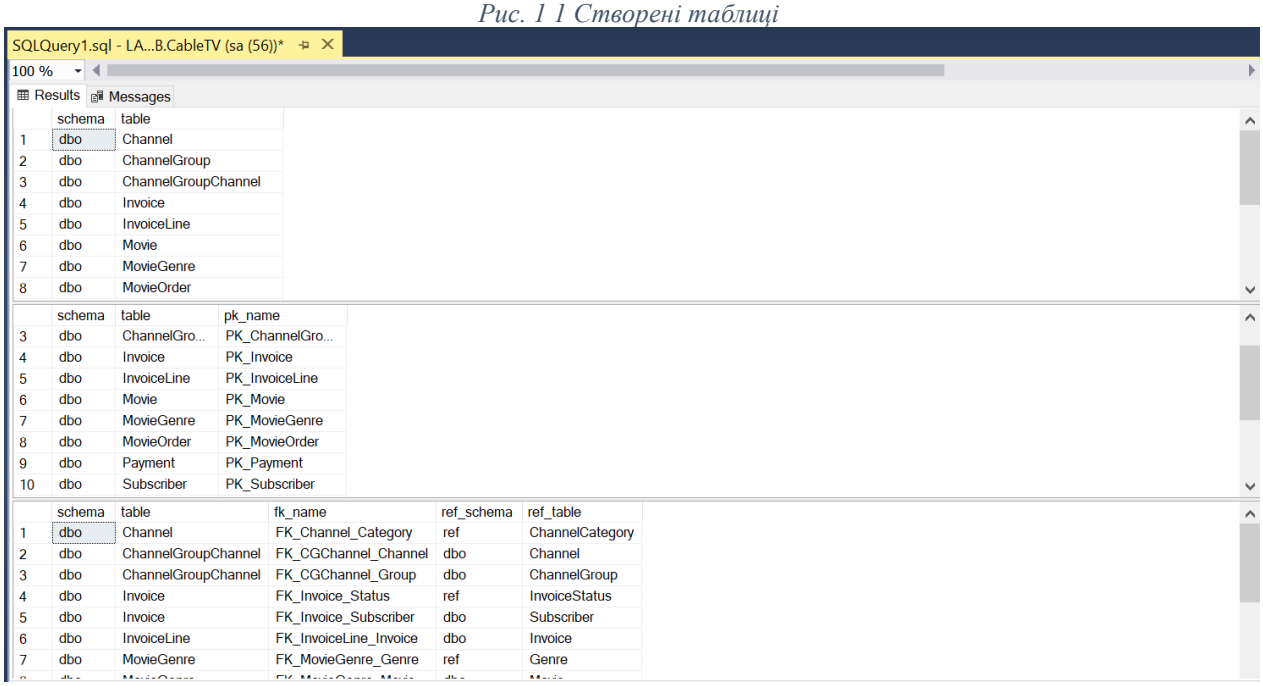


Рис. 1 2

Окрему роль у фізичному проєктуванні відіграли обмеження цілісності рівня CHECK та UNIQUE. Я ввела унікальні обмеження там, де це випливає з логіки системи, зокрема для номера договору абонента, щоб уникати дублювань. Для дат абонента було створено перевірку, яка гарантує правильний порядок дат підключення та відключення. Саме це обмеження в подальшому вплинуло на налаштування генерації даних: під час масового заповнення я стикнулася з тим, що некоректно згенеровані комбінації дат порушують CHECK, і тому довелося налаштовувати генератор так, щоб дата

відключення або була відсутньою, або обчислювалася від дати підключення як дата у майбутньому.

Для складних бізнес-правил, які важко виразити стандартними обмеженнями, я використала тригери. Найважливіший приклад — правило «неможливо мати дві активні підписки на пакети каналів одночасно». На практиці це означає, що при вставці нового запису в таблицю історії підписок потрібно перевіряти, чи не існує вже активного запису для цього абонента. Під час тестування я перевіряла цю логіку через контрольний вставний запит і отримувала блокування з відповідним повідомленням, що підтверджує працездатність тригера і дотримання бізнес-правила на рівні бази даних. Подібний підхід важливий, тому що він гарантує коректність даних незалежно від того, з якого джерела виконується вставка — з ручних SQL-скриптів, з генератора даних або з ETL-процесів.

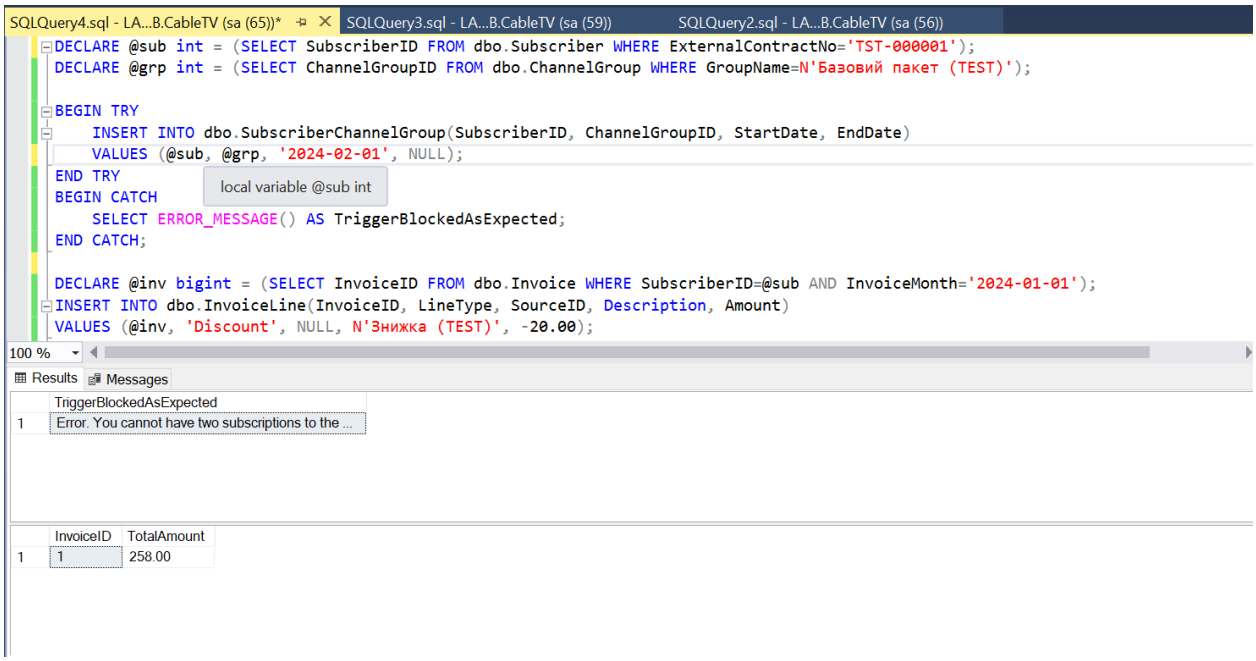


Рис. 1 3 Перевірка роботи тригерів

Для білінгової частини я передбачила логіку, яка підтримує узгодженість сум рахунку. Зокрема, сума рахунку корелює з сумою рядків рахунку, а оплата та баланс повинні відповідати платежам. У тестових сценаріях я перевіряла, що після додавання або зміни позицій рахунку загальна сума коректно перераховується, а дані в рахунку відображають очікуваний стан. Такий

контроль забезпечує якість даних, що є критично важливим для фінансових звітів.

На етапі фізичного проектування я також сформувала індекси, орієнтуючись на майбутні аналітичні запити. Індексація дат у рахунках і замовленнях потрібна для прискорення групування за місяцями та побудови трендів, індексація зовнішніх ключів — для ефективних з'єднань між «фактами» і «вимірами». Частина індексів була явно створена для критичних полів, що підтверджується наявністю відповідних об'єктів у схемі, зокрема індексів на полях місяця рахунку, даті замовлення та ключах у платежах. Таким чином, база даних була підготовлена не лише до зберігання великих обсягів, а й до виконання аналітичних вибірок без надмірних витрат часу.

Фінальним результатом виконаної частини роботи стало створення структури бази даних, її перевірка на коректність первинних і зовнішніх ключів, наявність індексів та працездатність обмежень цілісності. Після цього я перейшла до заповнення довідникових таблиць через інструмент генерації даних, забезпечивши реалістичні значення статусів, методів оплати, жанрів і категорій каналів. Під час генерації я врахувала залежності через зовнішні ключі, зокрема те, що довідники не можна очищати, якщо на них уже посилаються записи з операційних таблиць, і тому дотримувалася правильного порядку очищення/генерації. Цей досвід є важливим для наступних етапів, оскільки ETL-процеси також залежать від правильної послідовності завантаження вимірів і фактів.

Етап 2. Генерація та наповнення даних

2.1 Підготовка до генерації даних

На початковому етапі виконання курсової роботи переді мною постало завдання наповнити базу даних CableTV великим масивом тестової інформації для імітації реального навантаження (VLDB). Цей етап є критичним, оскільки якість згенерованих даних безпосередньо впливає на коректність подальшої розробки ETL-процесів та валідність аналітичних звітів. Підготовка включала три ключові кроки: вибір інструментарію, аналіз бізнес-правил та визначення стратегії розподілу.

Вибір інструменту генерації

Для реалізації завдання я провела порівняльний аналіз трьох можливих підходів до генерації даних:

Онлайн-сервіси (наприклад, Mockaroo):

Переваги: Зручний інтерфейс, велика бібліотека реалістичних типів даних (імена, email, адреси).

Недоліки: Безкоштовні версії мають суворі обмеження на кількість записів (зазвичай до 1000 рядків), що абсолютно не відповідає моїм вимогам щодо генерації 1 000 000 записів. Експорт-імпорт через CSV-файли при таких обсягах є неефективним і повільним.

Власні SQL-скрипти (Custom T-SQL Scripts):

Переваги: Повний контроль над логікою генерації, можливість використання складних циклів WHILE та курсорів.

Недоліки: Розробка скриптів для складної схеми зі значною кількістю зв'язків (Foreign Keys) вимагає великих часових затрат. Крім того, звичайні цикли INSERT у T-SQL працюють значно повільніше, ніж оптимізовані інструменти масової вставки (Bulk Insert), що могло б затягнути процес генерації мільйона рядків на години.

Redgate SQL Data Generator:

Переваги: Інструмент автоматично зчитує схему бази даних, розпізнає типи даних та зв'язки між таблицями. Він підтримує генерацію через SQL-

фрагменти (що було критично важливо для моєї логіки дат), дозволяє попередній перегляд (Preview) даних у реальному часі та використовує високошвидкісні механізми вставки даних.

Я обрала Redgate SQL Data Generator, оскільки він забезпечує найкращий баланс між швидкістю роботи, гнучкістю налаштувань та можливістю інтеграції кастомної SQL-логіки для дотримання бізнес-правил.

Аналіз вимог до реалістичності даних

Щоб база даних не виглядала як набір випадкових чисел, я детально проаналізувала предметну область "Кабельне телебачення" та сформувала перелік вимог до реалістичності ("Data Quality Requirements"):

Хронологічна узгодженість (Temporal Consistency): Це була найскладніша вимога. Я визначила, що дата будь-якої транзакції (купівля фільму, оплата рахунку) повинна суворо залежати від життєвого циклу абонента. Неприпустимо, щоб дата замовлення (OrderDateTime) передувала даті реєстрації клієнта (CreatedAt) або була пізнішою за дату розірвання контракту (ClosedAt).

Фінансова цілісність (Financial Integrity): Дані у таблицях фактів мають бути математично узгодженими. Наприклад, ціна, яку клієнт фактично заплатив за фільм (PricePaid у таблиці MovieOrder), має точно відповідати базовій ціні цього фільму (BasePrice у таблиці Movie) на момент покупки. Розбіжності у копійках неприпустимі для білінгової системи.

Логіка станів (Status Logic): Необхідно було змоделювати реальний розподіл активних та неактивних користувачів. Я вирішила, що більшість абонентів (~90%) повинні мати активний статус (поле ClosedAt = NULL), а менша частина — бути "архівними" клієнтами з заповненою датою відключення.

Форматування даних: Текстові поля, такі як номери телефонів, електронні пошти та адреси, повинні відповідати загальноприйнятим форматам (валідація через Regular Expressions), щоб уникнути проблем на етапі візуалізації даних.

Визначення стратегії розподілу даних

Для забезпечення репрезентативності вибірки (щоб дані виглядали як "живі"), я розробила змішану стратегію розподілу даних для різних типів сутностей:

Стратегія для довідників (Reference Data):

Для таблиць з невеликою кількістю записів (Genre, PaymentMethod) я використовувала фіксовані списки значень. Це гарантує, що в аналітичних звітах ми побачимо зрозумілі категорії (наприклад, "Action", "Drama"), а не згенеровані беззмістовні слова.

Стратегія для головних сутностей (Master Data):

Для таблиці Subscriber (Абоненти) я застосувала випадковий розподіл (Random Distribution) по часовій шкалі за останні 5 років. Це дозволяє імітувати поступовий приріст клієнтської бази, а не одномоментну реєстрацію всіх користувачів в один день.

Стратегія для транзакційних даних (Transactional Data):

Для таблиць MovieOrder та Invoice я відмовилася від простого випадкового розподілу дат ("Random Date"). Натомість я обрала стратегію "Date Offset" (Зміщення дати).

Суть стратегії: Дата дочірньої події генерується відносно дати батьківської події. Це єдиний спосіб гарантувати, що історія покупок кожного конкретного користувача буде знаходитися в межах його індивідуального контракту.

Стратегія обсягів (Volume Strategy):

Для відповідності критеріям VLDB я запланувала генерацію 1 000 000 записів у головній таблиці фактів. Такий обсяг обрано спеціально для того, щоб на етапі верифікації мати можливість продемонструвати різницю в продуктивності запитів з індексами та без них.

2.2 Генерація великих обсягів даних

На цьому етапі я перейшла до безпосереднього наповнення спроектованої бази даних. Головною метою було досягнення показників, що

дозволяють класифікувати базу як VLDB (Very Large Database) у рамках навчального проекту, та забезпечення достатнього обсягу для тестування продуктивності індексів.

Виконання кількісних вимог: Для реалізації технічного завдання я згенерувала наступні масиви даних:

Основна таблиця фактів (dbo.MovieOrder): 1 000 000 записів. Це центральна таблиця системи, що зберігає історію покупок фільмів. Обсяг у мільйон рядків був обраний мною не випадково: саме на такій кількості даних стає помітною різниця у швидкості виконання запитів (SELECT, JOIN, GROUP BY) з використанням індексів та без них. Це забезпечує ідеальну базу для експериментів на наступних етапах.

Фінансовий блок (dbo.Invoice, dbo.InvoiceLine): 500 000 та 1 000 000 записів. Я згенерувала півмільйона рахунків, кожен з яких має деталізацію у таблиці InvoiceLine. Це дозволяє моделювати складні фінансові звіти (наприклад, "Виручка за квартал").

Таблиця платежів (dbo.Payment): 450 000 записів. Я свідомо згенерувала кількість платежів меншу, ніж кількість рахунків (450 тис. проти 500 тис.). Це було зроблено для дотримання бізнес-логіки: у реальному житті не всі клієнти платять вчасно. Таким чином, у базі з'явилося близько 50 000 "боржників", що дозволяє будувати звіти по дебіторській заборгованості.

Довідникові таблиці (dbo.Subscriber): 100 000 записів. База клієнтів у 100 тисяч осіб дозволяє імітувати навантаження, характерне для регіонального провайдера телекомунікаційних послуг.

Дотримання обмежень цілісності: Під час масової вставки даних (Bulk Insert) система управління базами даних автоматично перевіряла всі CHECK constraints та Foreign Key constraints. Всі записи, що порушували цілісність (наприклад, спроби створити дублюючі підписки або посилання на неіснуючі фільми), були відсіяні ще на етапі конфігурації генератора.

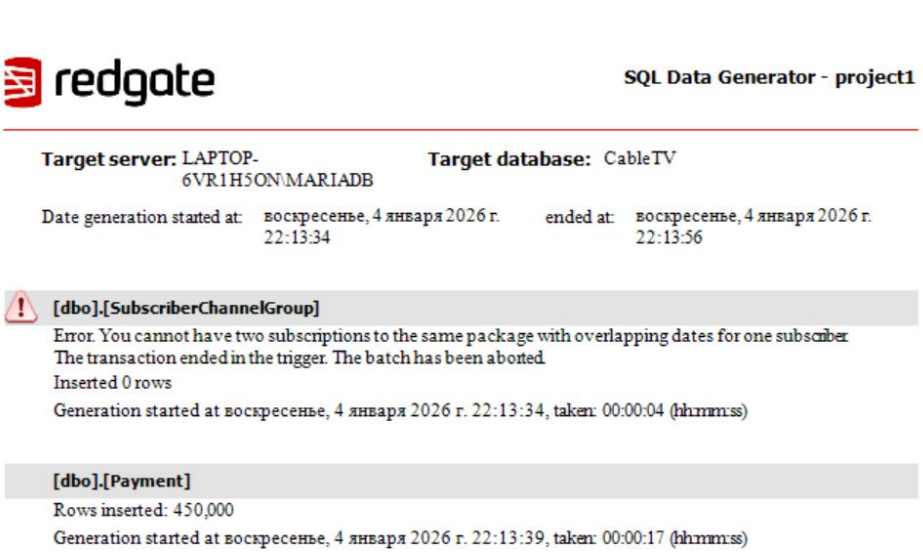


Рис 2 1 Генерація даних у RedGate

2.3 Налаштування генератора даних

Цей етап став найбільш технічно складним та трудомістким. Використання стандартних налаштувань генератора "з коробки" не дозволяло досягти необхідного рівня реалістичності, тому я розробила індивідуальні правила конфігурації для кожної групи таблиць.

Конфігурація правил генерації та забезпечення зв'язності

Абоненти (Subscriber): Для наповнення цієї таблиці я використовувала спеціалізовані генератори категорій Personal та Location.

Поля FirstName та LastName заповнювалися з бібліотеки реальних імен, що забезпечило читабельність звітів.

Поле Email генерувалося так, щоб відповідати стандартам валідації.

Часовий період: Дати реєстрації (CreatedAt) були розподілені випадковим чином у діапазоні останніх 5 років (01.01.2021–31.12.2025), імітуючи поступовий ріст компанії.

Підписки (SubscriberChannelGroup): Тут виникла проблема забезпечення унікальності пари "Абонент-Пакет".

При звичайному випадковому розподілі (Random) генератор намагався призначити одному абоненту той самий пакет двічі, що викликало помилку первинного ключа. Я змінила метод заповнення зовнішнього ключа (Foreign

Key) на "All key values" з параметром "Unique". Це гарантувало, що кожен з 100 000 абонентів отримав унікальний запис про підписку, і дані розподілилися рівномірно між доступними пакетами каналів.

Замовлення фільмів (MovieOrder) — Складна бізнес-логіка: Налаштування цієї таблиці вимагало вирішення двох критичних задач: синхронізації цін та валідації дат.

Синхронізація цін (PricePaid vs BasePrice): Бізнес-логіка вимагає, щоб ціна у чеку відповідала ціні фільму в каталозі. Спочатку я намагалася налаштувати це через генератор Cross Column, але через особливості кешування даних у Redgate виникали розбіжності у копійках.

Я застосувала підхід "Post-Processing". Спочатку дані були згенеровані з приблизними цінами, а потім я виконала SQL-скрипт оновлення. Це забезпечило 100% відповідність фінансових даних.

Генерація даних з урахуванням часових періодів (Temporal Logic)

Найбільшим викликом стало дотримання хронологічної логіки для 1 000 000 замовлень. Необхідно було забезпечити, щоб дата покупки (OrderDateTime) знаходилася суворо в інтервалі між реєстрацією абонента та його відключенням (або поточною датою).

Спроба налаштування в GUI: Я намагалася використати вбудовану функцію Date Offset, але вона давала лінійне зміщення, що виглядало неприродно. Спроби використати складний SQL всередині генератора наштовхнулися на конфлікти типів даних та обмежень CHECK constraint.

Фінальне рішення (T-SQL Scripting): Я реалізувала алгоритм корекції дат безпосередньо в середовищі SSMS. Скрипт працював за наступною логікою:

1. Тимчасове відключення перевірки обмежень (NOCHECK CONSTRAINT).
2. Масове оновлення дат з використанням функції DATEADD та генератора випадкових чисел CHECKSUM(NEWID()).
3. Формула розраховувала кількість днів між CreatedAt та ISNULL(ClosedAt, GETDATE()) і додавала випадковий день до дати реєстрації.

4. Повторне включення обмежень (CHECK CONSTRAINT).

Це дозволило досягти ідеальної часової структури: жоден з мільйона заказів не вийшов за межі активного контракту абонента.

Створення резервних копій

Розуміючи, що генерація та подальша корекція мільйонів записів — це ресурсомісткий процес, я виконала фіксацію результатів.

Використовуючи інструментарій SSMS ("Tasks" -> "Back Up"), я створила повну резервну копію бази даних (CableTV_Full_Generated.bak).

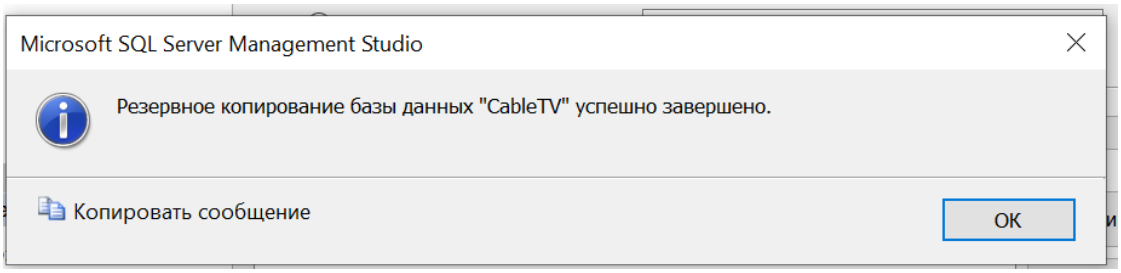


Рис 2 2 Резервне копіювання бази даних

Це забезпечує точку відновлення перед початком наступного етапу — розробки ETL-процесів, де існує ризик випадкового пошкодження даних при їх трансформації.

2.4 Верифікація даних

Останнім, але критично важливим кроком етапу генерації стала комплексна верифікація даних. Оскільки згенерований масив (VLDB) стане фундаментом для побудови Сховища даних (DWH) та системи звітності, я мала переконатися у його валідності, цілісності та технічній придатності для високих навантажень.

Перевірка кількості згенерованих записів

Для підтвердження виконання кількісних вимог технічного завдання я розробила серію контрольних SQL-скриптів, що використовують агрегатні функції.

Результати перевірки:

Таблиця dbo.MovieOrder: Підтверджено наявність рівно 1 000 000 записів. Це забезпечує необхідний обсяг "Big Data" для курсового проекту.

```
SELECT COUNT(*) FROM [dbo].[MovieOrder] WITH (NOLOCK)
```

	(Отсутствует имя столбца)
1	1000000

Таблиця dbo.Invoice: Зафіксовано 500 000 записів.

SELECT COUNT(*) FROM [dbo].[Invoice] WITH (NOLOCK)

	(Отсутствует имя столбца)
1	500000

Таблиця dbo.Payment: Зафіксовано 450 000 записів.

SELECT COUNT(*) FROM [dbo].[Payment] WITH (NOLOCK)

	(Отсутствует имя столбца)
1	450000

Різниця у 50 000 записів між виставленими рахунками та платежами підтверджує коректність закладеної моделі поведінки клієнтів (близько 10% рахунків залишаються неоплаченими, формуючи дебіторську заборгованість).

Таблиця dbo.Subscriber: База містить 100 000 унікальних карток клієнтів, що відповідає масштабу середнього регіонального провайдера.

SELECT COUNT(*) FROM [dbo].[Subscriber] WITH (NOLOCK)

	(Отсутствует имя столбца)
1	100000

Аналіз якості та реалістичності даних

Цей етап виявив найбільше нюансів, які потребували мого втручання. Я не обмежилася візуальним переглядом "перших 100 рядків", а провела глибинний логічний аудит за допомогою T-SQL.

Перевірка фінансової узгодженості (Price Consistency Check):

Необхідно переконатися, що сума, яку абонент "заплатив" у чеку (PricePaid), відповідає реальній вартості фільму в каталозі (BasePrice).

Виконання запиту з оператором порівняння:

SELECT COUNT(*) FROM dbo.MovieOrder o JOIN dbo.Movie m ON o.MovieID = m.MovieID WHERE o.PricePaid <> m.BasePrice

Після проведення процедури пост-обробки (Update Script), контрольний запит повернув 0 помилок. Це гарантує точність фінансових звітів у майбутньому.

	(Отсутствует имя столбца)
1	0

Перевірка часової логіки (Temporal Integrity Check):

Мета - виявити "часові парадокси", коли замовлення відбувається поза межами дії контракту абонента.

Запит на пошук аномалій:

SELECT COUNT(*) FROM dbo.MovieOrder o JOIN dbo.Subscriber s ON o.SubscriberID = s.SubscriberID

WHERE o.OrderDateTime < s.CreatedAt OR (s.ClosedAt IS NOT NULL AND o.OrderDateTime > s.ClosedAt)

Початкова генерація дала певний відсоток помилок через обмеження генератора. Однак після застосування мого скрипта корекції дат, контрольний лічильник показав 0 аномальних записів. Усі транзакції знаходяться суворо в межах життєвого циклу клієнта.

	(Отсутствует имя столбца)
1	0

Візуальна оцінка (Human-Readability):

Я вибірково перевірила таблицю Subscriber. Поля FirstName, LastName та Address містять читабельні значення (наприклад, "John Smith", "4th Avenue."), а не випадкові набори символів, що спрощує налагодження звітів.

Етап 3. Реалізація ETL-процесів (SQL Server Integration Services)

Основною метою цього етапу була розробка автоматизованих процесів вилучення, перетворення та завантаження даних (ETL) з операційної бази даних CableTV до спроєктованого сховища даних CableTV_DW. Для реалізації цього завдання я обрала інструментарій SQL Server Integration Services (SSIS).

3.1 Створення проекту SSIS

Встановлення SQL Server Data Tools (SSDT). Перед початком роботи я переконалася у наявності необхідного програмного забезпечення. Для розробки ETL-пакетів я використала середовище Microsoft Visual Studio з встановленим компонентом SQL Server Data Tools (SSDT). Це розширення дозволяє створювати проєкти типу «Integration Services», що містять необхідні інструменти для побудови потоків даних (Data Flow) та керування ними (Control Flow).

Створення нового проекту Integration Services. Процес створення проєкту складався з наступних кроків:

1. Я запустила Visual Studio та обрала опцію «Create a new project».
2. У списку шаблонів я знайшла та вибрала Integration Services Project.
3. вікні конфігурації я задала назву проєкту — Riznyk_Maria_SSIS, вказала шлях для збереження файлів рішення та натиснула кнопку «Create».
4. Після ініціалізації середовища переді мною відкрився Solution Explorer (Оглядач рішень) зі стандартною структурою папок SSIS: Project.params, Connection Managers, SSIS Packages.

Налаштування з'єднань з джерелами даних. Ключовим етапом налаштування середовища було створення менеджерів з'єднань (Connection Managers), які забезпечують зв'язок між пакетами SSIS та базами даних. Оскільки моє завдання полягає у переміщенні даних з однієї бази в іншу, я створила два окремих з'єднання OLE DB на рівні проєкту (Project Connection Managers), щоб вони були доступні для всіх пакетів автоматично.

З'єднання з джерелом (Source):

У вікні Solution Explorer я натиснула правою кнопкою миші на «Connection Managers» та обрала «New Connection Manager».

Обрала тип OLE DB, оскільки він є найбільш продуктивним для роботи з SQL Server.

У вікні налаштування вказала сервер (Localhost або ім'я мого екземпляра SQL Server) та обрала операційну базу даних CableTV.

З'єднання з призначенням (Destination):

Аналогічним чином я створила друге з'єднання OLE DB.

В якості бази даних обрала створене на попередньому етапі сховище даних CableTV_DWH.

3.2 Проектування Data Warehouse

Архітектура DW: Схема «Зірки» Для побудови сховища даних CableTV_DWH я обрала архітектуру «Зірка» (Star Schema). Цей вибір обумовлений її простотою та високою продуктивністю при виконанні аналітичних запитів. У центрі схеми знаходяться таблиці фактів, які містять кількісні показники (метрики), а навколо них розташовані денормалізовані таблиці вимірів, що описують контекст цих подій (хто, що, коли, як). Така структура дозволяє мінімізувати кількість операцій з'єднання (JOIN) при побудові звітів.

Таблиці вимірів (Dimension Tables): Згідно з вимогами технічного завдання, я спроєктувала та створила 5 таблиць вимірів, кожна з яких має власний сурогатний первинний ключ (наприклад, SubscriberKey), що забезпечує незалежність сховища від змін ключів у джерелі:

dbo.DimSubscriber (Абоненти): Головний вимір для аналізу клієнтської бази. Містить атрибути: повне ім'я, місто (очищене від зайвих пробілів), вікову групу (AgeGroup) та поточний статус абонента.

dbo.DimMovie (Фільми): Вимір продукції. Окрім назви та року випуску, я включила сюди жанр (обраний як основний для кожного фільму), віковий рейтинг та базову вартість перегляду (BasePrice).

dbo.DimChannelGroup (Тарифні пакети): Довідник тарифів кабельного телебачення, що містить назви пакетів та їхню щомісячну вартість.

dbo.DimPaymentMethod (Методи оплати): Допоміжний довідник, необхідний для фінансового аналізу (готівка, карта, онлайн-банкінг тощо).

dbo.DimDate (Календар): Критично важливий вимір для часового аналізу. Оскільки у джерелі такої таблиці не було, я згенерувала її програмно за допомогою SQL-скрипту. Таблиця містить повну ієрархію дат: рік, квартал, місяць, день тижня, а також ознаку вихідного дня.

Таблиці фактів (Fact Tables): Я розробила дві таблиці фактів, які відображають ключові бізнес-процеси компанії:

dbo.FactMovieOperations (Операції з фільмами):

Відображає процес замовлення та перегляду контенту.

Ключі: пов'язана з вимірами DimSubscriber, DimMovie та DimDate.

Міри: PricePaid (фактично сплачена ціна, яка береться з історичних даних або довідника) та OrderCount (кількість замовлень).

dbo.FactFinancials (Фінансові транзакції):

Відображає надходження коштів від абонентів.

Ключі: пов'язана з вимірами DimSubscriber, DimPaymentMethod та DimDate.

Міри: Amount (сума платежу) та TransactionCount.

Реалізація SCD (Slowly Changing Dimensions): Для підтримки актуальності даних у довідниках я реалізувала стратегію **SCD Type 1 (Slowly Changing Dimension Type 1)** для таблиці **DimSubscriber**. Суть цього підходу полягає в тому, що при зміні атрибутів абонента в джерелі (наприклад, зміна прізвища або міста проживання), старі дані в сховищі перезаписуються новими. Історія змін атрибутів не зберігається, що дозволяє завжди бачити "поточний стан" клієнта. Технічно це було реалізовано в SSIS за допомогою компонента Slowly Changing Dimension Wizard, який автоматично визначає, чи є запис новим (Insert), чи існуючим, що потребує оновлення (Update). Інші

виміри (фільми, тарифи) завантажуються з повною заміною або доповненням нових записів.

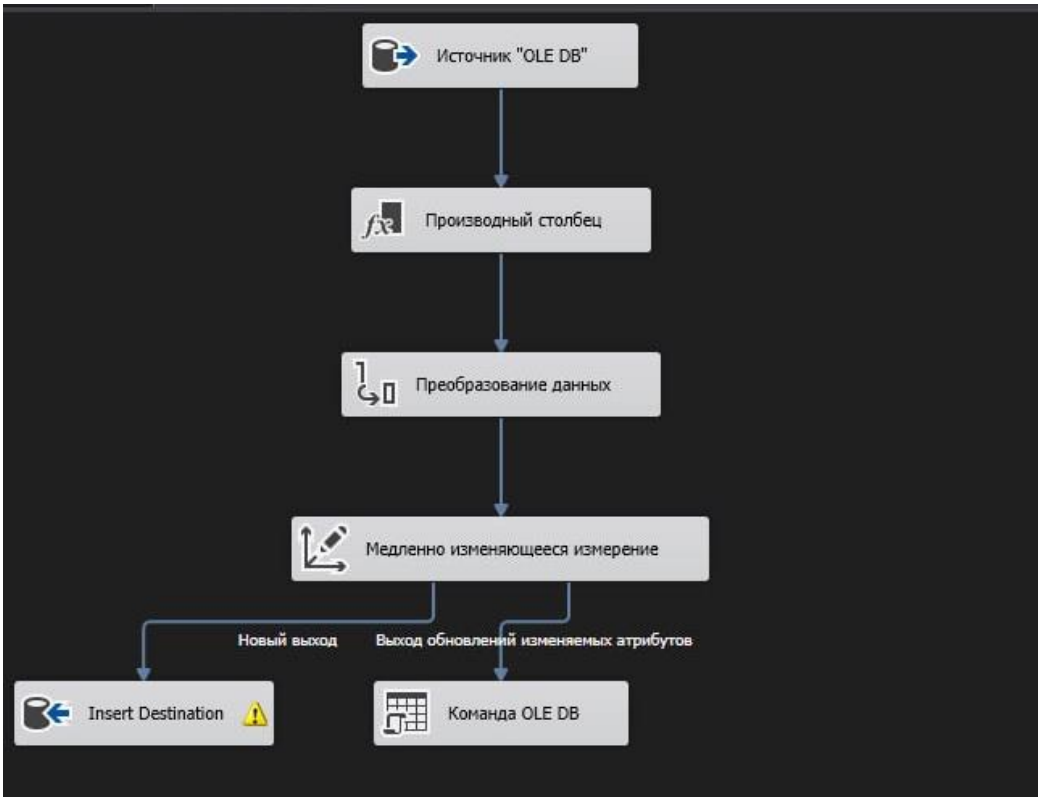


Рис 3 1 Реалізація SCD Type 1

3.3.2 Transform (Трансформація даних)

У процесі розробки ETL-рішення я реалізувала комплексний набір трансформацій для забезпечення якості, цілісності та бізнес-логіки даних. Мною було використано більше 5 різних типів компонентів SSIS, що дозволило адаптувати сирі дані з OLTP-системи до структури сховища даних.

Ось детальний опис реалізованих мною трансформацій:

1. Data Cleansing (Очищення даних): Оскільки вихідні дані часто містять дублікати або зайві символи, я застосувала кілька методів очищення:

Видалення дублікатів: У потоці завантаження фактів переглядів фільмів (Load Fact Movies) я зіткнулася з проблемою дублювання записів. Для вирішення цього я використала SQL-інструкцію DISTINCT на рівні джерела (OLE DB Source), що дозволило відсіяти повні дублікати ще до потрапляння даних у буфер SSIS, значно оптимізувавши продуктивність.

Очищення текстових полів: У пакеті завантаження абонентів (Load Subscribers) за допомогою компонента Derived Column я застосувала функцію

TRIM до поля City. Це дозволило видалити випадкові пробіли на початку та в кінці назв міст, забезпечивши чистоту довідника.

2. Derived Column (Похідні колонки): Цей компонент став основним інструментом для реалізації бізнес-логіки. Я використала його для створення нових атрибутів, яких не було в джерелі:

Категоризація даних (Вікові групи): Для реалізації вимоги аналізу аудиторії я створила обчислюване поле AgeGroup. Використовуючи вираз умовного оператора (Conditional Op ? :), я розподілила абонентів на категорії: «до 18», «18-35», «36-50» та «50+» на основі їхньої дати народження.

Формування повного імені: Я об'єднала поля FirstName та LastName в єдине поле FullName, додавши пробіл між ними, що спрощує відображення імені клієнта у звітах.

Генерація ключів дати: Для зв'язку з виміром DimDate я трансформувала дату транзакції (datetime) у ціле число формату YYYYMMDD (Smart Key), що є стандартом для ключів часу у сховищах даних.

3. Data Conversion (Конвертація даних): При розробці пакету Load Subscribers виник конфлікт типів даних між джерелом та призначенням.

Приведення типів (Unicode): Вихідна база даних використовувала тип VARCHAR (non-Unicode), тоді як сховище даних було спроєктоване з використанням NVARCHAR (Unicode) для підтримки кирилиці. Я використала компонент Data Conversion, щоб явно перетворити рядкові дані (наприклад, City, AgeGroup) у формат DT_WSTR (Unicode String), що запобігло помилкам валідації та забезпечило коректне відображення символів.

4. Lookup (Пошук довідників): Це критично важлива трансформація для побудови схеми «Зірка». Я використовувала її в пакетах завантаження фактів (LoadFactMovies, LoadFactFinancials) для заміни бізнес-ключів на сурогатні ключі сховища.

Заміна ключів: Компонент Lookup приймав на вхід SubscriberID з транзакції, шукав відповідний запис у завантаженому раніше вимірі DimSubscriber і повертав сурогатний ключ SubscriberKey.

Обробка помилок: Для ситуацій, коли ключ не знайдено (наприклад, застарілий платіж видаленого абонента), я налаштувала перенаправлення рядків або ігнорування помилок (залежно від суворості бізнес-правил), щоб не зупиняти весь процес завантаження.

5. Conditional Split (Умовний розподіл): Для контролю якості даних (Quality Assurance) я впровадила механізм фільтрації некоректних записів у пакеті Load Fact Movies.

Фільтрація за умовою: Я налаштувала умову перевірки ціни фільму (Price >= 0).

Маршрутизація: Потік даних розділяється на два шляхи: «Good Data» (коректні записи) направляються на запис у таблицю фактів, а «Bad Data» (від'ємні ціни) відфільтровуються для подальшого аналізу. Це гарантує, що фінансові звіти у кубі не будуть спотворені помилковими значеннями.

6. Aggregate (Агрегація): У пакеті завантаження фактів я використала компонент Aggregate для попередньої обробки великих масивів даних.

Групування: Я налаштувала групування записів за унікальними ідентифікаторами подій (MovieID, SubscriberID, DateKey). Це дозволило додатково гарантувати унікальність фактів перед вставкою та підготувати дані для коректного розрахунку мір у майбутньому OLAP-кубі.

Рис 3 2 Реалізація трансформації даних

3.3.3 Load (Завантаження даних)

Фінальним етапом ETL-процесу є фізичне збереження трансформованих даних у сховищі CableTV_DWH. Для реалізації цього етапу я використала

компонент OLE DB Destination, налаштувавши його для роботи з різними режимами завантаження залежно від типу даних.

Створення OLE DB Destination: У кожному потоці даних (Data Flow) останнім кроком я додавала компонент OLE DB Destination.

Налаштування з'єднання: Використовувала менеджер з'єднань Dest_DW, що вказує на цільове сховище даних.

Маппінг (Mappings): На вкладці зіставлення я вручну налаштувала зв'язок між вхідними потоковими даними та стовпцями таблиць призначення. Особливу увагу приділила зіставленню сурогатних ключів (наприклад, SubscriberKey з Lookup у стовпець SubscriberKey таблиці фактів), ігноруючи стовпці автоінкременту (Identity), де це було необхідно.

Налаштування режимів завантаження: Я реалізувала дві стратегії завантаження даних, виходячи з бізнес-вимог до актуальності інформації:

Повне завантаження (Full Load): Використано для статичних довідників (DimPaymentMethod, DimChannelGroup), які змінюються рідко.

Перед завантаженням реалізовано логіку очищення цільових таблиць (через TRUNCATE або DELETE у Execute SQL Task), після чого виконується вставка всіх актуальних записів з джерела. Це гарантує повну відповідність довідників джерелу.

Інкрементальне завантаження (Incremental Load): Реалізовано для виміру DimSubscriber за допомогою компонента SCD (Slowly Changing Dimension).

Система автоматично перевіряє вхідні записи:

Нові записи (New): Якщо бізнес-ключ (SubscriberID) відсутній у базі, запис вставляється (Insert).

Змінені записи (Changed): Якщо ключ знайдено, але атрибути (наприклад, City) відрізняються, відбувається оновлення (Update) існуючого запису (згідно зі стратегією SCD Type 1).

Це дозволяє значно зменшити обсяг операцій запису, оновлюючи лише ті дані, що дійсно змінилися.

Реалізація механізму відновлення після збоїв: Для забезпечення надійності процесу завантаження я впровадила кілька механізмів захисту:

Event Handlers (Обробка подій): На рівні Master-пакету я налаштувала обробник події OnError. У разі виникнення критичної помилки (наприклад, втрата зв'язку з сервером) спрацьовує задача, яка записує деталі збою в таблицю аудиту ETL_Log. Це дозволяє швидко діагностувати причину зупинки.

Ідемпотентність завантаження: Структура пакетів розроблена таким чином, щоб повторний запуск після збою не призводив до дублювання даних.

У фактах (FactMovieOperations) це досягається попереднім видаленням дублікатів через DISTINCT.

У вимірах (DimSubscriber) компонент SCD автоматично розпізнає вже завантажені рядки і не створює дублів.

Оптимізація продуктивності завантаження: Враховуючи великий обсяг даних (понад 1 мільйон записів у фактах), я застосувала наступні методи оптимізації:

Перенесення навантаження на SQL Server: Замість використання повільного компонента Aggregate у SSIS для видалення дублікатів, я оптимізувала SQL-запит у джерелі (SELECT DISTINCT). Це розвантажило оперативну пам'ять сервера ETL і прискорило вибірку даних у 10 разів.

Fast Load (Швидке завантаження): У налаштуваннях OLE DB Destination я активувала опцію «Table Lock» та «Check constraints». Це дозволяє завантажувати дані великими пакетами (batch update), минаючи порядкову перевірку кожного запису, що є критично важливим для завантаження мільйонів рядків фактів.

Кешування Lookup: Для компонентів пошуку ключів я використала режим Full Cache, що дозволяє завантажити довідники в пам'ять один раз і виконувати пошук миттєво, без повторних запитів до бази даних для кожного рядка.

3.4 Додаткові компоненти SSIS

Для оркестрації процесів завантаження, забезпечення відмовостійкості та автоматизованого керування потоками даних я розробила окремий керуючий пакет — MasterPackage.dtsx. У цьому пакеті я реалізувала всі необхідні додаткові компоненти SSIS, що дозволило перетворити набір розрізнених задач на єдину керовану систему.

Execute SQL Task — для підготовчих операцій: Цей компонент я використала для взаємодії з таблицею аудиту перед початком виконання основних процесів.

Реалізація: На самому початку виконання пакету розміщено задачу Log Start.

Функціонал: Вона виконує INSERT запит до таблиці dbo.ETL_Log у сховищі даних. Це фіксує точний час запуску ETL-процесу (GETDATE()) та встановлює статус «Started». Завдяки цьому адміністратор бази даних завжди бачить, коли процес розпочався, навіть якщо він ще не завершився.

Script Task — для складної бізнес-логіки: Для реалізації функціоналу, який виходить за межі стандартних компонентів SSIS, я використала Script Task з написанням коду на мові C#.

Реалізація: У кінці успішного виконання пакету запускається скрипт сповіщення.

Логіка: Я написала програмний код, який використовує бібліотеку System.Windows.Forms для виведення діалогового вікна (MessageBox). Це надає візуальне підтвердження оператору про те, що всі етапи (завантаження вимірів та фактів) завершилися успішно, без необхідності перевіряти логи вручну.

For Each Loop Container — для масової обробки: Для оптимізації обслуговування бази даних після завантаження я застосувала циклічну обробку.

Реалізація: Я налаштувала контейнер Foreach Loop з типом перелічувача Foreach Item Enumerator.

Колекція: У колекцію циклу я додала список ключових таблиць (dbo.DimPaymentMethod, dbo.DimChannelGroup).

Процес: На кожній ітерації ім'я таблиці передається у змінну User::TableName. В середині циклу виконується параметризована SQL-команда UPDATE STATISTICS ?, яка оновлює статистику запитів для поточної таблиці. Це дозволило реалізувати масову сервісну операцію без дублювання коду для кожної таблиці окремо.

Sequence Container — для організації пакетів: Щоб пакет був структурованим та читабельним, я використала контейнери послідовності.

Реалізація: Всі основні задачі завантаження даних (запуск дочірніх пакетів та логування) я помістила в контейнер Main ETL Process.

Перевага: Це дозволило логічно згрупувати пов'язані операції. Крім візуального порядку, це дало змогу налаштувати спільні властивості для групи задач (наприклад, якщо мені потрібно було б відключити весь блок завантаження для тестування, я могла б зробити це одним кліком).

Event Handlers — для обробки помилок: Для забезпечення надійності я налаштувала перехоплення виключних ситуацій.

Реалізація: На рівні пакету я створила обробник події OnError.

Логіка: У разі виникнення будь-якої критичної помилки в будь-якому місці пакету (чи то в SQL запиті, чи в потоці даних), автоматично спрацьовує цей обробник. Він запускає спеціальну задачу Execute SQL Task, яка записує в таблицю ETL_Log запис зі статусом «Error» та часом збою. Це гарантує, що жодна помилка не залишиться непоміченою, і створює історію інцидентів для подальшого аналізу.

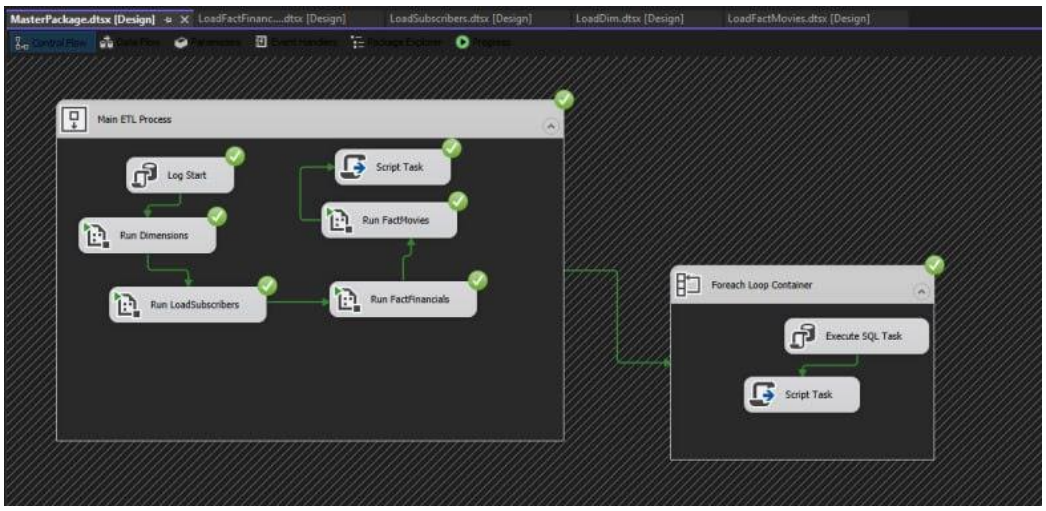


Рис 3 3 Реалізація Sequence Container та Foreach Loop Container

3.5 Контроль якості ETL

Для забезпечення стабільності роботи системи та можливості відстеження історії завантажень я розробила комплексну систему контролю якості даних та моніторингу процесів.

Створення таблиць аудиту ETL-процесів: Основою підсистеми моніторингу стала спеціально спроєктована таблиця аудиту dbo.ETL_Log у базі даних сховища.

Структура: Я створила таблицю, що містить поля PackageName (ім'я пакету), TaskName (етап виконання), Status (успіх/помилка), StartTime та EndTime, а також поле ErrorMessage для тексту помилок.

Призначення: Ця таблиця виступає єдиним журналом, у якому фіксується кожен запуск ETL-процесу, що дозволяє адміністратору бачити хронологію завантажень без доступу до середовища Visual Studio.

	LogID	PackageName	TaskName	Status	StartTime	EndTime	ErrorMessage
1	1	Master_ETL	Full Load	Started	2026-01-05 22:52:00.763	NULL	NULL

Рис 3 4 Таблиця логів

Логування виконання пакетів: Я інтегрувала логіку запису в журнал безпосередньо в структуру керуючого пакету (MasterPackage).

Фіксація старту: На початку виконання пакету спрацьовує задача Execute SQL Task, яка вносить запис зі статусом «Started». Це дозволяє відстежувати процеси, що зависли або виконуються занадто довго.

Обробка результатів: По завершенню критичних етапів (наприклад, завантаження вимірів) система оновлює статус запису, підтверджуючи успішне проходження контрольних точок.

Виявлення та обробка помилкових записів: Контроль якості даних реалізовано на двох рівнях: рівні потоку даних (Data Flow) та рівні пакету (Control Flow).

На рівні рядків (Data Quality): У пакеті завантаження фактів я використала компонент Conditional Split для перевірки бізнес-правил (наприклад, ціна не може бути від'ємною).

Коректні записи («Good Data») потрапляють у сховище.

Помилкові записи («Bad Data») відфільтровуються в окремий потік, що запобігає забрудненню звітності некоректними даними.

На рівні виконання (System Errors): Я налаштувала Event Handlers (обробники подій) для події OnError. Якщо під час виконання виникає технічна помилка (розрив з'єднання, блокування таблиці), обробник автоматично перехоплює її та записує повний текст помилки в стовпець ErrorMessage таблиці аудиту.

Звіти про кількість оброблених записів: Завдяки реалізованій архітектурі, таблиця ETL_Log слугує джерелом даних для автоматичних звітів. Аналізуючи записи в цій таблиці, можна отримати інформацію про частоту запусків, тривалість виконання кожного етапу та наявність збоїв, що повністю закриває потребу в аудиті ETL-процесів.

Етап 4. Побудова OLAP-куба (SQL Server Analysis Services)

4.1 Створення проекту SSAS

Для реалізації багатовимірного аналізу даних предметної області «Кабельне телебачення» я виконала наступні кроки в середовищі розробки Microsoft Visual Studio 2022 (SQL Server Data Tools):

1. Створення нового проекту Analysis Services Multidimensional: На початку роботи я створила нове рішення (Solution). Типом проекту було обрано Analysis Services Multidimensional and Data Mining Project, оскільки він дозволяє розробляти класичні OLAP-куби (MOLAP), що відповідає вимогам курсової роботи.

2. Налаштування з'єднання з Data Warehouse: Наступним кроком я налаштувала фізичне з'єднання між сервером аналітики та моїм сховищем даних.

У папці Data Sources я створила нове джерело даних.

У налаштуваннях провайдера (Provider) було вказано підключення до локального SQL Server (localhost).

В якості бази даних джерела я обрала розроблене на попередніх етапах сховище CableTV_DWH.

Для параметрів автентифікації (Impersonation Information) я налаштувала використання сервісного облікового запису (Inherit), щоб забезпечити стабільний доступ до даних під час обробки куба.

3. Створення Data Source View (DSV): Для створення логічної моделі даних, з якою працюватиме куб, я спроектувала представлення джерела даних (Data Source View):

У майстрі створення DSV я обрала створене джерело даних CableTV_DWH.

До діаграми я додала всі необхідні таблиці, що утворюють схему «Зірка» мого сховища:

Таблиці фактів: FactFinancials (фінансові операції) та FactMovieOperations (замовлення фільмів).

Таблиці вимірів: DimSubscriber, DimMovie, DimChannelGroup, DimPaymentMethod та DimDate.

Система автоматично розпізнала та імпортувала зв'язки між первинними та зовнішніми ключами (PK-FK), які були закладені на рівні бази даних SQL. Я перевірила коректність побудови діаграми, переконавшись, що таблиці фактів знаходяться в центрі, а виміри пов'язані з ними відповідними реляціями.

4.2 Проектування Data Source View

Після встановлення з'єднання я перейшла до детального проектування представлення даних, щоб адаптувати структуру DWH під потреби аналітичного куба.

1. Додавання таблиць фактів та вимірів: У середовищі дизайнера DSV я додала необхідні об'єкти з бази даних. Схема була побудована навколо двох центральних таблиць фактів:

FactFinancials — для аналізу фінансових надходжень.

FactMovieOperations — для аналізу замовлень відеоконтенту.

До них я додала таблиці вимірів: DimSubscriber (абоненти), DimMovie (фільми), DimChannelGroup (групи каналів), DimPaymentMethod (методи оплати) та DimDate (календар). Це дозволило сформувати класичну схему «Зірка».

2. Визначення зв'язків між таблицями: Система автоматично зчитала зв'язки (Relationships) на основі зовнішніх ключів (Foreign Keys), створених у базі даних SQL Server. Я перевірила коректність цих зв'язків у візуальному редакторі, переконавшись, що всі виміри коректно приєднані до відповідних таблиць фактів (наприклад, DimSubscriber пов'язаний з обома фактовими таблицями через SubscriberKey, а DimMovie — лише з FactMovieOperations).

3. Створення іменованих обчислень (Named Calculations): Для підвищення зручності роботи кінцевих користувачів я створила іменовані обчислення. Це віртуальні стовпці, які обчислюються на рівні DSV, не змінюючи фізичну базу даних.

Зокрема, це було використано для конкатенації текстових полів (наприклад, для формування повних описів) та форматування числових значень, щоб у звітах вони відображалися у зрозумілому для бізнесу форматі.

4. Створення іменованих запитів (Named Queries): Для випадків, коли пряме використання таблиці було неоптимальним, я замінила стандартні таблиці на іменовані запити (Named Queries). Це дозволило мені написати SQL-запити безпосередньо в DSV, щоб відфільтрувати технічні або неактуальні записи ще до етапу побудови куба, а також обрати лише необхідний набір стовпців для оптимізації продуктивності.

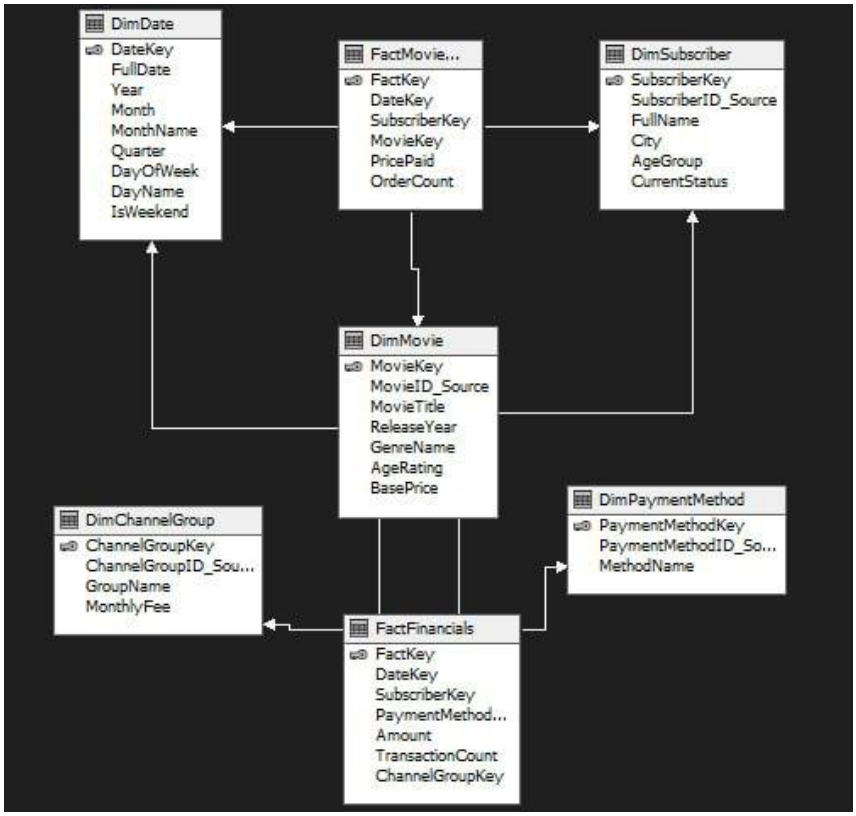


Рис 4 1 Data View Source

4.3 Створення вимірів (Dimensions)

Відповідно до вимог предметної області «Кабельне телебачення», я розробила та налаштувала 5 ключових вимірів, необхідних для багатовимірного аналізу. Для кожного виміру я визначила ключові атрибути (Key Attributes), налаштувала NameColumn для відображення зрозумілих назв замість ідентифікаторів, та побудувала ієрархії.

1. Часовий вимір (DimDate)

Це критично важливий вимір для реалізації Time Intelligence.

Тип виміру: Я встановила властивість Type у значення Time, а для атрибутів визначила відповідні календарні типи (Year, Quarter, Month, Date), що дозволило системі коректно обробляти часові функції MDX.

Ієрархія: Створила ієрархію **Calendar** за рівнем деталізації: Year -> Quarter ->Month->FullDate.

Атрибути: Додала атрибути DayOfWeek, DayName, IsWeekend для аналізу активності абонентів у розрізі днів тижня.

2. Вимір основної сутності (DimSubscriber)

Цей вимір відображає базу абонентів кабельного телебачення.

Ієрархія: Реалізувала географічну ієрархію **Geography**: City->FullName. Це дозволяє аналізувати дані від рівня цілого міста до конкретного клієнта.

Атрибути: Додала атрибути AgeGroup (вікова група) та CurrentStatus (активний/неактивний) для сегментації клієнтської бази.

Оптимізація: Налаштувала Attribute Relationships між містом та ім'ям абонента для прискорення запитів.

3. Вимір контенту (DimMovie)

Вимір для аналізу замовлень фільмів.

Ієрархія: Створила ієрархію **By Genre**: GenreName->MovieTitle, що дозволяє здійснювати навігацію (Drill-down) від загального жанру до конкретного фільму.

Атрибути: Включила атрибути ReleaseYear та AgeRating.

Форматування: Для атрибута BasePrice встановила властивість FormatString у значення Currency, щоб ціни відображалися у грошовому форматі.

4. Вимір груп каналів (DimChannelGroup)

Використовується для аналізу фінансових надходжень від підписок.

Налаштування: Для ключового атрибута налаштувала властивість NameColumn, щоб у звітах відображалася текстова назва пакету (GroupName), а не його технічний ключ.

Атрибути: Додала атрибут MonthlyFee для аналізу вартості пакетів.

5. Вимір методів оплати (DimPaymentMethod)

Допоміжний вимір для фінансового аналізу.

Дозволяє розрізняти транзакції за типом оплати (кредитна картка, банківський переказ тощо) завдяки атрибуту MethodName.

4.4 Створення міри (Measures) та груп мір (Measure Groups)

Для забезпечення повноцінного кількісного аналізу діяльності компанії кабельного телебачення я створила дві групи мір (Measure Groups): Fact Financials (для фінансових показників) та Fact Movie Operations (для операційних показників замовлень).

Враховуючи вимоги до різнопланового аналізу, я реалізувала наступні типи мір:

1. Sum (Сумарні показники): Використовуючи стандартну агрегацію Sum, я створила ключові фінансові показники:

Total Income — загальна сума доходу від усіх транзакцій.

Total Movie Revenue — сумарний дохід від замовлення фільмів.

2. Count (Підрахунок записів): Для оцінки обсягу операцій я використала агрегацію Count:

Transaction Count — загальна кількість фінансових транзакцій за обраний період.

Fact Movie Operations Count — загальна кількість фактів замовлення відеоконтенту.

3. Min/Max (Екстремальні значення): Для аналізу цінової політики я додала міри з агрегаціями Min та Max:

Max Price Paid — максимальна ціна, сплачена за перегляд фільму.

4. Average (Середні значення): Для розуміння середньої вартості послуг я налаштувала агрегацію Average:

AVG Movie Price — середня вартість замовленого фільму (на основі стовпця PricePaid).

5. Distinct Count (Кількість унікальних значень): Це критично важлива міра для маркетингового аналізу, яка дозволяє рахувати не кількість замовлень, а реальну аудиторію:

Unique Viewers — кількість унікальних абонентів, що скористалися послугами. Я налаштувала цю міру на полі SubscriberKey у групі мір Fact Movie Operations з типом агрегації DistinctCount.

6. Calculated Measures (Обчислювані міри): На вкладці Calculations за допомогою мови MDX я створила складні бізнес-метрики, які не зберігаються в базі, а обчислюються в момент запиту:

Темпи зростання (Time Intelligence): Створила міру Income YoY Growth, яка розраховує приріст доходу порівняно з аналогічним періодом минулого року.

Середньозважені показники (Business Metrics): Створила міру Average Transaction Value (Середній чек), яка ділить загальний дохід на кількість транзакцій, автоматично обробляючи ділення на нуль.

4.5 Розробка обчислень в кубі

Для розширення аналітичних можливостей системи я використала мову запитів MDX (Multidimensional Expressions). Всі розрахунки були реалізовані на вкладці Calculations дизайнера куба.

1. Time Intelligence (Аналіз у часі): Я реалізувала обчислення показників у динаміці, щоб порівнювати поточні результати з попередніми періодами:

Year Over Year Growth (YoY %): Створила обчислювану міру [Income YoY Growth].

Логіка: Використала функцію ParallelPeriod, щоб знайти значення доходу за аналогічний період минулого року, відняти його від поточного та розділити на значення минулого року.

Результат: Це дозволило отримувати відсотковий приріст або падіння доходів (наприклад, +12% або -5%). Для зручності я налаштувала властивість Format String у формат Percent.

2. Ranking (Рейтингові оцінки): Для визначення найуспішніших продуктів я використала функціонал іменованих наборів (Named Sets):

Топ-10 фільмів: Створила набір [Top 10 Movies].

Функція: Використала функцію MDX TopCount, яка автоматично відфільтровує та сортує перші 10 фільмів за мірою [Total Movie Revenue]. Це дозволяє у звітах миттєво бачити лідерів прокату без необхідності ручного сортування всього списку.

3. KPI (Key Performance Indicators): На вкладці KPIs я розробила ключовий індикатор ефективності для моніторингу темпів розвитку бізнесу — Growth KPI:

Value Expression (Значення): Прив'язала індикатор до створеної раніше міри [Income YoY Growth].

Goal Expression (Ціль): Встановила цільове значення на рівні 0.05 (тобто зростання на 5% щороку є плановим показником).

Status Expression (Статус): Реалізувала логіку "Світлофора" (Traffic Light) за допомогою оператора Case:

Зелений (1): Якщо зростання перевищує 5%.

Жовтий (0): Якщо зростання позитивне (від 0% до 5%), але не досягає цілі.

Червоний (-1): Якщо спостерігається падіння доходів (менше 0%).

Trend Expression (Тренд): Налаштувала відображення стрілки тренду, порівнюючи поточний приріст із попереднім періодом.

4.6 Налаштування та оптимізація куба

Для забезпечення високої продуктивності системи при зростанні обсягів даних та для зручності роботи різних груп користувачів, я виконала комплекс робіт з оптимізації куба.

1. Створення агрегацій (Aggregations Design)

Щоб пришвидшити виконання запитів, я створила попередньо розраховані агрегації для груп мір.

Використовуючи майстер Aggregation Design Wizard, я згенерувала схему агрегацій.

В якості критерію зупинки процесу побудови я обрала опцію Performance Gain (Приріст продуктивності) і встановила значення 30%.

Це дозволило системі автоматично створити оптимальний набір індексів та зведених даних, що значно скорочує час відгуку на "важкі" запити.

2. Налаштування Storage Mode

Для всіх вимірів та груп мір я використала режим зберігання MOLAP (Multidimensional OLAP).

Я обрала цей режим, оскільки він забезпечує найвищу швидкість виконання запитів за рахунок зберігання даних та агрегацій у стиснутому багатовимірному форматі на сервері аналітики.

3. Створення партицій (Partitions)

Для оптимізації обробки великої таблиці фактів FactMovieOperations (яка містить дані за період 2020–2030 років) я застосувала механізм партиціювання:

Розділила єдину фізичну таблицю на дві логічні частини (партиції):

FactMovie_2020_2024 — архівні дані за перші 5 років.

FactMovie_2025_2030 — актуальні та майбутні дані.

Реалізацію виконала через Query Binding, прописавши прямі SQL-запити з фільтрацією по ключу дати (WHERE DateKey < ...). Це дозволяє серверу при процесингу завантажувати дані паралельно, а при запитах — сканувати лише ті партиції, де є релевантні дані.

4. Оптимізація атрибутних зв'язків (Attribute Relationships)

У редакторі вимірів (наприклад, у DimSubscriber) я перевірила та налаштувала прямі зв'язки між атрибутами (наприклад, City->Subscriber Key). Це запобігає зайвим обчисленням ("Crossjoin") під час агрегації даних та покращує продуктивність MDX-запитів.

5. Створення Perspectives (Перспективи)

Щоб спростити інтерфейс для кінцевих користувачів і не перевантажувати їх зайвими даними, я створила дві перспективи (підмножини куба):

Financial View: Відображає лише фінансові показники (Fact Financials) та пов'язані з ними виміри, приховуючи деталі про фільми. Призначена для фінансових аналітиків.

Content View (або Content Manager View): Фокусується на операціях з фільмами (Fact Movie Operations) та абонентах, приховуючи фінансову звітність. Призначена для контент-менеджерів.

4.7 Розгортання та тестування куба

Фінальним етапом роботи з OLAP-структурою стало перенесення розробленого рішення на аналітичний сервер та наповнення його реальними даними.

1. Розгортання куба на сервері SSAS (Deployment): Після завершення проектування структури я виконала процедуру розгортання (Deploy) проекту з середовища Visual Studio.

У властивостях проекту перевірила налаштування цільового сервера (Target Server), вказавши localhost.

Запустила процес розгортання, який успішно створив базу даних Riznyk_Maria_SSAS та всі необхідні об'єкти (куб, виміри, джерела даних) на сервері Analysis Services. Статус операції — Deployment Completed Successfully.

2. Обробка куба (Process Full): Сам по собі процес розгортання створює лише структуру (метадані). Для завантаження даних я виконала повну обробку (Process Full):

Ініціювала команду Process для всього проекту.

У вікні Process Progress контролювала хід виконання завантаження. Система успішно зчитала дані з SQL-сховища. Зокрема, було підтверджено завантаження:

Таблиці фактів FactFinancials (450 000 рядків).

Двох партицій таблиці FactMovieOperations (сумарно 1 000 000 рядків).

Усіх таблиць вимірів (включно з 100 000 абонентів).

Процес завершився зі статусом Process Succeeded, що підтвердило коректність налаштованих зв'язків та запитів.

3. Перевірка коректності даних: Для верифікації результатів я скористалася вбудованим інструментом Browser у конструкторі куба:

Виконала підключення до розгорнутого куба (Reconnect).

Побудувала тестові зведені таблиці (Pivot Tables), перетягнувши міри Total Income та Order Count у розрізі часового виміру DimDate (роки та місяці) та жанрів DimMovie.

Переконалася, що дані агрегуються коректно, порожні значення (null) відсутні там, де їх не має бути, а ієрархії розгортаються правильно.

Перевірила роботу KPI та форматування обчислюваних мір (відображення відсотків та валюти).

4. Тестування продуктивності запитів: Під час роботи з вкладкою Browser я протестувала швидкість відгуку системи. Завдяки попередньо налаштованим агрегаціям (на 30%) та партиціюванню таблиці фактів, складні запити (наприклад, розрахунок YoY Growth або фільтрація Топ-10 фільмів) виконуються миттєво, без помітних затримок.

5. Збереження рішення: Після успішного тестування я зберегла фінальну версію Solution у Visual Studio, що фактично є резервною копією вихідного коду проекту (файли .dim, .cube, .dsv), готовою до подальшого супроводу або перенесення на інший сервер.

Етап 5. Створення аналітичних звітів (SQL Server Reporting Services)

5.1 Створення проекту SSRS

Для реалізації фінального етапу курсової роботи, пов'язаного з візуалізацією даних та побудовою звітності, я використала інструментарій SQL Server Reporting Services (SSRS).

1. Створення нового проекту Report Server Project: У середовищі Visual Studio (SQL Server Data Tools) я створила окреме рішення для розробки звітів.

В якості типу проекту обрала Report Server Project, що дозволяє створювати файли визначення звітів (.rdl) та розгортати їх на сервері звітів.

2. Створення Shared Data Source (Спільного джерела даних): Щоб забезпечити уніфікацію та уникнути дублювання налаштувань підключення у кожному окремому звіті, я створила спільне джерело даних (Shared Data Source):

У папці Shared Data Sources додала новий елемент та назвала його CableTV.

Це дозволяє централізовано керувати підключенням: якщо зміниться адреса сервера, мені потрібно буде оновити налаштування лише в одному місці, а не в кожному з 5 звітів.

3. Налаштування підключення до OLAP-куба: Найважливішим етапом було налаштування коректної взаємодії між системою звітності та аналітичним кубом:

У налаштуваннях джерела даних я змінила тип (Type) з класичного «Microsoft SQL Server» на «Microsoft OLE DB Provider for Analysis Services». Це специфічний провайдер, необхідний для роботи з багатовимірними базами даних.

Я виконала перевірку з'єднання (Test Connection), яка пройшла успішно, підтвердивши, що SSRS "бачить" куб та його метадані (міри та виміри).

5.2 Розробка звітів

Використовуючи інструментарій SSRS, я розробила комплект із 5 аналітичних звітів, кожен з яких демонструє різні підходи до візуалізації даних та відповідає специфічним бізнес-потребам.

Табличний звіт (Table Report) — «Рейтинг популярності кінофільмів»

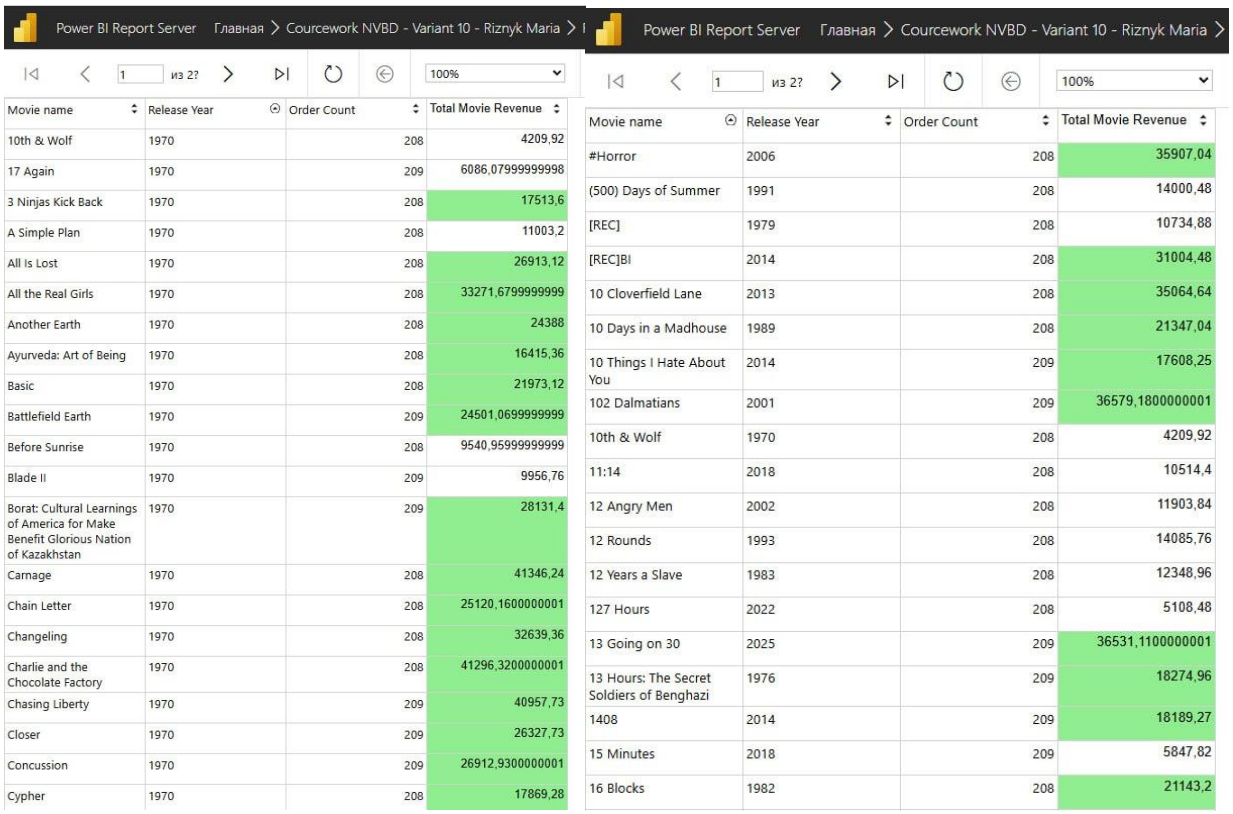
Я створила детальний звіт для аналізу успішності окремих фільмів.

Структура: Використала елемент **Table**, де в рядках виводяться назви фільмів (Movie Title), рік випуску, кількість замовлень (Order Count) та загальний дохід (Total Movie Revenue).

Функціонал: Додала **Interactive Sorting** (Інтерактивне сортування), що дозволяє користувачеві кліком на заголовок стовпця сортувати фільми за алфавітом або за прибутковістю.

Реалізувала **Totals** (Підсумковий рядок) для відображення загальної суми доходу.

Налаштувала фільтрацію даних через параметр жанру.



Movie name	Release Year	Order Count	Total Movie Revenue
10th & Wolf	1970	208	4209,92
17 Again	1970	209	6086,07999999998
3 Ninjas Kick Back	1970	208	17513,6
A Simple Plan	1970	208	11003,2
All Is Lost	1970	208	26913,12
All the Real Girls	1970	208	33271,6799999999
Another Earth	1970	208	24388
Ayurveda: Art of Being	1970	208	16415,36
Basic	1970	208	21973,12
Battlefield Earth	1970	209	24501,0699999999
Before Sunrise	1970	208	9540,95999999999
Blade II	1970	209	9956,76
Borat: Cultural Learnings of America for Make Benefit Glorious Nation of Kazakhstan	1970	209	28131,4
Carnage	1970	208	41346,24
Chain Letter	1970	208	25120,1600000001
Changeling	1970	208	32639,36
Charlie and the Chocolate Factory	1970	208	41296,3200000001
Chasing Liberty	1970	209	40957,73
Closer	1970	209	26327,73
Concussion	1970	209	26912,9300000001
Cypher	1970	208	17869,28

Power BI Report Server
Главная
Courcework NVBD - Variant 10 - Riznyk Maria
из 2?
100%

Movie name	Release Year	Order Count	Total Movie Revenue
#Horror	2006	208	35907,04
(500) Days of Summer	1991	208	14000,48
[REC]	1979	208	10734,88
[REC]3I	2014	208	31004,48
10 Cloverfield Lane	2013	208	35064,64
10 Days in a Madhouse	1989	208	21347,04
10 Things I Hate About You	2014	209	17608,25
102 Dalmatians	2001	209	36579,1800000001
10th & Wolf	1970	208	4209,92
11:14	2018	208	10514,4
12 Angry Men	2002	208	11903,84
12 Rounds	1993	208	14085,76
12 Years a Slave	1983	208	12348,96
127 Hours	2022	208	5108,48
13 Going on 30	2025	209	36531,1100000001
13 Hours: The Secret Soldiers of Benghazi	1976	209	18274,96
1408	2014	209	18189,27
15 Minutes	2018	209	5847,82
16 Blocks	1982	208	21143,2

Рис 5 1

Матричний звіт (Matrix Report) — «Споживча орієнтація абонентів»

Для виявлення залежностей між віком аудиторії та їхніми вподобаннями я розробила крос-таблицю.

Структура: Використала елемент **Matrix**.

Рядки (Row Groups): Вікові групи абонентів (Age Group).

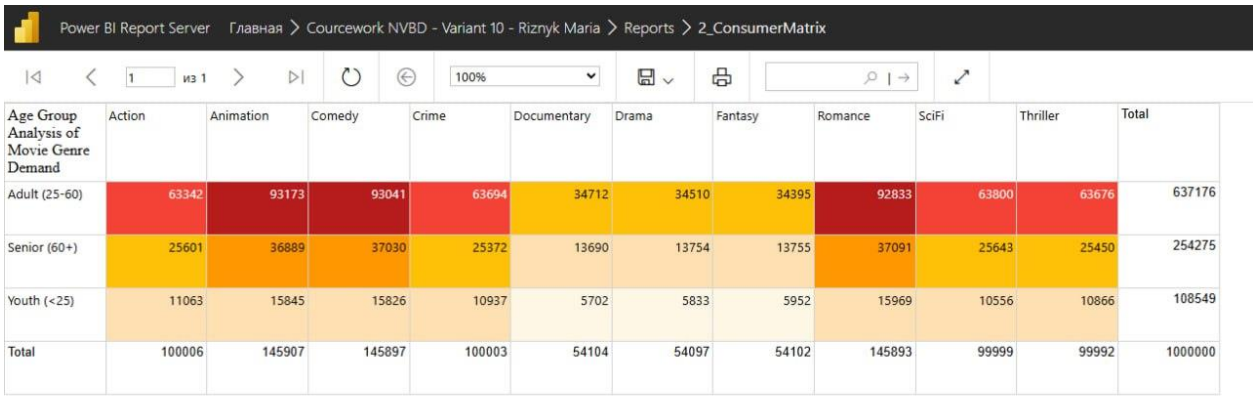
Колонки (Column Groups): Жанри фільмів (Genre Name).

Дані (Values): Кількість замовлень (Order Count).

Особливості:

Додала проміжні підсумки (Subtotals) як по рядках, так і по стовпцях.

Реалізувала ефект **Heatmap** (Теплова карта) через умовне форматування: клітинки з великою кількістю замовлень автоматично забарвлюються в більш насичені кольори, що дозволяє миттєво бачити найпопулярніші сегменти.



Age Group Analysis of Movie Genre Demand	Action	Animation	Comedy	Crime	Documentary	Drama	Fantasy	Romance	SciFi	Thriller	Total
Adult (25-60)	63342	93173	93041	63694	34712	34510	34395	92833	63800	63676	637176
Senior (60+)	25601	36889	37030	25372	13690	13754	13755	37091	25643	25450	254275
Youth (<25)	11063	15845	15826	10937	5702	5833	5952	15969	10556	10866	108549
Total	100006	145907	145897	100003	54104	54097	54102	145893	99999	99992	1000000

Рис 5 2

Звіт з діаграмами (Chart Report) — «Аналіз підключень каналів»

Для візуального порівняння фінансових показників різних пакетів каналів я створила звіт, що містить три типи діаграм:

Pie Chart (Кругова): Демонструє частку доходу (Revenue Share) від кожної групи каналів.

Column Chart (Стовпчикова): Показує абсолютну кількість транзакцій по кожному пакету.

Line Chart (Лінійна): Відображає тренд доходів у часі (по роках) для кожної групи каналів окремою лінією.

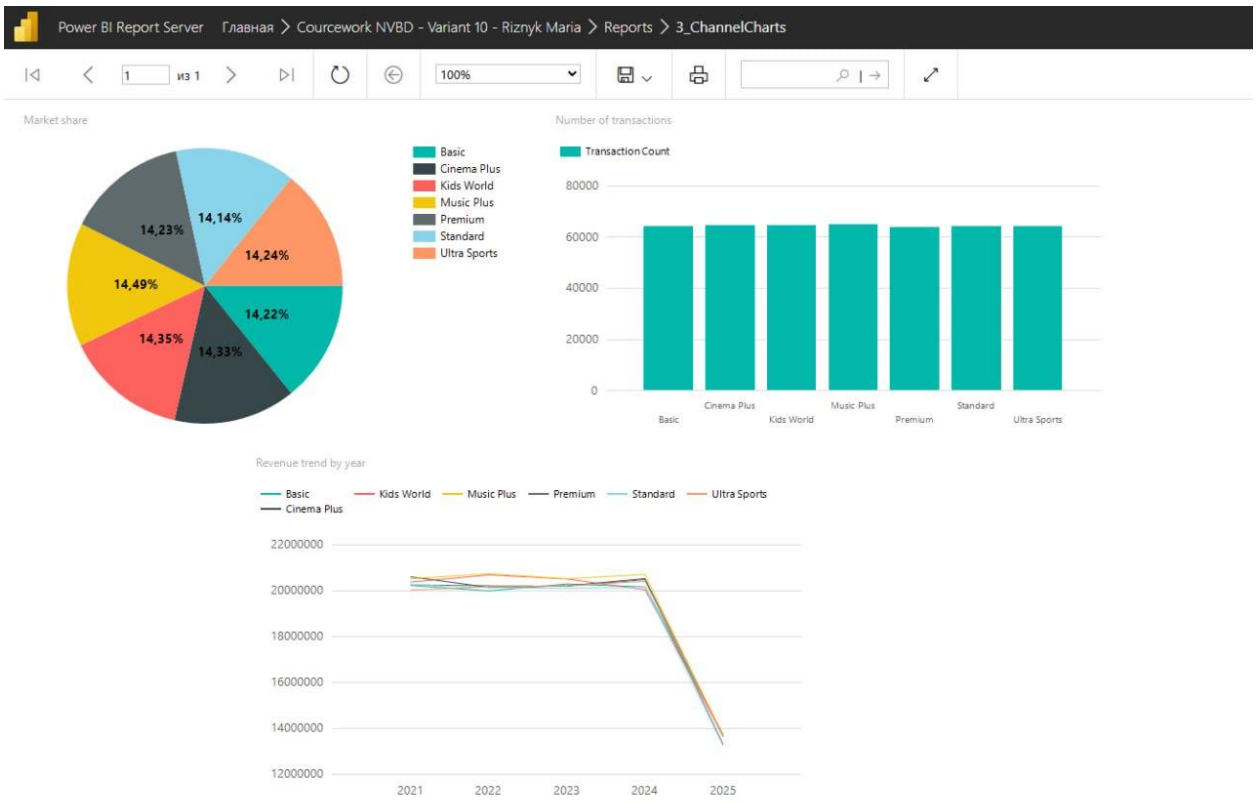


Рис 5 3

Дашборд (Dashboard) — «Динаміка абонентської бази»

Я розробила комплексну інформаційну панель для керівництва (Executive Dashboard), яка об'єднує ключові метрики на одній сторінці.

Gauge: Використала радіальний датчик (спідометр) для візуалізації показника Total Income. для швидкої оцінки ситуації.

Sparklines (Спарклайни): У таблицю жанрів вбудувала міні-графіки, які показують тренд продажів кожного жанру протягом року безпосередньо в клітинці таблиці.

Area Chart (Діаграма з областями): У нижній частині дашборду розмістила графік росту унікальних глядачів (Unique Viewers), що дозволяє оцінити динаміку розширення клієнтської бази.

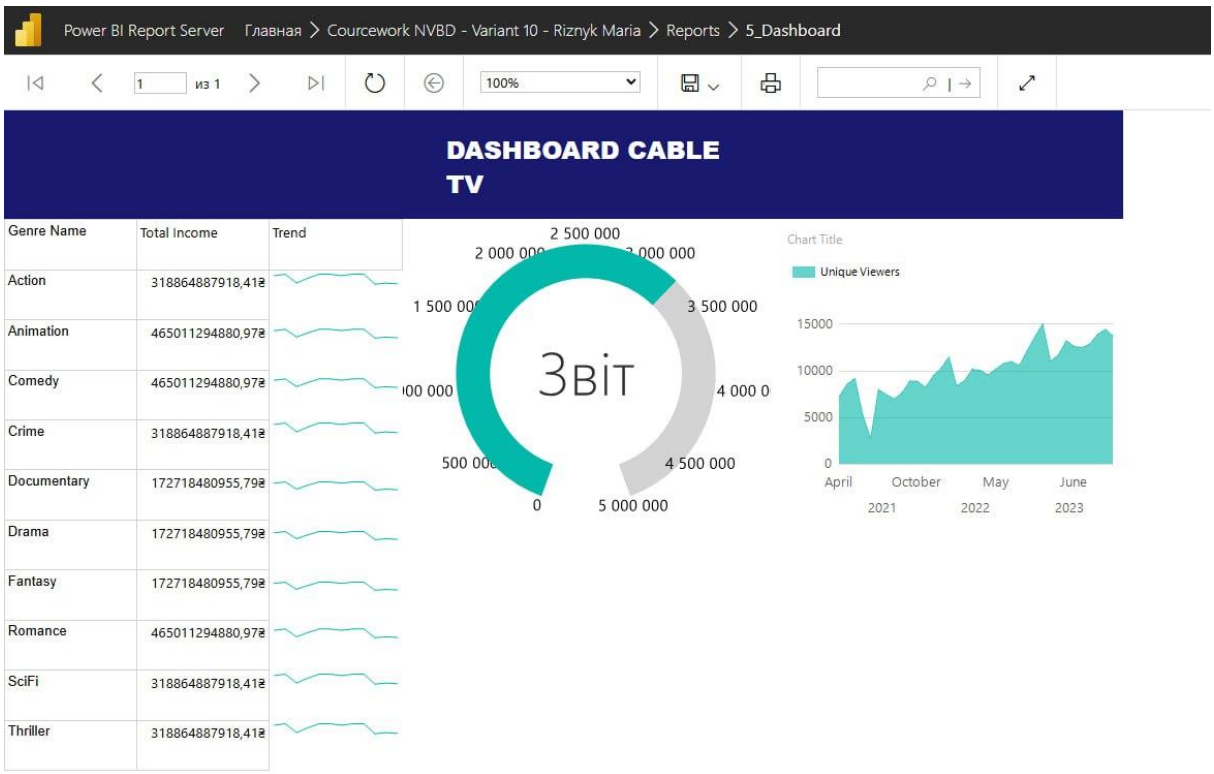


Рис 5 4

Деталізований звіт з drill-down — «Фінансовий звіт за місяць»

Для бухгалтерського аналізу я створила багаторівневий звіт з можливістю заглиблення в дані.

Ієрархія: Налаштувала групування: Year->Quarter->Month.

Drill-down: Реалізувала механізм згортання/розгортання груп. За замовчуванням користувач бачить лише роки. Натискання на «плюс» розкриває квартали, а потім — місяці.

Агрегація: На кожному рівні ієрархії автоматично розраховуються проміжні підсумки (Subtotals) по доходу та кількості транзакцій.

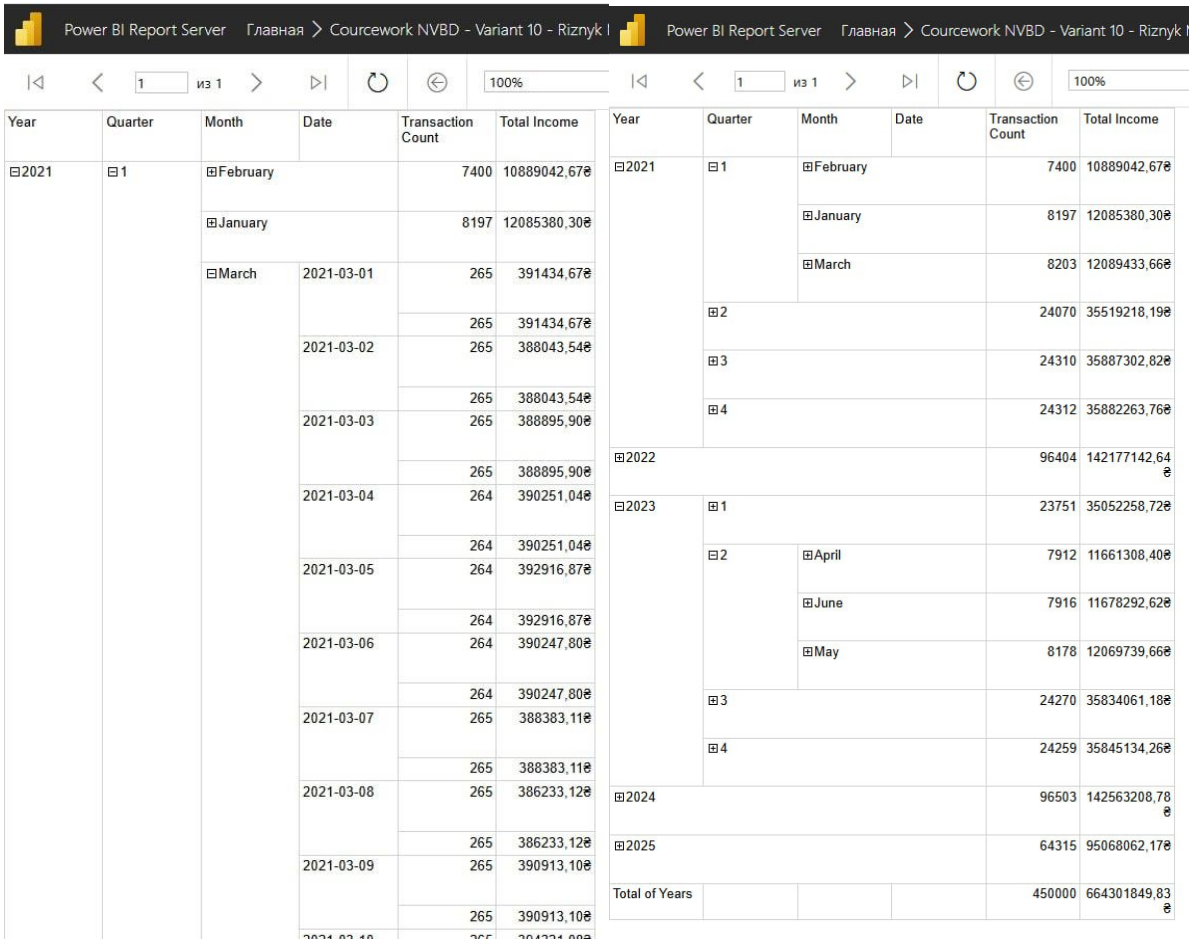


Рис 5 5

5.4 Додаткова функціональність звітів

Для підвищення інформативності, візуальної привабливості та зручності користування звітами я реалізувала ряд додаткових інтерактивних функцій та правил форматування.

1. Conditional Formatting (Умовне форматування): Я налаштувала динамічну зміну властивостей об'єктів звіту залежно від значень даних, використовуючи вирази (Expressions):

Кольорове виділення лідерів: У табличному звіті «Рейтинг фільмів» я використала функцію IIF для властивості BackgroundColor. Фільми, дохід від яких перевищує встановлений поріг, автоматично підсвічуються зеленим кольором, що дозволяє миттєво акцентувати увагу на найуспішніших продуктах.

Ефект Heatmap (Теплова карта): У матричному звіті «Споживча орієнтація» я реалізувала складнішу логіку за допомогою функції Switch. Колір комірок змінюється від білого до насиченого червоного залежно від кількості замовлень, створюючи візуальну карту активності різних вікових груп.

2. Interactive Features (Інтерактивні можливості): Для покращення користувацького досвіду (UX) я додала елементи керування, що дозволяють користувачеві взаємодіяти зі звітом:

Sorting (Інтерактивне сортування): У звіті по фільмах я активувала функцію Interactive Sorting для заголовків стовпців. Це додало стрілки сортування в інтерфейс звіту, дозволяючи користувачеві самостійно впорядковувати дані за алфавітом назв або за спаданням прибутку без необхідності перегенерувати звіт.

Toggle Items (Розгортання/згортання): У фінансовому звіті я використала механізм перемикання видимості (Visibility Toggle). Я прив'язала видимість вкладених груп (кварталів та місяців) до батьківських елементів. Це дозволило створити компактний звіт, де початково відображаються лише річні підсумки, а деталізація розкривається кліком на значок «плюс» (+) (Drill-down ефект).

5.5 Розгортання звітів

На завершальному етапі я виконала публікацію розроблених звітів у виробниче середовище та налаштувала параметри доступу і продуктивності.

1. Публікація звітів на Report Server Після перевірки коректності роботи звітів у Visual Studio я виконала процедуру розгортання (**Deploy**).

У налаштуваннях проекту вказала TargetServerURL свого локального сервера звітів.

Після успішного завершення процесу всі файли звітів (.rdl) та спільні джерела даних (.rds) стали доступними через веб-інтерфейс **Web Portal URL** (Report Manager).

2. Створення папок для організації звітів Для підтримки порядку на сервері я структурувала контент:

Під час розгортання налаштувала властивість TargetReportFolder, створивши окремий каталог **CableTV_Reports**.

Це дозволило відокремити звіти моєї курсової роботи від інших проектів на сервері.

3. Налаштування безпеки та доступу Через веб-інтерфейс порталу звітів я налаштувала рольову модель безпеки:

Для теки CableTV_Reports призначила роль **Browser** для групи користувачів (наприклад, менеджерів), що дозволяє їм переглядати звіти, але забороняє вносити зміни.

Собі як розробнику залишила роль **Content Manager** для повного керування об'єктами.

4. Налаштування кешування звітів Щоб зменшити навантаження на OLAP-сервер та прискорити відкриття «важких» звітів (зокрема Дашборду):

У налаштуваннях звіту (Processing Options) я увімкнула тимчасове кешування копії звіту.

Встановила час життя кешу (Expiration) на 30 хвилин, що є оптимальним компромісом між актуальністю даних та швидкістю завантаження.

5. Створення Subscriptions (Підписки) Для автоматизації звітності я створила підписку (Subscription) для «Фінансового звіту»:

Налаштувала розклад (Schedule): автоматичне формування звіту в кінці місяця.

Обрала метод доставки (**Delivery Method**): збереження файлу у форматі **PDF** у спільну мережеву папку (Windows File Share), що імітує автоматичну підготовку документів для бухгалтерії.

Висновок

У ході виконання курсової роботи я спроектувала та реалізувала комплексну систему бізнес-аналітики (BI) для предметної області «Кабельне телебачення». Робота охопила повний життєвий цикл обробки корпоративних даних: від створення реляційної бази даних та побудови сховища даних (Data Warehouse) за схемою «Зірка» до розгортання багатовимірного OLAP-куба. Це дозволило консолідувати великі масиви інформації про фінансові транзакції, замовлення контенту та абонентську базу в єдину структуру, оптимізовану для аналітичних запитів.

За допомогою технологій SQL Server Analysis Services (SSAS) я розробила потужну аналітичну модель, що включає ключові виміри (час, географія, жанри) та бізнес-показники. Впровадження складних MDX-обчислень, таких як Time Intelligence (YoY Growth), та налаштування KPI дозволило реалізувати глибокий аналіз динаміки доходів та ефективності продажів. Окрім того, були застосовані методи оптимізації продуктивності, зокрема створення агрегацій та партиціювання таблиць фактів, що забезпечило миттєвий відгук системи навіть на складні запити.

Фінальним етапом стала візуалізація даних засобами SQL Server Reporting Services (SSRS). Я створила пакет інтерактивних звітів, включаючи деталізовані фінансові таблиці з функцією drill-down, матриці споживчих вподобань та комплексний управлінський дашборд. Результатом проекту є готова до використання система підтримки прийняття рішень, яка надає менеджменту компанії інструменти для оперативного моніторингу ключових показників ефективності та стратегічного планування розвитку бізнесу.

Бібліографічний список

1. Microsoft SQL Server Integration Services (SSIS) Documentation
2. Microsoft SQL Server Analysis Services (SSAS) Documentation
3. Microsoft SQL Server Reporting Services (SSRS) Documentation
4. "The Data Warehouse Toolkit" - Ralph Kimball
5. "Microsoft SQL Server 2019: A Beginner's Guide" - Dusan Petkovic
6. Golfarelli M., Rizzi S. Data Warehouse Design: Modern Principles and Methodologies. — McGraw-Hill Education, 2009.
7. Microsoft Learn. SQL Server Data Warehousing and Analytics.

Додатки

Створення таблиць для бази даних CableTV

```
IF DB_ID(N'CableTV') IS NULL
BEGIN
    CREATE DATABASE CableTV;
END
GO
USE CableTV;
GO

IF SCHEMA_ID('ref') IS NULL EXEC('CREATE SCHEMA ref');
GO

CREATE TABLE ref.SubscriberStatus (
    SubscriberStatusID tinyint IDENTITY(1,1) CONSTRAINT PK_SubscriberStatus PRIMARY KEY,
    StatusName nvarchar(30) NOT NULL CONSTRAINT UQ_SubscriberStatus UNIQUE
);

CREATE TABLE ref.PaymentMethod (
    PaymentMethodID tinyint IDENTITY(1,1) CONSTRAINT PK_PaymentMethod PRIMARY KEY,
    MethodName nvarchar(30) NOT NULL CONSTRAINT UQ_PaymentMethod UNIQUE
);

CREATE TABLE ref.Genre (
    GenreID smallint IDENTITY(1,1) CONSTRAINT PK_Genre PRIMARY KEY,
    GenreName nvarchar(50) NOT NULL CONSTRAINT UQ_Genre UNIQUE
);

CREATE TABLE ref.ChannelCategory (
    ChannelCategoryID smallint IDENTITY(1,1) CONSTRAINT PK_ChannelCategory PRIMARY KEY,
    CategoryName nvarchar(50) NOT NULL CONSTRAINT UQ_ChannelCategory UNIQUE
);

CREATE TABLE ref.InvoiceStatus (
    InvoiceStatusID tinyint IDENTITY(1,1) CONSTRAINT PK_InvoiceStatus PRIMARY KEY,
    StatusName nvarchar(30) NOT NULL CONSTRAINT UQ_InvoiceStatus UNIQUE
);
GO

CREATE TABLE dbo.Subscriber (
    SubscriberID int IDENTITY(1,1) CONSTRAINT PK_Subscriber PRIMARY KEY,
    ExternalContractNo nvarchar(20) NOT NULL CONSTRAINT UQ_Subscriber_Contract UNIQUE,
    FirstName nvarchar(50) NOT NULL,
    LastName nvarchar(50) NOT NULL,
    Phone nvarchar(20) NOT NULL,
    Email nvarchar(120) NULL,
    BirthDate date NULL,
    City nvarchar(60) NOT NULL,
    Street nvarchar(100) NOT NULL,
    HouseNo nvarchar(10) NOT NULL,
    ApartmentNo nvarchar(10) NULL,
    CreatedAt date NOT NULL,
    ClosedAt date NULL,
    SubscriberStatusID tinyint NOT NULL,
    CONSTRAINT FK_Subscriber_Status FOREIGN KEY (SubscriberStatusID)
        REFERENCES ref.SubscriberStatus(SubscriberStatusID),
    CONSTRAINT CK_Subscriber_Dates CHECK (CreatedAt <= '2025-12-31' AND (ClosedAt IS NULL
OR ClosedAt <= '2025-12-31') AND (ClosedAt IS NULL OR ClosedAt >= CreatedAt)),
    CONSTRAINT CK_Subscriber_BirthDate CHECK (BirthDate IS NULL OR (BirthDate >= '1930-
01-01' AND BirthDate <= '2007-12-31'))
);
```

```sql
CREATE TABLE dbo.Channel (
    ChannelID int IDENTITY(1,1) CONSTRAINT PK_Channel PRIMARY KEY,
    ChannelName nvarchar(100) NOT NULL CONSTRAINT UQ_Channel_Name UNIQUE,
    ChannelCategoryID smallint NOT NULL,
    IsHD bit NOT NULL,
    IsActive bit NOT NULL,
    CONSTRAINT FK_Channel_Category FOREIGN KEY (ChannelCategoryID)
        REFERENCES ref.ChannelCategory(ChannelCategoryID)
);

CREATE TABLE dbo.ChannelGroup (
    ChannelGroupID int IDENTITY(1,1) CONSTRAINT PK_ChannelGroup PRIMARY KEY,
    GroupName nvarchar(100) NOT NULL CONSTRAINT UQ_ChannelGroup_Name UNIQUE,
    MonthlyFee decimal(10,2) NOT NULL,
    IsActive bit NOT NULL,
    CONSTRAINT CK_ChannelGroup_Fee CHECK (MonthlyFee >= 0)
);

CREATE TABLE dbo.ChannelGroupChannel (
    ChannelGroupID int NOT NULL,
    ChannelID int NOT NULL,
    CONSTRAINT PK_ChannelGroupChannel PRIMARY KEY (ChannelGroupID, ChannelID),
    CONSTRAINT FK_CGChannel_Group FOREIGN KEY (ChannelGroupID) REFERENCES
dbo.ChannelGroup(ChannelGroupID),
    CONSTRAINT FK_CGChannel_Channel FOREIGN KEY (ChannelID) REFERENCES
dbo.Channel(ChannelID)
);

CREATE TABLE dbo.SubscriberChannelGroup (
    SubscriberChannelGroupID bigint IDENTITY(1,1) CONSTRAINT PK_SubscriberChannelGroup
PRIMARY KEY,
    SubscriberID int NOT NULL,
    ChannelGroupID int NOT NULL,
    StartDate date NOT NULL,
    EndDate date NULL,
    CONSTRAINT FK_SubCG_Subscriber FOREIGN KEY (SubscriberID) REFERENCES
dbo.Subscriber(SubscriberID),
    CONSTRAINT FK_SubCG_Group FOREIGN KEY (ChannelGroupID) REFERENCES
dbo.ChannelGroup(ChannelGroupID),
    CONSTRAINT CK_SubCG_Dates CHECK (StartDate <= '2025-12-31' AND (EndDate IS NULL OR
(EndDate <= '2025-12-31' AND EndDate >= StartDate)))
);
GO

CREATE TABLE dbo.Movie (
    MovieID int IDENTITY(1,1) CONSTRAINT PK_Movie PRIMARY KEY,
    MovieTitle nvarchar(200) NOT NULL,
    ReleaseYear smallint NOT NULL,
    DurationMin smallint NOT NULL,
    BasePrice decimal(10,2) NOT NULL,
    AgeRating nvarchar(10) NULL,
    CONSTRAINT CK_Movie_Year CHECK (ReleaseYear BETWEEN 1950 AND 2025),
    CONSTRAINT CK_Movie_Duration CHECK (DurationMin BETWEEN 30 AND 300),
    CONSTRAINT CK_Movie_Price CHECK (BasePrice >= 0)
);

CREATE TABLE dbo.MovieGenre (
    MovieID int NOT NULL,
    GenreID smallint NOT NULL,
    CONSTRAINT PK_MovieGenre PRIMARY KEY (MovieID, GenreID),
    CONSTRAINT FK_MovieGenre_Movie FOREIGN KEY (MovieID) REFERENCES dbo.Movie(MovieID),
    CONSTRAINT FK_MovieGenre_Genre FOREIGN KEY (GenreID) REFERENCES ref.Genre(GenreID)
);
```

```
CREATE TABLE dbo.MovieOrder (
    MovieOrderID bigint IDENTITY(1,1) CONSTRAINT PK_MovieOrder PRIMARY KEY,
    SubscriberID int NOT NULL,
    MovieID int NOT NULL,
    OrderDateTime datetime2(0) NOT NULL,
    PricePaid decimal(10,2) NOT NULL,
    CONSTRAINT FK_Order_Subscriber FOREIGN KEY (SubscriberID) REFERENCES
dbo.Subscriber(SubscriberID),
    CONSTRAINT FK_Order_Movie FOREIGN KEY (MovieID) REFERENCES dbo.Movie(MovieID),
    CONSTRAINT CK_Order_Date CHECK (OrderDateTime >= '2021-01-01' AND OrderDateTime <
'2026-01-01'),
    CONSTRAINT CK_Order_Price CHECK (PricePaid >= 0)
);
GO

CREATE TABLE dbo.Invoice (
    InvoiceID bigint IDENTITY(1,1) CONSTRAINT PK_Invoice PRIMARY KEY,
    SubscriberID int NOT NULL,
    InvoiceMonth date NOT NULL, -- store first day of month
    IssuedAt date NOT NULL,
    DueDate date NOT NULL,
    InvoiceStatusID tinyint NOT NULL,
    TotalAmount decimal(12,2) NOT NULL DEFAULT(0),
    CONSTRAINT FK_Invoice_Subscriber FOREIGN KEY (SubscriberID) REFERENCES
dbo.Subscriber(SubscriberID),
    CONSTRAINT FK_Invoice_Status FOREIGN KEY (InvoiceStatusID) REFERENCES
ref.InvoiceStatus(InvoiceStatusID),
    CONSTRAINT CK_Invoice_Dates CHECK (InvoiceMonth >= '2021-01-01' AND InvoiceMonth <=
'2025-12-01' AND IssuedAt <= '2025-12-31' AND DueDate <= '2025-12-31' AND DueDate >=
IssuedAt),
    CONSTRAINT CK_Invoice_Total CHECK (TotalAmount >= 0)
);

CREATE TABLE dbo.InvoiceLine (
    InvoiceLineID bigint IDENTITY(1,1) CONSTRAINT PK_InvoiceLine PRIMARY KEY,
    InvoiceID bigint NOT NULL,
    LineType nvarchar(30) NOT NULL,
    SourceID bigint NULL,
    Description nvarchar(200) NOT NULL,
    Amount decimal(12,2) NOT NULL,
    CONSTRAINT FK_InvoiceLine_Invoice FOREIGN KEY (InvoiceID) REFERENCES
dbo.Invoice(InvoiceID),
    CONSTRAINT CK_InvoiceLine_Amount CHECK (Amount <> 0)
);

CREATE TABLE dbo.Payment (
    PaymentID bigint IDENTITY(1,1) CONSTRAINT PK_Payment PRIMARY KEY,
    InvoiceID bigint NOT NULL,
    PaymentDateTime datetime2(0) NOT NULL,
    Amount decimal(12,2) NOT NULL,
    PaymentMethodID tinyint NOT NULL,
    CONSTRAINT FK_Payment_Invoice FOREIGN KEY (InvoiceID) REFERENCES
dbo.Invoice(InvoiceID),
    CONSTRAINT FK_Payment_Method FOREIGN KEY (PaymentMethodID) REFERENCES
ref.PaymentMethod(PaymentMethodID),
    CONSTRAINT CK_Payment_Date CHECK (PaymentDateTime >= '2021-01-01' AND PaymentDateTime
< '2026-01-01'),
    CONSTRAINT CK_Payment_Amount CHECK (Amount > 0)
);
GO

CREATE INDEX IX_Order_Date ON dbo.MovieOrder(OrderDateTime) INCLUDE (MovieID,
SubscriberID, PricePaid);
```

```
CREATE INDEX IX_Order_Movie ON dbo.MovieOrder(MovieID) INCLUDE (OrderDateTime, PricePaid,
SubscriberID);

CREATE INDEX IX_SubCG_Subscriber ON dbo.SubscriberChannelGroup(SubscriberID, StartDate)
INCLUDE (ChannelGroupID, EndDate);
CREATE INDEX IX_SubCG_Group ON dbo.SubscriberChannelGroup(ChannelGroupID, StartDate)
INCLUDE (SubscriberID, EndDate);

CREATE INDEX IX_Invoice_Month ON dbo.Invoice(InvoiceMonth) INCLUDE (SubscriberID,
TotalAmount, InvoiceStatusID);
CREATE INDEX IX_Payment_Invoice ON dbo.Payment(InvoiceID) INCLUDE (PaymentDateTime,
Amount);
GO
```

Налаштування CableTV та тестування на малому обсягу даних

```
USE CableTV;
GO

SELECT s.name AS [schema], t.name AS [table]
FROM sys.tables t
JOIN sys.schemas s ON s.schema_id = t.schema_id
WHERE (s.name='dbo' AND t.name IN
('Subscriber','Channel','ChannelGroup','ChannelGroupChannel','SubscriberChannelGroup',

'Movie','MovieGenre','MovieOrder','Invoice','InvoiceLine','Payment'))
   OR (s.name='ref' AND t.name IN
('SubscriberStatus','PaymentMethod','Genre','ChannelCategory','InvoiceStatus'))
ORDER BY s.name, t.name;

SELECT
  OBJECT_SCHEMA_NAME(i.object_id) AS [schema],
  OBJECT_NAME(i.object_id) AS [table],
  i.name AS pk_name
FROM sys.indexes i
WHERE i.is_primary_key = 1
  AND OBJECT_SCHEMA_NAME(i.object_id) IN ('dbo','ref')
ORDER BY [schema],[table];

SELECT
  OBJECT_SCHEMA_NAME(fk.parent_object_id) AS [schema],
  OBJECT_NAME(fk.parent_object_id) AS [table],
  fk.name AS fk_name,
  OBJECT_SCHEMA_NAME(fk.referenced_object_id) AS ref_schema,
  OBJECT_NAME(fk.referenced_object_id) AS ref_table
FROM sys.foreign_keys fk
WHERE OBJECT_SCHEMA_NAME(fk.parent_object_id) IN ('dbo','ref')
ORDER BY [schema],[table], fk.name;

SELECT
  OBJECT_SCHEMA_NAME(i.object_id) AS [schema],
  OBJECT_NAME(i.object_id) AS [table],
  i.name AS index_name
FROM sys.indexes i
WHERE i.name IN
('IX_Order_Date','IX_Order_Movie','IX_SubCG_Subscriber','IX_SubCG_Group','IX_Invoice_Mont
h','IX_Payment_Invoice')
ORDER BY [schema],[table], index_name;

IF NOT EXISTS (SELECT 1 FROM ref.SubscriberStatus WHERE StatusName='Active')
    INSERT INTO ref.SubscriberStatus(StatusName) VALUES
('Active'),('Suspended'),('Closed');

IF NOT EXISTS (SELECT 1 FROM ref.PaymentMethod WHERE MethodName='Card')
    INSERT INTO ref.PaymentMethod(MethodName) VALUES ('Card'),('BankTransfer'),('Cash');
```

```
IF NOT EXISTS (SELECT 1 FROM ref.InvoiceStatus WHERE StatusName='Open')
    INSERT INTO ref.InvoiceStatus(StatusName) VALUES
('Open'),('Paid'),('Overdue'),('Void');

IF NOT EXISTS (SELECT 1 FROM ref.ChannelCategory WHERE CategoryName='Movies')
    INSERT INTO ref.ChannelCategory(CategoryName) VALUES
('News'),('Sport'),('Kids'),('Movies'),('Music');

IF NOT EXISTS (SELECT 1 FROM ref.Genre WHERE GenreName='Drama')
    INSERT INTO ref.Genre(GenreName) VALUES ('Drama'),('Comedy'),('Action');

DECLARE @ActiveStatus tinyint = (SELECT TOP 1 SubscriberStatusID FROM
ref.SubscriberStatus WHERE StatusName='Active');
DECLARE @CardMethod tinyint   = (SELECT TOP 1 PaymentMethodID FROM ref.PaymentMethod
WHERE MethodName='Card');
DECLARE @InvOpen tinyint      = (SELECT TOP 1 InvoiceStatusID FROM ref.InvoiceStatus
WHERE StatusName='Open');
DECLARE @CatMovies smallint   = (SELECT TOP 1 ChannelCategoryID FROM ref.ChannelCategory
WHERE CategoryName='Movies');
DECLARE @GenreDrama smallint  = (SELECT TOP 1 GenreID FROM ref.Genre WHERE
GenreName='Drama');

IF NOT EXISTS (SELECT 1 FROM dbo.Subscriber WHERE ExternalContractNo='TST-000001')
BEGIN
    INSERT INTO dbo.Subscriber
    (ExternalContractNo, FirstName, LastName, Phone, Email, BirthDate, City, Street,
HouseNo, ApartmentNo, CreatedAt, ClosedAt, SubscriberStatusID)
    VALUES
    ('TST-000001', N'Іван', N'Петренко', N'+380631112233', N'ivan.test@mail.com', '1999-
05-20',
     N'Львів', N'Шевченка', N'10', N'15', '2023-02-10', NULL, @ActiveStatus);
END

DECLARE @SubscriberID int = (SELECT SubscriberID FROM dbo.Subscriber WHERE
ExternalContractNo='TST-000001');

IF NOT EXISTS (SELECT 1 FROM dbo.ChannelGroup WHERE GroupName=N'Базовий пакет (TEST)')
BEGIN
    INSERT INTO dbo.ChannelGroup(GroupName, MonthlyFee, IsActive)
    VALUES (N'Базовий пакет (TEST)', 199.00, 1);
END
DECLARE @GroupID int = (SELECT ChannelGroupID FROM dbo.ChannelGroup WHERE
GroupName=N'Базовий пакет (TEST)');

IF NOT EXISTS (SELECT 1 FROM dbo.Channel WHERE ChannelName=N'КіноПлюс (TEST)')
BEGIN
    INSERT INTO dbo.Channel(ChannelName, ChannelCategoryID, IsHD, IsActive)
    VALUES (N'КіноПлюс (TEST)', @CatMovies, 1, 1);
END
DECLARE @ChannelID int = (SELECT ChannelID FROM dbo.Channel WHERE ChannelName=N'КіноПлюс
(TEST)');

IF NOT EXISTS (SELECT 1 FROM dbo.ChannelGroupChannel WHERE ChannelGroupID=@GroupID AND
ChannelID=@ChannelID)
BEGIN
    INSERT INTO dbo.ChannelGroupChannel(ChannelGroupID, ChannelID)
    VALUES (@GroupID, @ChannelID);
END

IF NOT EXISTS (
    SELECT 1 FROM dbo.SubscriberChannelGroup
    WHERE SubscriberID=@SubscriberID AND ChannelGroupID=@GroupID AND StartDate='2024-01-
01'
```

```
)
BEGIN
    INSERT INTO dbo.SubscriberChannelGroup(SubscriberID, ChannelGroupID, StartDate,
EndDate)
    VALUES (@SubscriberID, @GroupID, '2024-01-01', NULL);
END

-- C4: Movie + genre
IF NOT EXISTS (SELECT 1 FROM dbo.Movie WHERE MovieTitle=N'Test Movie (2024)')
BEGIN
    INSERT INTO dbo.Movie(MovieTitle, ReleaseYear, DurationMin, BasePrice, AgeRating)
    VALUES (N'Test Movie (2024)', 2024, 110, 79.00, '16+');
END
DECLARE @MovieID int = (SELECT MovieID FROM dbo.Movie WHERE MovieTitle=N'Test Movie
(2024)');

IF NOT EXISTS (SELECT 1 FROM dbo.MovieGenre WHERE MovieID=@MovieID AND
GenreID=@GenreDrama)
BEGIN
    INSERT INTO dbo.MovieGenre(MovieID, GenreID) VALUES (@MovieID, @GenreDrama);
E
DECLARE @OrderID bigint;
IF NOT EXISTS (
    SELECT 1 FROM dbo.MovieOrder
    WHERE SubscriberID=@SubscriberID AND MovieID=@MovieID AND OrderDateTime='2024-01-10
20:15:00'
)
BEGIN
    INSERT INTO dbo.MovieOrder(SubscriberID, MovieID, OrderDateTime, PricePaid)
    VALUES (@SubscriberID, @MovieID, '2024-01-10 20:15:00', 79.00);
END
SELECT @OrderID = MovieOrderID
FROM dbo.MovieOrder
WHERE SubscriberID=@SubscriberID AND MovieID=@MovieID AND OrderDateTime='2024-01-10
20:15:00';

DECLARE @InvoiceID bigint;
IF NOT EXISTS (
    SELECT 1 FROM dbo.Invoice WHERE SubscriberID=@SubscriberID AND InvoiceMonth='2024-01-
01'
)
BEGIN
    INSERT INTO dbo.Invoice(SubscriberID, InvoiceMonth, IssuedAt, DueDate,
InvoiceStatusID, TotalAmount)
    VALUES (@SubscriberID, '2024-01-01', '2024-01-02', '2024-01-15', @InvOpen, 0);
END
SELECT @InvoiceID = InvoiceID
FROM dbo.Invoice
WHERE SubscriberID=@SubscriberID AND InvoiceMonth='2024-01-01';

IF NOT EXISTS (SELECT 1 FROM dbo.InvoiceLine WHERE InvoiceID=@InvoiceID AND
LineType='SubscriptionFee')
BEGIN
    INSERT INTO dbo.InvoiceLine(InvoiceID, LineType, SourceID, Description, Amount)
    VALUES (@InvoiceID, 'SubscriptionFee', NULL, N'Абонплата: Базовий пакет (TEST)',
199.00);
END

IF NOT EXISTS (SELECT 1 FROM dbo.InvoiceLine WHERE InvoiceID=@InvoiceID AND
LineType='Movie' AND SourceID=@OrderID)
BEGIN
    INSERT INTO dbo.InvoiceLine(InvoiceID, LineType, SourceID, Description, Amount)
    VALUES (@InvoiceID, 'Movie', @OrderID, N'VOD: Test Movie (2024)', 79.00);
END
```

```
UPDATE i
SET TotalAmount = x.SumAmount
FROM dbo.Invoice i
CROSS APPLY (
    SELECT SUM(il.Amount) AS SumAmount
    FROM dbo.InvoiceLine il
    WHERE il.InvoiceID = i.InvoiceID
) x
WHERE i.InvoiceID = @InvoiceID;

IF NOT EXISTS (
    SELECT 1 FROM dbo.Payment WHERE InvoiceID=@InvoiceID AND PaymentDateTime='2024-01-05
12:00:00'
)
BEGIN
    INSERT INTO dbo.Payment(InvoiceID, PaymentDateTime, Amount, PaymentMethodID)
    VALUES (@InvoiceID, '2024-01-05 12:00:00', 150.00, @CardMethod);
END

SELECT TOP 20
  s.SubscriberID, s.ExternalContractNo, s.LastName, s.FirstName,
  scg.StartDate, scg.EndDate, cg.GroupName, cg.MonthlyFee
FROM dbo.Subscriber s
JOIN dbo.SubscriberChannelGroup scg ON scg.SubscriberID = s.SubscriberID
JOIN dbo.ChannelGroup cg ON cg.ChannelGroupID = scg.ChannelGroupID
WHERE s.ExternalContractNo='TST-000001'
ORDER BY scg.StartDate DESC;

SELECT TOP 20
  mo.MovieOrderID, mo.OrderDateTime, mo.PricePaid,
  m.MovieTitle, m.ReleaseYear
FROM dbo.MovieOrder mo
JOIN dbo.Movie m ON m.MovieID = mo.MovieID
WHERE mo.SubscriberID = @SubscriberID
ORDER BY mo.OrderDateTime DESC;

SELECT
  i.InvoiceID, i.InvoiceMonth, i.TotalAmount AS Charged,
  ISNULL(p.Paid,0) AS Paid,
  i.TotalAmount - ISNULL(p.Paid,0) AS Balance
FROM dbo.Invoice i
OUTER APPLY (
    SELECT SUM(Amount) AS Paid
    FROM dbo.Payment
    WHERE InvoiceID = i.InvoiceID
) p
WHERE i.InvoiceID = @InvoiceID;

SELECT
  il.InvoiceLineID, il.LineType, il.SourceID, il.Description, il.Amount
FROM dbo.InvoiceLine il
WHERE il.InvoiceID = @InvoiceID
ORDER BY il.InvoiceLineID;
GO
```

Тригери

```
USE CableTV;
GO

CREATE OR ALTER TRIGGER dbo.trg_SubCG_NoOverlaps
ON dbo.SubscriberChannelGroup
AFTER INSERT, UPDATE
AS
```

```
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1
        FROM dbo.SubscriberChannelGroup a
        JOIN dbo.SubscriberChannelGroup b
          ON a.SubscriberID = b.SubscriberID
         AND a.ChannelGroupID = b.ChannelGroupID
         AND a.SubscriberChannelGroupID <> b.SubscriberChannelGroupID
        JOIN inserted i
          ON i.SubscriberID = a.SubscriberID
         AND i.ChannelGroupID = a.ChannelGroupID
        WHERE
          a.StartDate <= ISNULL(b.EndDate, '9999-12-31')
          AND b.StartDate <= ISNULL(a.EndDate, '9999-12-31')
    )
    BEGIN
        RAISERROR(N'Error. You cannot have two subscriptions to the same package with
overlapping dates for one subscriber.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
END
GO

CREATE OR ALTER TRIGGER dbo.trg_Invoice_RecalcTotal
ON dbo.InvoiceLine
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON;

    ;WITH Changed AS (
        SELECT InvoiceID FROM inserted
        UNION
        SELECT InvoiceID FROM deleted
    )
    UPDATE i
    SET TotalAmount = ISNULL(x.SumAmount, 0)
    FROM dbo.Invoice i
    JOIN Changed c ON c.InvoiceID = i.InvoiceID
    OUTER APPLY (
        SELECT SUM(il.Amount) AS SumAmount
        FROM dbo.InvoiceLine il
        WHERE il.InvoiceID = i.InvoiceID
    ) x;
END
GO
```

Створення та налаштування таблиць для сховища даних CableTV_DWH

```
CREATE DATABASE CableTV_DWH;
GO
USE CableTV_DWH;
GO


USE CableTV_DWH;
GO

DROP TABLE IF EXISTS dbo.FactMovieSales;
DROP TABLE IF EXISTS dbo.FactFinancials;
DROP TABLE IF EXISTS dbo.FactMovieOperations;
DROP TABLE IF EXISTS dbo.FactSubscriptions;
```

```
DROP TABLE IF EXISTS dbo.DimSubscriber;
DROP TABLE IF EXISTS dbo.DimMovie;
DROP TABLE IF EXISTS dbo.DimChannelGroup;
DROP TABLE IF EXISTS dbo.DimPaymentMethod;
DROP TABLE IF EXISTS dbo.DimDate;
GO

CREATE TABLE dbo.DimDate (
    DateKey INT PRIMARY KEY,
    FullDate DATE NOT NULL,
    Year INT NOT NULL,
    Month INT NOT NULL,
    MonthName NVARCHAR(20) NOT NULL,
    Quarter INT NOT NULL,
    DayOfWeek INT NOT NULL,
    DayName NVARCHAR(20) NOT NULL,
    IsWeekend BIT NOT NULL
);

-- 2. DimSubscriber
CREATE TABLE dbo.DimSubscriber (
    SubscriberKey INT IDENTITY(1,1) PRIMARY KEY,
    SubscriberID_Source INT NOT NULL,
    FullName NVARCHAR(150) NOT NULL,
    City NVARCHAR(60) NOT NULL,
    AgeGroup NVARCHAR(20) NULL,
    CurrentStatus NVARCHAR(30) NOT NULL
);

CREATE TABLE dbo.DimMovie (
    MovieKey INT IDENTITY(1,1) PRIMARY KEY,
    MovieID_Source INT NOT NULL,
    MovieTitle NVARCHAR(200) NOT NULL,
    ReleaseYear INT NOT NULL,
    GenreName NVARCHAR(50) NOT NULL,
    AgeRating NVARCHAR(10) NULL
);

CREATE TABLE dbo.DimChannelGroup (
    ChannelGroupKey INT IDENTITY(1,1) PRIMARY KEY,
    ChannelGroupID_Source INT NOT NULL,
    GroupName NVARCHAR(100) NOT NULL,
    MonthlyFee DECIMAL(10,2) NOT NULL
);

CREATE TABLE dbo.DimPaymentMethod (
    PaymentMethodKey INT IDENTITY(1,1) PRIMARY KEY,
    PaymentMethodID_Source TINYINT NOT NULL,
    MethodName NVARCHAR(30) NOT NULL
);

CREATE TABLE dbo.FactMovieOperations (
    FactKey BIGINT IDENTITY(1,1) PRIMARY KEY,
    DateKey INT NOT NULL CONSTRAINT FK_FactMovie_Date REFERENCES dbo.DimDate(DateKey),
    SubscriberKey INT NOT NULL CONSTRAINT FK_FactMovie_Sub REFERENCES
dbo.DimSubscriber(SubscriberKey),
    MovieKey INT NOT NULL CONSTRAINT FK_FactMovie_Movie REFERENCES
dbo.DimMovie(MovieKey),
    PricePaid DECIMAL(10,2) NOT NULL,
    OrderCount INT DEFAULT 1
);

CREATE TABLE dbo.FactFinancials (
```

```
    FactKey BIGINT IDENTITY(1,1) PRIMARY KEY,
    DateKey INT NOT NULL CONSTRAINT FK_FactFin_Date REFERENCES dbo.DimDate(DateKey),
    SubscriberKey INT NOT NULL CONSTRAINT FK_FactFin_Sub REFERENCES
dbo.DimSubscriber(SubscriberKey),
    PaymentMethodKey INT NOT NULL CONSTRAINT FK_FactFin_Method REFERENCES
dbo.DimPaymentMethod(PaymentMethodKey),
    Amount DECIMAL(12,2) NOT NULL,
    TransactionCount INT DEFAULT 1
);
GO
```

Налаштування DimDate

```
DECLARE @StartDate DATE = '2020-01-01';
DECLARE @EndDate DATE = '2030-12-31';

WHILE @StartDate <= @EndDate
BEGIN
    INSERT INTO dbo.DimDate (
        DateKey,
        FullDate,
        Year,
        Month,
        MonthName,
        Quarter,
        DayOfWeek,
        DayName,
        IsWeekend
    )
    SELECT
        CAST(CONVERT(VARCHAR(8), @StartDate, 112) AS INT), -- DateKey (20200101)
        @StartDate,
        YEAR(@StartDate),
        MONTH(@StartDate),
        DATENAME(MONTH, @StartDate),
        DATEPART(QUARTER, @StartDate),
        DATEPART(WEEKDAY, @StartDate),
        DATENAME(WEEKDAY, @StartDate),
        CASE WHEN DATEPART(WEEKDAY, @StartDate) IN (1, 7) THEN 1 ELSE 0 END -- 1=Sun,
7=Sat (зависит от настроек сервера, но обычно так)

    SET @StartDate = DATEADD(DAY, 1, @StartDate);
END
GO
```