Національний університет «Львівська політехніка»

Кафедра програмного забезпечення

**КУРСОВА РОБОТА**

**з дисципліни «Бази даних»**

на тему:

«Система бізнес-аналітики для мережі автосалонів»

Виконав: студент

спеціальності 121

«Інженерія програмного забезпечення»

групи П3-35

Іншин В. В.

Керівник:

ст. викл. кафедри програмного забезпечення,

Цимбалюк Т.М.

Оцінка:

Національна шкала\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів\_\_\_\_\_\_\_Оцінка ECTS\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_ Цимбалюк Т.М.

(підпис)

\_\_\_\_\_\_\_\_\_\_\_\_ Білоіваненко М.В.

(підпис)

Львів – 2024 рік

**ЗМІСТ**

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ РЕАЛІЗАЦІЇ БІЗНЕС-АНАЛІТИКИ ДЛЯ МЕРЕЖІ АВТОСАЛОНІВ

* 1. Опис предметної області мережі автосалонів
  2. Збір вимог до сховища даних системи для мережі автосалонів
  3. Концептуальне моделювання предметної області
  4. Схема корпоративної OLTP системи
  5. Вимоги до архівних даних для OLTP системи
  6. Вимоги до ETL
  7. Постановка завдання

РОЗДІЛ 2. ПРОЕКТУВАННЯ СХОВИЩА ДАНИХ ТА ІМПЛЕМЕНТАЦІЯ ETL ПАКЕТІВ

1. Багатовимірне моделювання предметної області
2. Проектування процесу ETL
3. Імплементація первинного завантаження даних
4. Імплементація інкрементального завантаження даних

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ БІЗНЕС-АНАЛІТИКИ ДЛЯ МЕРЕЖІ АВТОСАЛОНІВ

1. Реалізація архітектури ПЗ та доступу до даних
2. Реалізація функціональних характеристик системи
3. Опис роботи системи

ВИСНОВКИ

СПИСОК ЛІТЕРАТУРИ

ДОДАТКИ

Додаток А. Скрипт створення бази даних

Додаток Б. Скрипт завантаження історичних даних у базу даних

Додаток В. Скрипт створення сховища даних

Додаток Г. Скрипт заповнення БД метаданих

Додаток Д. Скрипт первинної заливки даних у сховище даних

Додаток Е. Скрипт інкрементальної заливки даних у сховище даних

**РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ РЕАЛІЗАЦІЇ БІЗНЕС-АНАЛІТИКИ ДЛЯ МЕРЕЖІ АВТОСАЛОНІВ**

* 1. **Опис предметної області мережі автосалонів**

Компанії, які спеціалізуються на продажу автомобілів, пропонують широкий спектр транспортних засобів для різних категорій покупців, від економічних міських авто до високопродуктивних спортивних машин. Вони надають різноманітні послуги, включаючи продаж нових та б/у автомобілів, фінансування, страхування та післяпродажне обслуговування, що робить їх важливими учасниками автомобільної індустрії. Основні характеристики таких компаній включають індивідуальний підхід до кожного клієнта, з можливістю персоналізації автомобілів за індивідуальними вимогами, та пропозиції тест-драйвів, які дозволяють покупцям випробувати автомобілі перед купівлею. Команди автосалонів складаються з досвідчених професіоналів, які не тільки знають всі технічні деталі представлених автомобілів, але й можуть допомогти в розумінні усіх аспектів автомобільної власності. Гнучкі фінансові плани та лізингові програми роблять придбання більш доступним.

Типова мережа автосалонів має кілька виставкових центрів, розташованих у різних містах чи навіть країнах, що дозволяє охопити більшу кількість потенційних клієнтів. Кожен автосалон містить виставкову залу, де можуть бути одночасно представлені різні моделі автомобілів, і організовують комплексні маркетингові кампанії та промоційні заходи для залучення нових клієнтів та збільшення продажів. Автосалони також розробляють детальні плани послуг, які включають обслуговування, ремонт, оновлення комплектуючих та інші послуги, щоб забезпечити повний цикл обслуговування автомобілів.

Крім традиційних методів фінансування, автосалони все частіше пропонують програми лізингу, що забезпечують клієнтам можливість користуватися новітніми автомобілями без необхідності їх повного придбання. Лізингові плани можуть включати гнучкі терміни платежів, обмежену відповідальність за знос та опції купівлі автомобіля в кінці терміну лізингу. Ці програми особливо привабливі для бізнес-клієнтів та осіб, які прагнуть змінювати автомобілі кожні кілька років.

Онлайн-платформи для автосалонів грають важливу роль у сучасному автомобільному ринку, дозволяючи покупцям переглядати, конфігурувати та навіть замовляти автомобілі через Інтернет[1]. Важливість таких платформ полягає в їхній доступності 24/7, можливості легко порівнювати різні моделі[2] та комплектації[3], а також в здатності приваблювати більше клієнтів через цифрові канали. За допомогою онлайн-автосалонів, компанії можуть значно розширити свій географічний охоплення та покращити клієнтський досвід, надаючи детальну інформацію про продукцію і актуальні пропозиції в реальному часі.

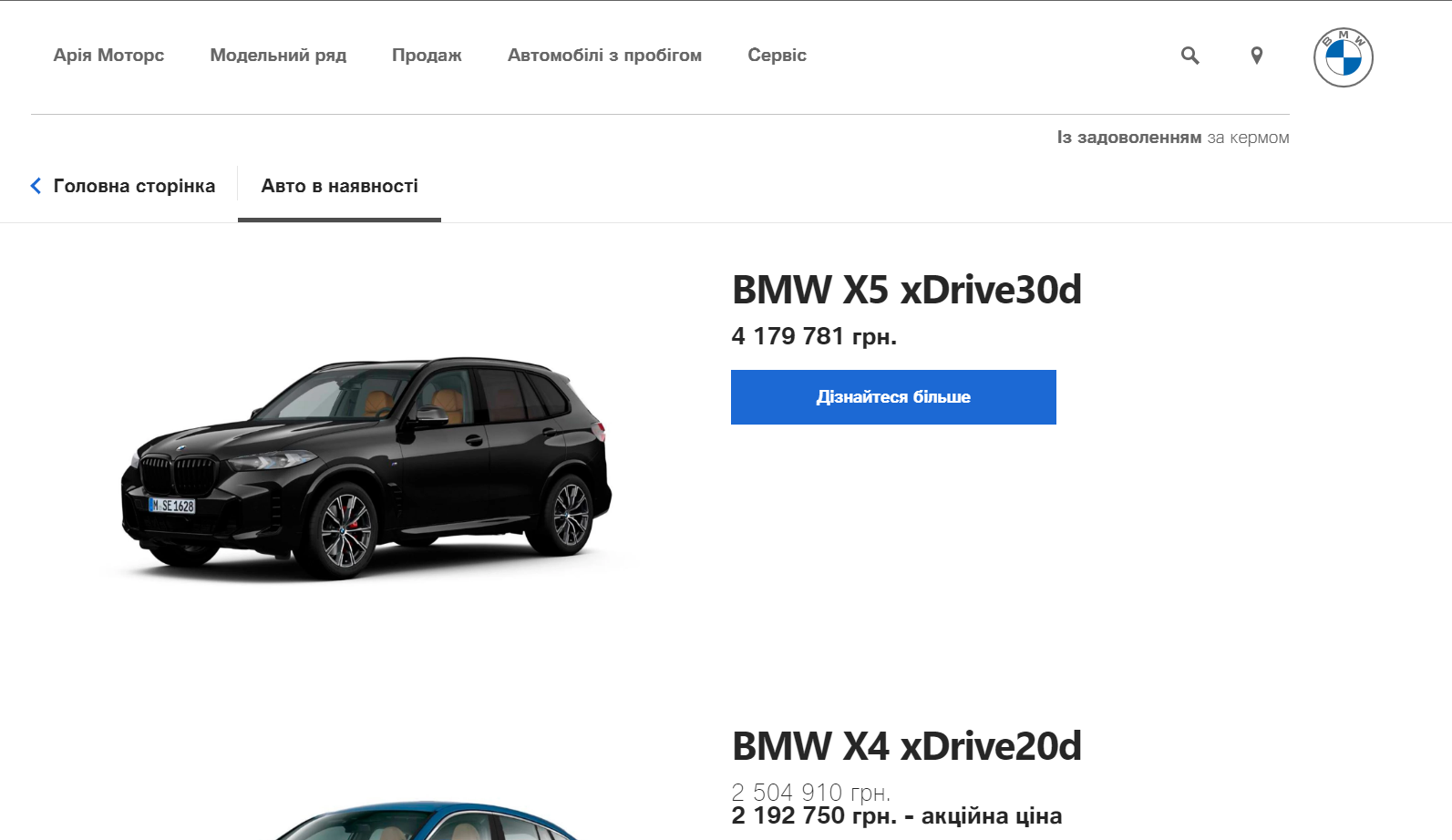


Рис. 1.1. Інформація про наявні в автосалоні авто у офіційного дилера у м.Львові

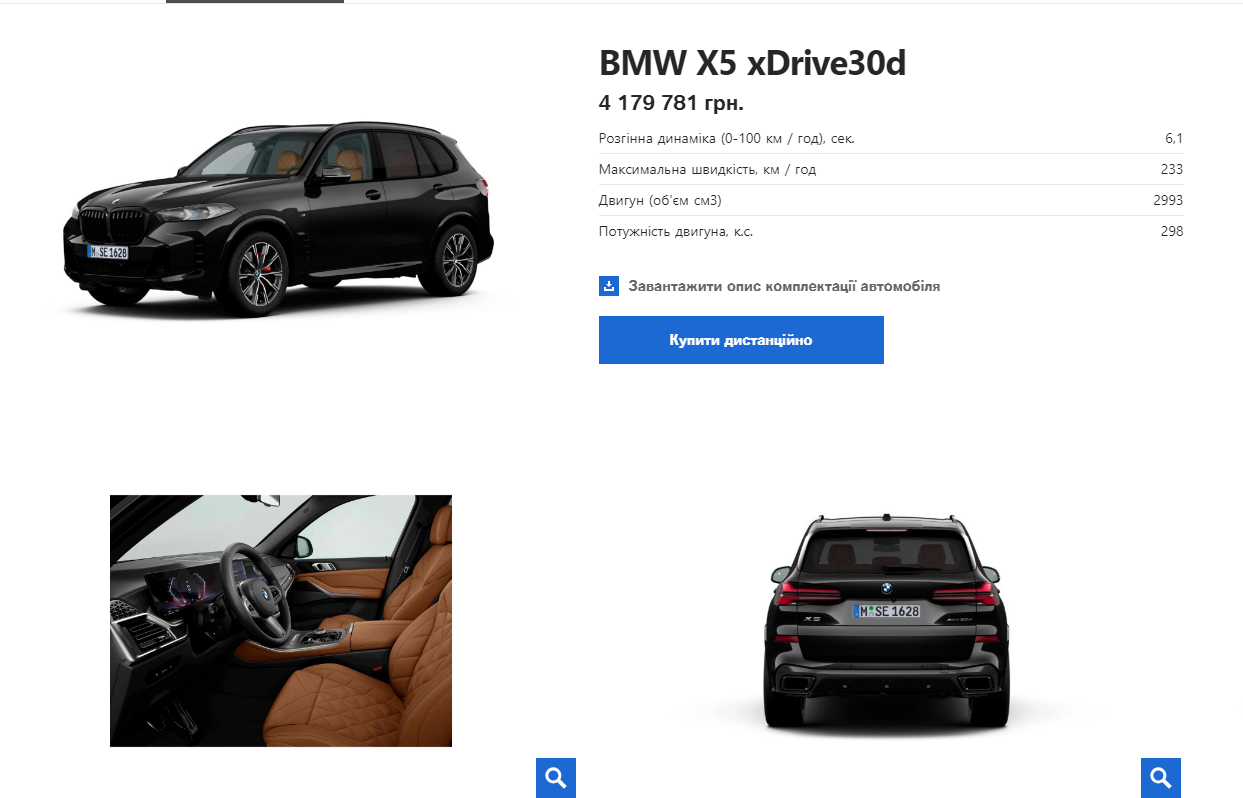


Рис. 1.2. Інформація про авто із фото-демонстрацією зовнішнього вигляду



Рис. 1.3. Детальна інформація про технічні характеристики авто

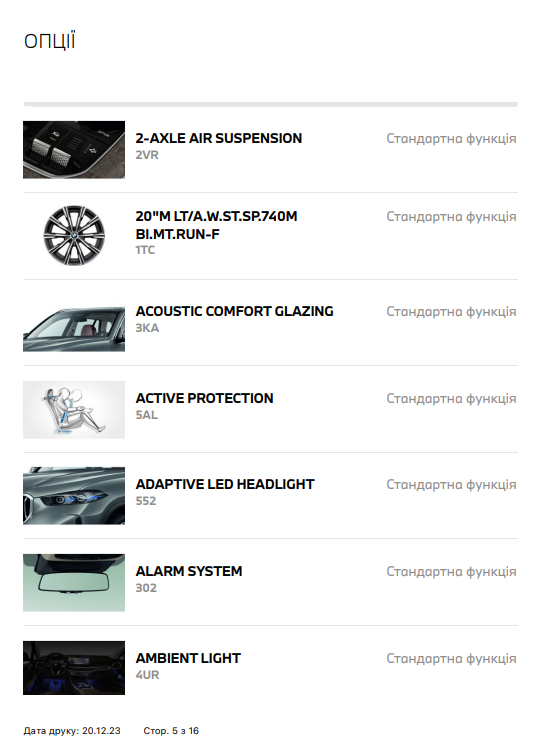


Рис. 1.4. Інформація про опції авто (безпека, комфорт, мультимедіа)

Після перегляду інформації про авто[4], користувач може заповнити контакту інформацію[6] для подальшого встановлення зв’язку із менеджером, і, можливо, купівлі транспортного засобу.

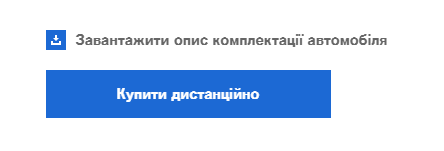


Рис. 1.5. Перехід нас форму оформлення запиту

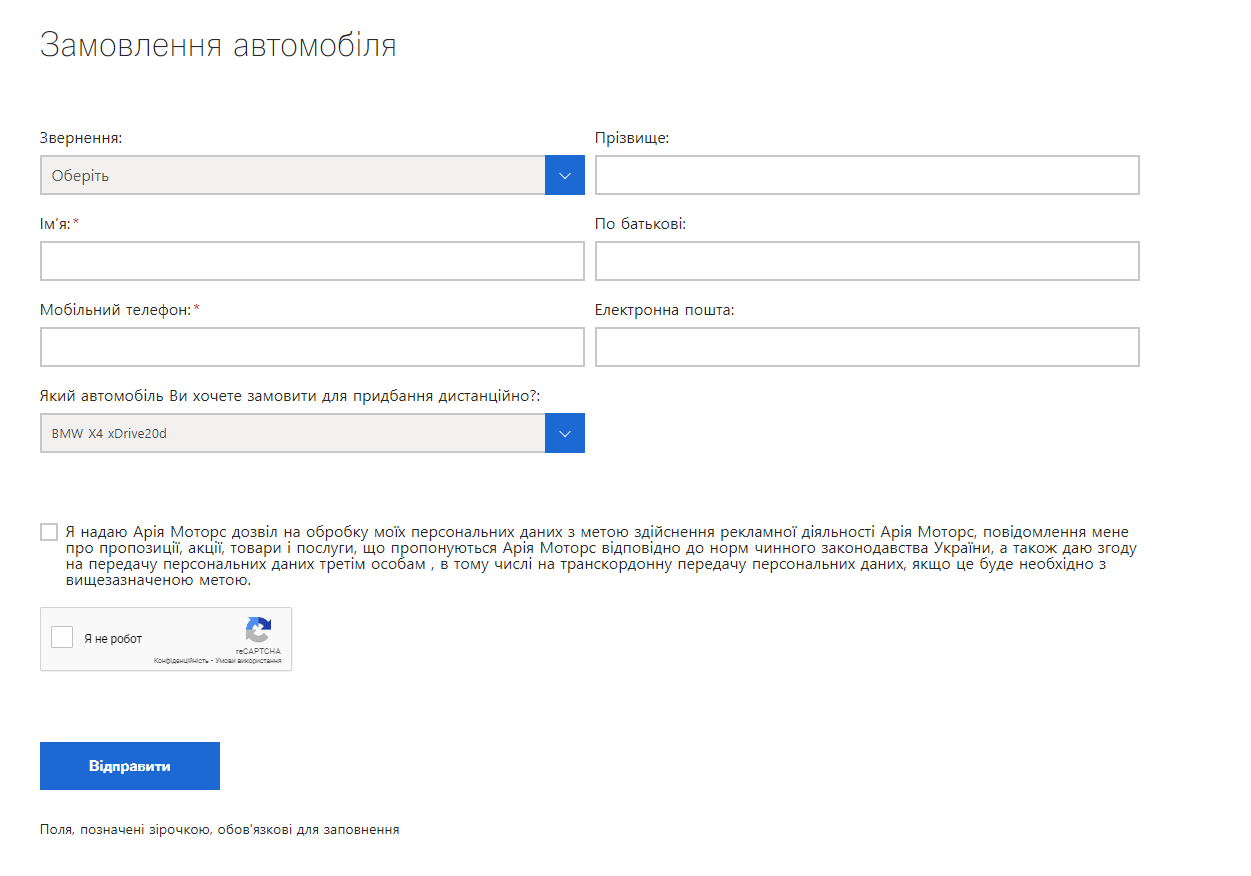


Рис. 1.6. Форма для замовлення автомобіля

Далі відбувається процес оформлення замовлення і уточнення даних клієнта. Це може відбуватися онлайн за допомогою листування, по телефону або в очному режимі у дилера даної марки у місті клієнта[7].

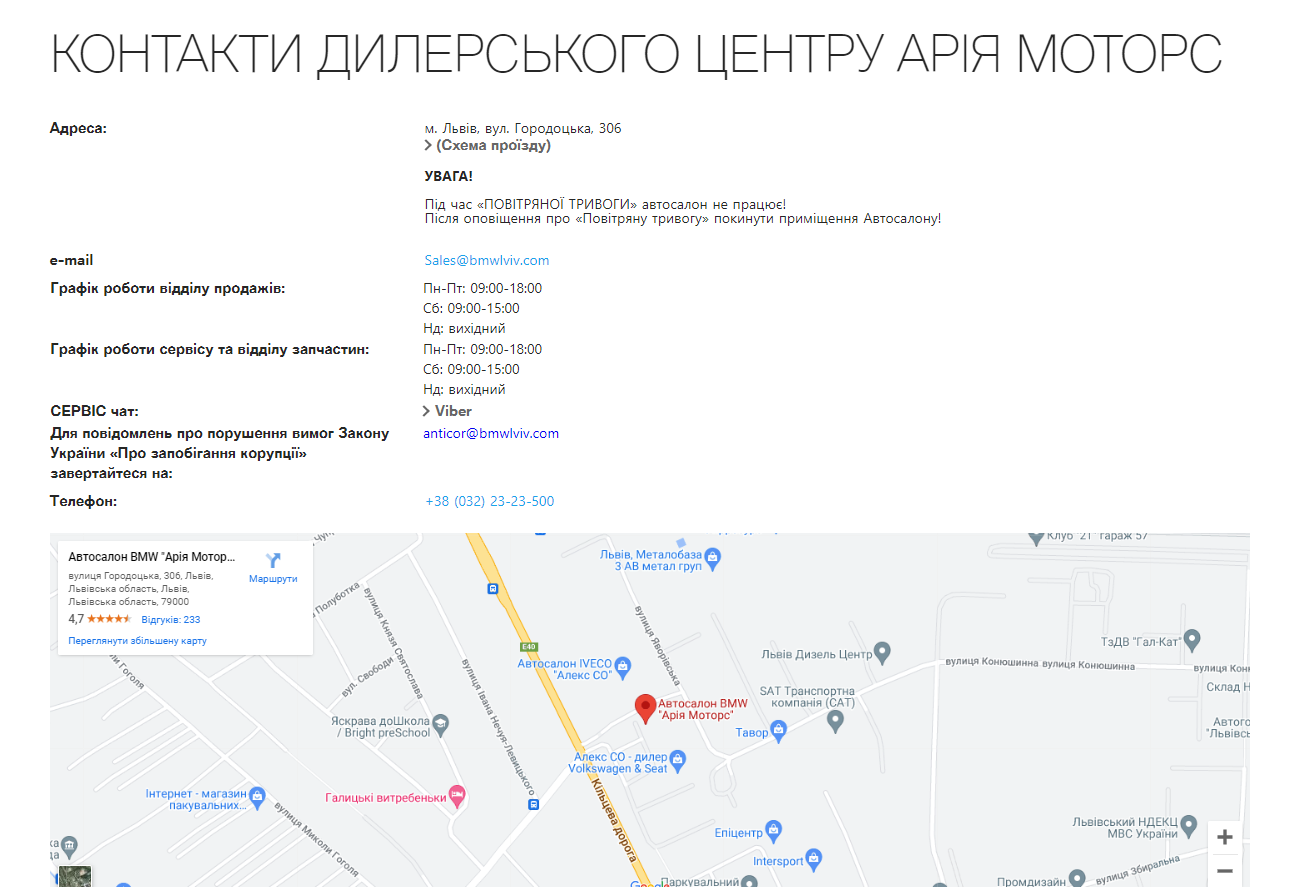


Рис. 1.7. Інформація про геолокацію дилера у місті клієнта

Серед основних бізнес-напрямів компанії можна виділити:

* Продаж авто як основний напрям розвитку компанії.
* Надання сервісу авто конкретної марки або також і інших марок.
* Надання послуг лізингу авто для клієнтів, які обирають спосіб тимчасового використання транспортного засобу.
* Найм кращих співробітників для покращення кількості продажів.
* Пошук впливових та багатших клієнтів для розвитку прибутку компанії.

Виділеними бізнес-процесами для системи є організація надання послуг лізингу авто та безпосередньо продаж транспортних засобів онлайн або у автосалоні.

Продаж авто – клієнт укладає контракт про купівлю авто, з повною оплатою транспортного засобу одноразово, з подальшою доставкою до клієнта. Типовими бізнес-правилами є:

* Якщо авто немає в наявності в автосалоні, то формується акт доставки авто від його дистрибютора до автосалону.
* Оплата здійснюється єдиною транзакцією методами, що є в переліку. Далі йде її опрацювання і в разі успішного надходження коштів стан оплати фіксується системою компанії.

Лізинг авто – клієнт укладає акт лізингу, обравши один із можливих пакетів лізингу. Типовими бізнес-правилами є:

* Виплата зарплати здійснюється єдиною транзакцією на початку кожного місяця. Метод виплати обирається компанією.
* Обрахунок ціни лізингу залежить від тарифу і терміну, на який укладається лізинг.
* У разі виникнення якихось непередбачених випадків, страхова компанія клієнта оплачує усі витрати. У випадку, якщо клієнт обрав тариф лізингу без страхування – усі витрати і відшкодування виплачує сам клієнт.
  1. **Збір вимог до сховища даних системи для мережі автосалонів**
     1. **Концептуальне моделювання предметної області**

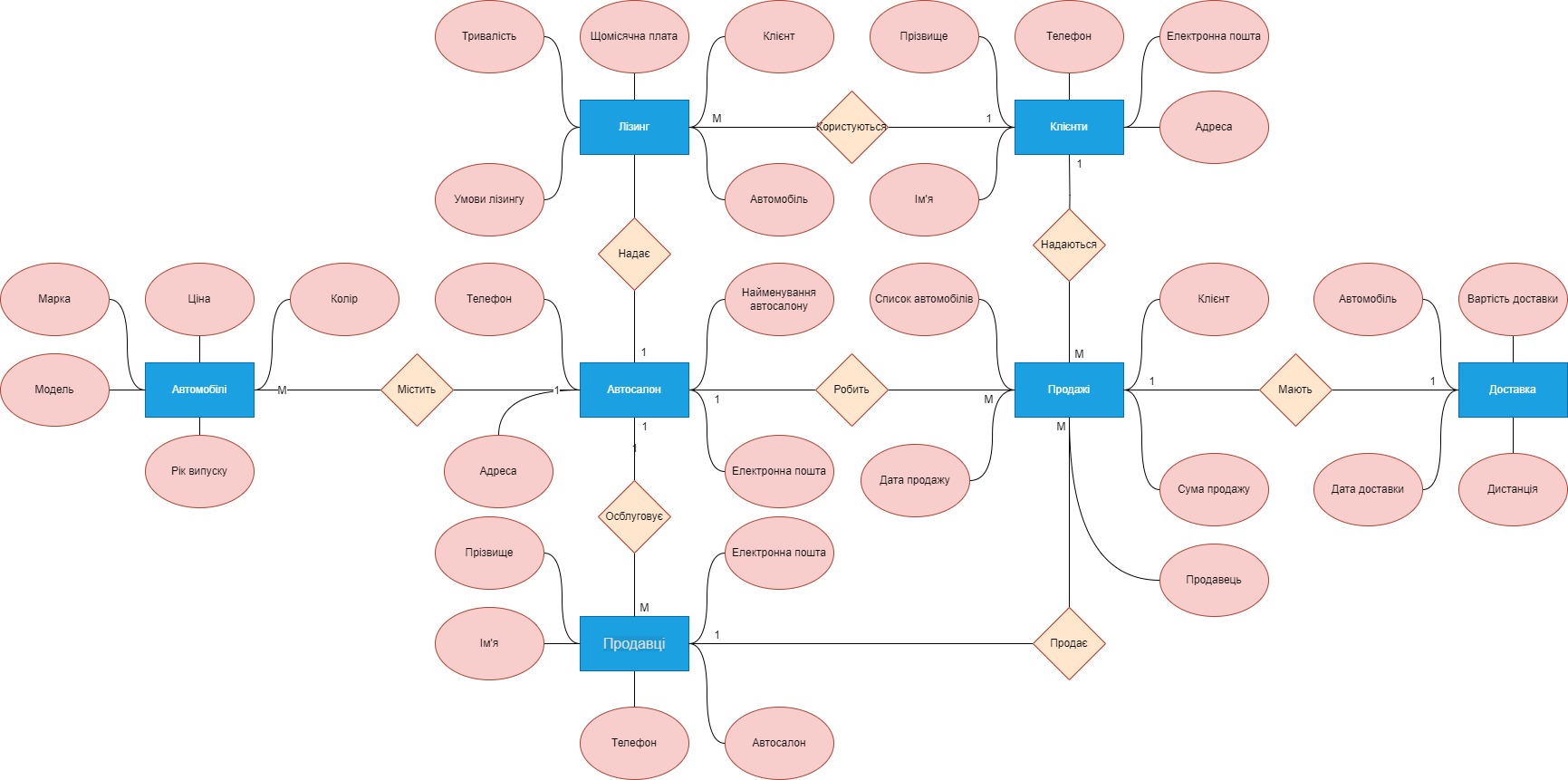
****

Рис. 1.8. Концептуальна діаграма предметної області

Основними сутностями діаграми є:

* Клієнти: Люди, які орендують автомобілі.
* Автомобілі: Автомобілі, які орендують клієнти.
* Автосалони: Місця, де клієнти орендують автомобілі.
* Продавці: Люди, які працюють в автосалонах і продають автомобілі клієнтам.
* Лізінг: Угоди, за якими клієнти орендують автомобілі в автосалонах.
* Продажі: Угоди, за якими клієнти купують автомобілі в автосалонах.
* Доставка: Сервіс доставки автомобіля після придбання.

Зв'язки:

* Клієнт орендує Автомобіль: Один клієнт може орендувати один або декілька автомобілів. Один автомобіль може бути орендований одним або декількома клієнтами.
* Автомобіль належить Автосалону: Один автомобіль належить одному автосалону. Один автосалон може мати один або декілька автомобілів.
* Продавець працює в Автосалоні: Один продавець може працювати в одному автосалоні. Один автосалон може мати одного або декілька продавців.
* Оренда відбувається в Автосалоні: Одна оренда відбувається в одному автосалоні. Один автосалон може мати одну або декілька оренд.
  + 1. **Схема корпоративної OLTP схеми**

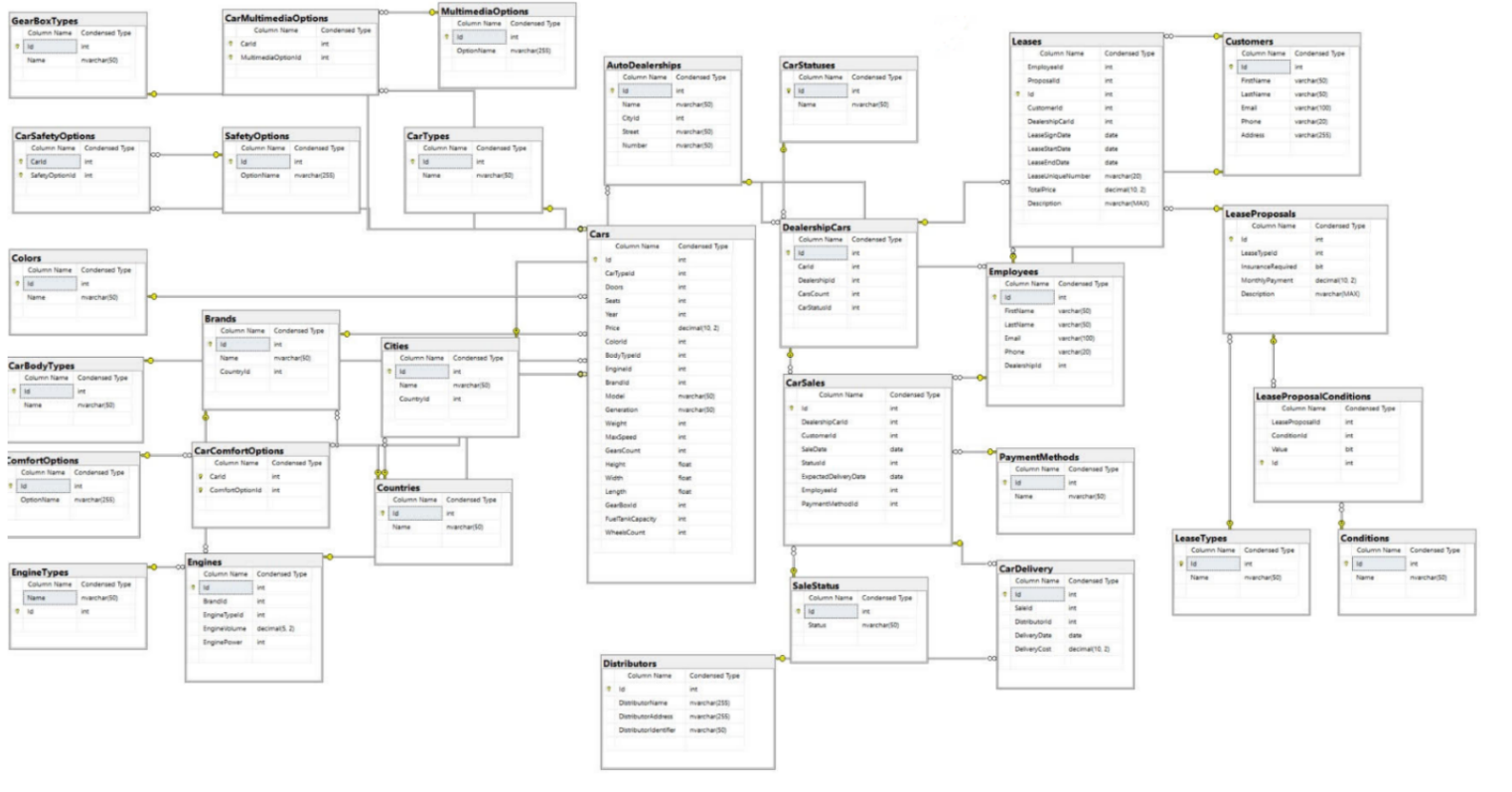
****

Рис. 1.9. Схема OLTP БД

*Опис таблиць OLTP БД:*

1. Об’єктні:

* Autodealerships - зберігає інформацію про автосалони, включаючи їх назву, місцезнаходження та дати створення та оновлення записів.
  + Id: Унікальний ідентифікатор автосалону.
  + Name: Назва автосалону.
  + CityId: Ідентифікатор міста, де розташований автосалон.
  + Street: Назва вулиці, на якій розташований автосалон.
  + Number: Номер будинку, за яким розташований автосалон.
  + CreateDate: Дата створення запису про автосалон.
  + UpdateDate: Дата останнього оновлення запису про автосалон.
* Cars - містить деталі про автомобілі, включно з характеристиками та ідентифікаторами типів і брендів, датами створення та оновлення.
  + Id: Унікальний ідентифікатор автомобіля.
  + CarTypeId: Унікальний ідентифікатор типу автомобіля.
  + Doors: Кількість дверей у автомобіля.
  + Seats: Кількість сидінь у автомобіля.
  + Year: Рік випуску автомобіля.
  + Price: Ціна автомобіля.
  + ColorId: Унікальний ідентифікатор кольору автомобіля.
  + BodyTypeId: Унікальний ідентифікатор типу кузова автомобіля.
  + EngineId: Унікальний ідентифікатор двигуна автомобіля.
  + BrandId: Унікальний ідентифікатор бренду автомобіля.
  + Model: Модель автомобіля.
  + Generation: Покоління автомобіля.
  + Weight: Вага автомобіля.
  + MaxSpeed: Максимальна швидкість автомобіля.
  + GearsCount: Кількість передач автомобіля.
  + Height: Висота автомобіля.
  + Width: Ширина автомобіля.
  + Length: Довжина автомобіля.
  + GearBoxId: Унікальний ідентифікатор типу коробки передач автомобіля.
  + FuelTankCapacity: Об'єм паливного баку автомобіля.
  + WheelsCount: Кількість коліс автомобіля.
  + CreateDate: Дата створення запису про автомобіль.
  + UpdateDate: Дата останнього оновлення запису про автомобіль.
* Customers - веде облік інформації про клієнтів, включаючи контактні дані та адресу.
  + Id: Унікальний ідентифікатор клієнта.
  + FirstName: Ім'я клієнта.
  + LastName: Прізвище клієнта.
  + Email: Електронна адреса клієнта.
  + Phone: Номер телефону клієнта.
  + Address: Адреса клієнта.
  + CreateDate: Дата створення запису про клієнта.
  + UpdateDate: Дата останнього оновлення запису про клієнта.
* Distributors - записує інформацію про дистриб'юторів, включаючи назву, адресу та унікальні ідентифікатори.
  + Id: Унікальний ідентифікатор запису про дистриб'ютора.
  + DistributorName: Назва дистриб'ютора.
  + DistributorAddress: Адреса дистриб'ютоАра.
  + DistributorIdentifier: Ідентифікатор дистриб'ютора (може бути різними формами ідентифікації, такими як номера, коди тощо).
  + CreateDate: Дата створення запису про дистриб'ютора.
  + UpdateDate: Дата останнього оновлення запису про дистриб'ютора.
* Employees - список співробітників автосалонів з деталями контактної інформації та ідентифікатором дилерського центру.
  + Id: Унікальний ідентифікатор запису про співробітника.
  + FirstName: Ім'я співробітника.
  + LastName: Прізвище співробітника.
  + Email: Електронна адреса співробітника.
  + Phone: Телефонний номер співробітника.
  + DealershipId: Ідентифікатор автодилерського центру, до якого належить співробітник.
  + CreateDate: Дата створення запису про співробітника.
  + UpdateDate: Дата оновлення запису про співробітника.
* Engines - інформація про двигуни, включаючи тип, об'єм, потужність та дати створення та оновлення.
  + Id: Унікальний ідентифікатор запису про двигун.
  + BrandId: Ідентифікатор бренду або виробника двигуна.
  + EngineTypeId: Ідентифікатор типу двигуна (наприклад, бензиновий, дизельний тощо).
  + EngineVolume: Об'єм двигуна у літрах.
  + EnginePower: Потужність двигуна в кінських силах (к.с.) або кіловатах (кВ).
  + CreateDate: Дата створення запису про двигун.
  + UpdateDate: Дата оновлення запису про двигун.
* LeaseProposals - пропозиції оренди з деталями, такими як тип оренди, обов'язковість страхування, місячні платежі та описи
  + Id: Унікальний ідентифікатор пропозиції щодо оренди.
  + LeaseTypeId: Ідентифікатор типу оренди, пов'язаний з цією пропозицією.
  + InsuranceRequired: Показник, який вказує, чи потрібне страхування для цієї пропозиції щодо оренди.
  + MonthlyPayment: Місячний платіж за оренду автомобіля.
  + Description: Опис пропозиції щодо оренди.
  + CreateDate: Дата створення запису про пропозицію щодо оренди.
  + UpdateDate: Дата останнього оновлення запису про пропозицію щодо оренди.

1. Документні:
   * CarDeliveries - відомості про доставку автомобілів, включаючи ідентифікатори продажів та дистриб'юторів, а також вартість доставки.
     + Id: Унікальний ідентифікатор запису про доставку автомобіля.
     + SaleId: Унікальний ідентифікатор продажу, до якого відноситься дана доставка.
     + DistributorId: Унікальний ідентифікатор дистриб'ютора, який здійснює доставку.
     + DeliveryDate: Дата доставки автомобіля.
     + DeliveryCost: Вартість доставки автомобіля.
     + CreateDate: Дата створення запису про доставку.
     + UpdateDate: Дата останнього оновлення запису про доставку.
   * Leases - договори оренди автомобілів з повними деталями про умови, тривалість та вартість.
     + Id: Унікальний ідентифікатор договору оренди.
     + EmployeeId: Ідентифікатор працівника, який уклав договір оренди автомобіля.
     + ProposalId: Ідентифікатор пропозиції щодо оренди, пов'язаний з цим договором оренди.
     + CustomerId: Ідентифікатор клієнта, який орендує автомобіль.
     + DealershipCarId: Ідентифікатор автомобіля, який орендується.
     + LeaseSignDate: Дата підписання договору оренди.
     + LeaseStartDate: Дата початку терміну дії оренди.
     + LeaseEndDate: Дата закінчення терміну дії оренди.
     + LeaseUniqueNumber: Унікальний номер договору оренди.
     + TotalPrice: Загальна вартість оренди.
     + Description: Опис договору оренди.
     + CreateDate: Дата створення запису про договір оренди.
     + UpdateDate: Дата останнього оновлення запису про договір оренди.
   * CarSales - записи про продажі автомобілів, включаючи ідентифікатори клієнтів, способи оплати та статуси продажів.
     + Id: Унікальний ідентифікатор продажу автомобіля.
     + DealershipCarId: Унікальний ідентифікатор автомобіля на продажу у дилерському центрі.
     + CustomerId: Унікальний ідентифікатор клієнта, який придбав автомобіль.
     + SaleDate: Дата продажу автомобіля.
     + StatusId: Статус продажу (наприклад, "відкритий", "закритий" тощо).
     + ExpectedDeliveryDate: Очікувана дата доставки автомобіля клієнту.
     + EmployeeId: Унікальний ідентифікатор працівника, який обробив цей продаж.
     + PaymentMethodId: Унікальний ідентифікатор способу оплати для цього продажу.
     + CreateDate: Дата створення запису про продаж автомобіля.
     + UpdateDate: Дата останнього оновлення запису про продаж автомобіля.
   * DealershipCars - інформація про автомобілі, наявні в автосалонах, включаючи їх кількість та статус.
     + Id: Унікальний ідентифікатор запису про автомобіль в автосалоні.
     + CarId: Унікальний ідентифікатор автомобіля, який відповідає цьому запису.
     + DealershipId: Унікальний ідентифікатор автосалону, в якому знаходиться цей автомобіль.
     + CarsCount: Кількість автомобілів з однаковим CarId, що знаходяться в цьому автосалоні.
     + CarStatusId: Статус автомобіля в автосалоні (наприклад, "В наявності", "Проданий", "На тест-драйві" і т. д.).
     + CreateDate: Дата створення запису про автомобіль в автосалоні.
     + UpdateDate: Дата останнього оновлення запису про автомобіль в автосалоні.
2. Класифікатори:

* LeaseTypes - зберігає різні типи оренди, які можна використовувати для договорів оренди.
  + Id: Унікальний ідентифікатор типу оренди.
  + Name: Назва типу оренди.
  + CreateDate: Дата створення запису про тип оренди.
  + UpdateDate: Дата останнього оновлення запису про тип оренди.
* GearBoxTypes - визначає типи коробок передач, доступні для автомобілів.
  + Id: Унікальний ідентифікатор запису про тип коробки передач.
  + Name: Назва типу коробки передач.
  + CreateDate: Дата створення запису про тип коробки передач.
  + UpdateDate: Дата останнього оновлення запису про тип коробки передач.
* EngineTypes - класифікує типи двигунів за їхніми технічними характеристиками, як-от бензинові, дизельні та інші.
  + Name: Назва типу двигуна.
  + Id: Унікальний ідентифікатор запису про тип двигуна.
  + CreateDate: Дата створення запису про тип двигуна.
  + UpdateDate: Дата останнього оновлення запису про тип двигуна.
* DealershipCarStatuses – відображає статуси автомобілів в автосалонах, наприклад, чи автомобіль в наявності, проданий, або на тест-драйві.
  + Id: Унікальний ідентифікатор запису про статус автомобіля в автосалоні.
  + Name: Назва статусу автомобіля (наприклад, "В наявності", "Проданий", "На тест-драйві" і т. д.).
  + CreateDate: Дата створення запису про статус автомобіля в автосалоні.
  + UpdateDate: Дата останнього оновлення запису про статус автомобіля в автосалоні.
* Countries - містить перелік країн, який використовується для визначення країн-виробників автомобілів або брендів.
  + Id: Унікальний ідентифікатор країни.
  + Name: Назва країни.
  + CreateDate: Дата створення запису про країну.
  + UpdateDate: Дата останнього оновлення запису про країну.
* CarTypes – класифікує автомобілі за типом, наприклад, седан, кросовер, хетчбек.
  + Id: Унікальний ідентифікатор типу автомобіля.
  + Name: Назва типу автомобіля.
  + CreateDate: Дата створення запису про тип автомобіля.
  + UpdateDate: Дата останнього оновлення запису про тип автомобіля.
* Cities - зберігає інформацію про міста, зв'язані з ідентифікаторами країн.
  + Id: Унікальний ідентифікатор міста.
  + Name: Назва міста.
  + CountryId: Ідентифікатор країни, до якої належить це місто. Це може бути зовнішній ключ, який посилається на таблицю країн або код країни.
  + CreateDate: Дата створення запису про місто.
  + UpdateDate: Дата останнього оновлення запису про місто.
* Colors - включає список кольорів автомобілів для ідентифікації та класифікації.
  + Id: Унікальний ідентифікатор кольору.
  + Name: Назва кольору.
  + CreateDate: Дата створення запису про колір.
  + UpdateDate: Дата останнього оновлення запису про колір.
* ComfortOptions
  + Id: Унікальний ідентифікатор опції комфорту.
  + OptionName: Назва або опис опції комфорту.
  + CreateDate: Дата створення запису про опцію комфорту.
  + UpdateDate: Дата останнього оновлення запису про опцію комфорту.
* Conditions - відображає різні умови лізингу автомобілів.
  + Id: Унікальний ідентифікатор умови автомобіля.
  + Name: Назва або опис умови автомобіля.
  + CreateDate: Дата створення запису про умову автомобіля.
  + UpdateDate: Дата останнього оновлення запису про умову. автомобіля.
* Brands - класифікує автомобільні бренди, що включає інформацію про країну-виробника.
  + Id: Унікальний ідентифікатор бренду.
  + Name: Назва бренду.
  + CountryId: Ідентифікатор країни, до якої належить бренд.
  + CreateDate: Дата створення запису про бренд.
  + UpdateDate: Дата останнього оновлення запису про бренд.
* CarBodyTypes - містить типи кузовів автомобілів, наприклад, седан, хетчбек, пікап тощо.
  + Id: Унікальний ідентифікатор типу кузова автомобіля.
  + Name: Назва типу кузова автомобіля.
  + CreateDate: Дата створення запису про тип кузова автомобіля.
  + UpdateDate: Дата останнього оновлення запису про тип кузова автомобіля.
* MultimediaOptions - класифікує мультимедійні опції, доступні для автомобілів.
  + Id: Унікальний ідентифікатор опції мультимедіа.
  + OptionName: Назва опції мультимедіа.
  + CreateDate: Дата створення запису про опцію мультимедіа.
  + UpdateDate: Дата останнього оновлення запису про опцію мультимедіа.
* PaymentMethods - відповідає за методи оплати, які можна використовувати при купівлі або оренді автомобілів.
  + Id: Унікальний ідентифікатор методу оплати.
  + Name: Назва методу оплати.
  + CreateDate: Дата створення запису про метод оплати.
  + UpdateDate: Дата останнього оновлення запису про метод оплати.
* SafetyOptions - визначає опції безпеки, які можуть бути встановлені в автомобілях.
  + Id: Унікальний ідентифікатор опції безпеки.
  + OptionName: Назва опції безпеки.
  + CreateDate: Дата створення запису про опцію безпеки.
  + UpdateDate: Дата останнього оновлення запису про опцію безпеки.
* SaleStatuses - містить статуси продажу автомобілів, такі як відкритий, закритий, очікує
  + Id: Унікальний ідентифікатор статусу продажу.
  + Status: Назва статусу продажу, така як "Відкритий", "Закритий" тощо.
  + CreateDate: Дата створення запису про статус продажу.
  + UpdateDate: Дата останнього оновлення запису про статус продажу.
* LeaseProposalConditions - містить умови, які застосовуються до конкретних пропозицій оренди, деталізуючи вимоги та умови пропозицій.
  + LeaseProposalId: Ідентифікатор пропозиції щодо оренди, до якої відноситься ця умова.
  + ConditionId: Ідентифікатор умови, пов'язаної з пропозицією щодо оренди.
  + Value: Значення умови пропозиції щодо оренди.
  + Id: Унікальний ідентифікатор запису про умову пропозиції щодо оренди.
  + CreateDate: Дата створення запису про умову пропозиції щодо оренди.
  + UpdateDate: Дата останнього оновлення запису про умову пропозиції щодо оренди.
* CarSafetyOptions - визначає наявність та типи опцій безпеки, які встановлені у конкретному автомобілі.
  + CarId: Унікальний ідентифікатор автомобіля.
  + SafetyOptionId: Унікальний ідентифікатор варіанту системи безпеки автомобіля.
  + CreateDate: Дата створення запису про опцію безпеки для автомобіля.
  + UpdateDate: Дата останнього оновлення запису про опцію безпеки для автомобіля.
* CarMultiMediaOptions - записує інформацію про мультимедійні опції, які інстальовані в автомобілі.
  + CarId: Унікальний ідентифікатор автомобіля, до якого відноситься дана мультимедійна опція.
  + MultimediaOptionId: Унікальний ідентифікатор мультимедійної опції, яка доступна для автомобіля.
  + CreateDate: Дата створення запису про мультимедійну опцію для автомобіля.
  + UpdateDate: Дата останнього оновлення запису про мультимедійну опцію для автомобіля.
* CarComfortOptions - тримає інформацію про опції комфорту, доступні в певному автомобілі.
  + CarId: Унікальний ідентифікатор автомобіля, до якого відноситься конкретна опція комфорту.
  + ComfortOptionId: Унікальний ідентифікатор конкретної опції комфорту, яка доступна для автомобіля.
  + CreateDate: Дата створення запису про зв'язок між автомобілем та опцією комфорту.
  + UpdateDate: Дата останнього оновлення запису про зв'язок між автомобілем та опцією комфорту.

*Зв’язки таблиць:*

* CityId в таблиці Autodealerships – зв'язок багато до одного між містами та автосалонами. Багато автосалонів можуть бути розташовані в одному місті.
* CarTypeId в таблиці Cars – зв'язок багато до одного між типами автомобілів та автомобілями. Багато автомобілів можуть мати один тип.
* BrandId в таблиці Cars – зв'язок багато до одного між брендами та автомобілями. Багато автомобілів можуть належати до одного бренду.
* ColorId в таблиці Cars – зв'язок багато до одного між кольорами та автомобілями. Багато автомобілів можуть бути одного кольору.
* EngineId в таблиці Cars – зв'язок багато до одного між двигунами та автомобілями. Багато автомобілів можуть мати один тип двигуна.
* GearBoxId в таблиці Cars – зв'язок багато до одного між типами коробок передач та автомобілями. Багато автомобілів можуть мати один тип коробки передач.
* CustomerId в таблиці CarSales – зв'язок багато до одного між клієнтами та продажами. Один клієнт може мати багато продажів.
* EmployeeId в таблиці CarSales та Leases – зв'язок багато до одного між працівниками та продажами/лізингами. Один працівник може обробити багато продажів чи угод лізингу.
* DealershipId в таблиці DealershipCars – зв'язок багато до одного між автосалонами та наявними в них автомобілями. Один автосалон може мати багато автомобілів.
* DistributorId в таблиці CarDeliveries – зв'язок багато до одного між дистриб'юторами та доставками. Один дистриб'ютор може здійснювати багато доставок.
* LeaseProposalId в таблиці Leases – зв'язок багато до одного між пропозиціями лізингу та угодами лізингу. Одна пропозиція лізингу може бути використана в багатьох угодах лізингу.
* CarId в таблицях CarDeliveries, DealershipCars, CarSales – зв'язок багато до одного між автомобілями та їхньою доставкою, наявністю в автосалонах, продажами. Один автомобіль може бути проданий, доставлений, і мати запис про наявність в автосалоні.
* CountryId в таблицях Autodealerships (через Cities) і Brands – зв'язок багато до одного між країнами та містами (де розташовані автосалони) або брендами (країна бренду). Багато міст або брендів можуть належати до однієї країни.
* LeaseTypeId в таблиці LeaseProposals – зв'язок багато до одного між типами лізингу та пропозиціями лізингу. Багато пропозицій лізингу можуть мати один тип лізингу.

*Додані тригери:*

* Тригер на оновлення UpdateDate – встановлений тригер на оновлення поля метаданих UpdateDate для всіх таблиць при оновленні будь-якого поля таблиці.
  + 1. **Вимоги до архівних даних для OLTP системи**

Джерелом історичних даних є дані у форматі csv. Для кожної таблиці створений окремий csv файл, всередині якого в кожному рядку знаходиться один запис з аналогічними назвами атрибутів і типів даних до колонок OLTP БД, розділених комами. Для забезпечення унікальності доданих даних в БД застосований тип Serial, який додає для integer автоматичний інкремент, який застосовується до первинного ключа.

Скрипт додавання даних копіює значення кожного рядка у колонки з відповідними назвами БД за допомогою функції COPY FROM з вказанням формату файлу csv. Для даних, що обчислюються на основі інших, застосовуються відповідні скрипти оновлення колонок, а саме обчислення суми оплати за курс та суми виплати зарплати за контрактом викладача.

Обсяг основних даних: автосалони – 14264, бренди – 181, авто – 54270, продажі – 54270, країни – 247, робітники – 15000, доставки – 54270, лізинги – 2000, тарифи лізингів – 200, клієнтів – 54270.

* + 1. **Вимоги до ETL**

ETL процес буде здійснений для даних, період яких починається 01.01.2014 та закінчується 31.12.2017, тобто період даних 3 роки. Кількість записів у основних таблицях відповідає обсягу основних історичних даних.

Процес ETL буде поділений на такі частини:

* Створення стейджингу – тимчасової БД без зв’язків, обмежень та тільки з необхідними для сховища даними;
* Виконання первинної заливки даних в сховище за допомогою збережених процедур – створюється дві збережені процедури, які будуть запускатись послідовно, у порядку, визначеному за допомогою програмного коду або прямого виклику командою CALL. Перша процедура буде виконувати копіювання всіх необхідних для задач аналітики даних з OLTP БД в стейджинг, друга – виконує копіювання простих даних зі стейджингу в сховище даних, формування таблиці транзакційного факту та агрегування даних для формування таблиці кумулятивного факту за допомогою трансформацій;
* Виконання інкрементальної заливки даних в сховище за допомогою збережених процедур – створюється дві збережені процедури, які будуть запускатись послідовно, у порядку, визначеному за допомогою програмного коду або прямого виклику командою CALL. Перша процедура буде виконувати копіювання тих даних, які були додані або оновлені після попередньої заливки, враховуючи додання даних батьківських таблиць з OLTP БД в стейджинг, друга – виконує злиття оновлених даних відповідно до типу вимірів та фактів зі стейджингу в сховище та додавання тих даних, яких в ньому ще немає.
  1. **Постановка завдання**

*Визначені завдання для транзакційної таблиці фактів (лізинг авто):*

* Аналіз зміни ціни угоди на лізинг. Метрика - відсоток зміни ціни між двома транзакціями.
* Аналіз частоти лізингу конкретного авто. Метрика - дата останньої транзакції з цим автомобілем.

*Визначені завдання для кумулятивної таблиці фактів (продаж курсів):*

* Аналіз доходу від продажу авто конкретної марки. Метрика – кількість проданих авто певної марки за минулий місяць.
* Аналіз прибутковості салону. Метрика - відсоток одного автосалону від загального доходу мережі за минулий місяць.

Для побудови сховища даних у проекті бази даних для мережі автосалонів, були визначені ключові виміри та факти, які дозволяють здійснювати глибокий аналіз бізнес-процесів, пов'язаних з продажем автомобілів і лізингом. Вимірами, необхідними для формування факту продажу автомобілів, є бренд автомобіля, автосалон, де здійснено продаж, і час кладення угоди, що здійснив цей продаж. Ці виміри дозволяють формувати детальну інформацію про кожну групу продажів, що є критично важливим для кумулятивного факту продажів. Додатково визначено виміри методу оплати і договору лізингу, що дозволяють відстежувати деталізацію лізингових угод.

Для фактів, визначено кумулятивну таблицю фактів для процесу продажів автомобілів, яка фіксує інформацію про завершені продажі, включаючи дату закінчення продажу, після якої зареєструватися на купівлю вже не можливо. Це дозволяє проводити аналіз загальних продажів, виявляти тренди в попиті на певні моделі та марки автомобілів, а також оцінювати ефективність різних автосалонів.

Транзакційна таблиця фактів для процесу виплати зарплати викладачам (у нашому випадку продавцям) зареєстрована як окрема подія, що включає всю необхідну інформацію про кожну виплату. Це дозволяє аналізувати зарплати, здійснювати порівняння між транзакціями, оцінювати ефективність і мотиваційні аспекти працівників в різних локаціях.

Завдяки цим визначенням можна будувати деталізовані звіти та аналітику, що допомагають керівництву автосалону приймати обґрунтовані рішення, базуючись на актуальних даних із сховища.

Загальні вимоги до ПЗ:

* Відображати дані OLTP БД в коректній, сприйнятній для зовнішніх користувачів системи формі (не відображати ключові атрибути, спрощення структур даних) з можливістю багатоопційної фільтрації основних даних.
* Надавати можливість додавати нові дані, модифікувати існуючі дані в OLTP БД, відміняти здійснені операції.
* Відображати метадані про підключене сховище даних (назва, список таблиць, їх тип, кількість кортежів, дані про останню заливку даних, кількість залитих даних і т.д.).
* Виконувати первинну та інкрементальну заливку даних в сховище та відображати метадані про здійснений процес заливки.
* Відображати дані сховища в коректній багатовимірній формі з повним набором фільтрів.
* Формувати та відображати звіти для мінімум 4 визначених задач аналізу (звіти повинні включати зведену текстову інформацію та діаграми різних типів).
* Надавати можливість експортувати дані із сховища даних відповідно до формального опису збережених метаданих.

**РОЗДІЛ 2. ПРОЕКТУВАННЯ СХОВИЩА ДАНИХ ТА ІМПЛЕМЕНТАЦІЯ ETL ПАКЕТІВ**

**2.1. Багатовимірне моделювання предметної області**

У відповідності до задач бізнес-аналізу, визначених у підрозділі 1.3, визначено такі базові таблиці фактів для бізнес-процесів: для продажу авто – CarSales, для лізингу авто – Leases.

Для забезпечення виконання задач та формування метрик необхідні такі дані:

* Бренд – назва, країна.
* Автосалон – назва автосалону, країна.
* Автомобіль – бренд, покоління, модель.
* Акт продажу – автосалон, де була здійснена покупка, бренд автомобіля, прибуток за минулий місяць, дата початку місяця, дата кінця місяця, загальний дохід за це місяць, процент зміни доходу між доходом за цей і минулий місяць, кількість продажів за минулий місяць, кількість продажів за цей місяць і різниця в кількості продажів.
* Дата – рік, місяць, день.
* Акт укладання лізингу – автомобіль, який був взятий в оренду, дата початку лізингу та дата кінця лізингу, ціна угоди про лізинг.

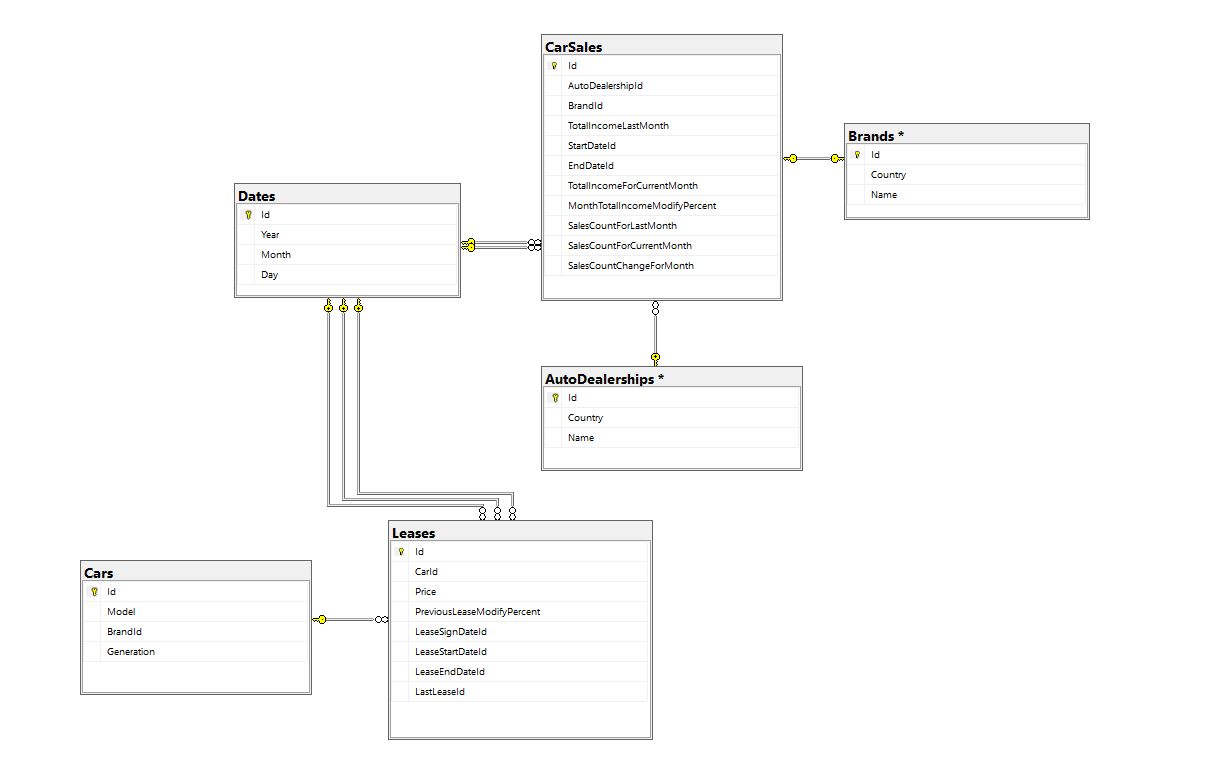
****

Рис. 2.1. Схема сховища даних

*Тип схеми сховища даних*: сузір’я фактів, оскільки СД містить 2 таблиці фактів, CarSales i Leases. Таблиці фактів містять спільні вимір Dates, Brands, Cars, а також таблиця CarSales має власний вимір – Autodealership.

*Опис таблиць сховища даних:*

1. Brands – повільно змінний вимір 1 типу, тому що зміна атрибутів виміру відбувається нечасто й не потребує створення нового рядка, або збереження старого значення атрибуту. Вимір характеризує окремий бренд автомобіля. Атрибути:

* Id – integer, ідентифікатор бренду в сховищі, є первинним ключем.
* Name – varchar, назва бренду.
* Country – varchar, назва країни.

1. AutoDealerships – повільно змінний вимір 1 типу, тому що зміна атрибутів виміру відбувається нечасто й не потребує створення нового рядка, або збереження старого значення атрибуту. Вимір характеризує офіс, в якому проводиться певний курс. Атрибути:

* Id – integer, ідентифікатор автосалону в сховищі, є первинним ключем.
* Name – varchar, назва бренду.
* Country – varchar, назва країни.

1. Cars – повільно змінний вимір 1 типу, тому що зміна атрибутів виміру відбувається нечасто й не потребує створення нового рядка, або збереження старого значення атрибуту. Вимір характеризує окремий автомобіль. Атрибути:

* Id – integer, ідентифікатор автомобілю в сховищі, є первинним ключем.
* Model – varchar, модель автомобіля.
* Generation – varchar, покоління автомобіля.
* BrandId – integer, ідентифікатор бренду автомобіля у сховищі.

1. DateDim – повільно змінний вимір 2 типу, тому що під час оновлення значення дати краще додати нове значення у вимір, адже старе значення може також знадобитись в майбутньому для інших записів. Вимір характеризує окрему дату, що поділена на номер дня, місяця та року. Атрибути:

* Id – integer, ідентифікатор дати в сховищі, є первинним ключем.
* Year – integer, рік дати.
* Month – integer, номер місяця дати.
* Day – integer, номер дня дати.

1. Leases – таблиця фактів транзакційного типу, тому що повністю описує одну окрему транзакцію укладення угоди лізингу, містить інформацію про оплату та відсоток зміни зарплати за місяць. Атрибути:

* Id – integer, ідентифікатор акту лізингу в сховищі, є первинним ключем.
* CarId – integer, посилання на контракт викладача, є зовнішнім ключем.
* Price – numeric, загальна сума угоди.
* LeaseSignDateId – integer, посилання на дату підписання угоди, є зовнішнім ключем.
* LeaseStartDateId – integer, посилання на дату початку угоди лізингу, є зовнішнім ключем.
* LeaseEndDateId – integer, посилання на дату кінця періоду лізингу, є зовнішнім ключем.
* payment\_period\_end\_date\_id – integer, посилання на дату закінчення періоду оплати, є зовнішнім ключем.
* LastLeaseId – integer, посилання на попередній лізинг, є зовнішнім ключем.
* PreviousLeaseModifyPercent – numeric, відсоток зміни ціни лізингу за місяць, є неадитивним фактом, оскільки не сумується за вимірами.

1. CarSales – таблиця фактів кумулятивного типу, тому що описує обмежену в кінцевій даті операцію продажу авто, а потім кумулює ці значення в єдиний запис. Містить інформацію про автосалон, в якому була здійснена покупка, а також кількість екземплярів, що була куплена, сумарний дохід з продажу та інтервал між купівлею автомобілів, що мали однакову марку та куплені в одному автосалоні. Атрибути:

* Id – integer, ідентифікатор акту продажу в сховищі, є первинним ключем.
* TotalIncomeLastMonth – numeric, сума доходу з продажу за минулий місяць, є адитивним фактом, оскільки сумується за всіма вимірами.
* TotalIncomeForCurrentMonth – numeric, сума доходу з продажу за даний місяць, є адитивним фактом, оскільки сумується за всіма вимірами.
* MonthTotalIncomeModifyPercent – numeric, різниця доходу між останнім місяцем і теперишнім, є адитивним фактом, оскільки сумується за всіма вимірами.
* SalesCountForLastMonth – numeric, кількість продажів за минулий місяць.
* StartDateId – integer, посилання на дату початку групи, є зовнішнім ключем.
* EndDateId– integer, посилання на дату закінчення групи, є зовнішнім ключем.
* SalesCountForCurrentMonth – numeric, кількість продажів за теперешній місяць.
* SalesCountChangeForMonth – integer, numeric, різниця продажів за минулий та теперишній місяць.
* AutoDealershipId – integer, посилання на автосалон, у якому продаються авто, є зовнішнім ключем.
* BrandId – integer, посилання на бренд, який буде проводити курс, є зовнішнім ключем.

**2.2. Проектування процесу ETL**

Для проектування процесу ETL для таблиці CourseDim необхідно взяти дані з таблиць courses, languages, categories, difficulty\_levels OLTP БД. Таблиця не містить ніяких трансформацій, тому виконується пряме копіювання даних з БД.

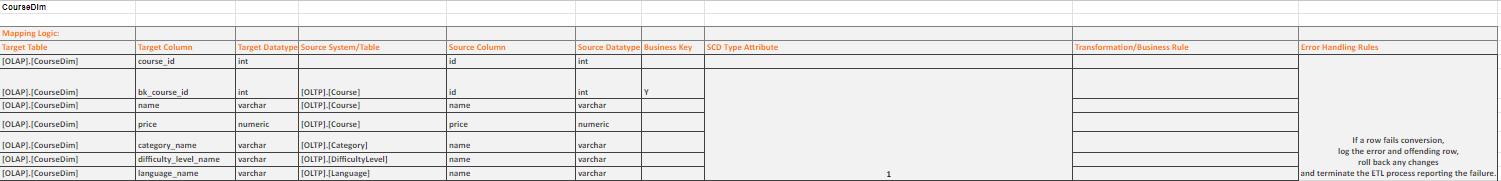


Рис. 2.2. STM-таблиця для CourseDim

Для таблиці PaymentMethodDim необхідно взяти дані з таблиці payment\_methods OLTP БД, таблиця не містить ніяких трансформацій, тому виконується пряме копіювання даних з БД.

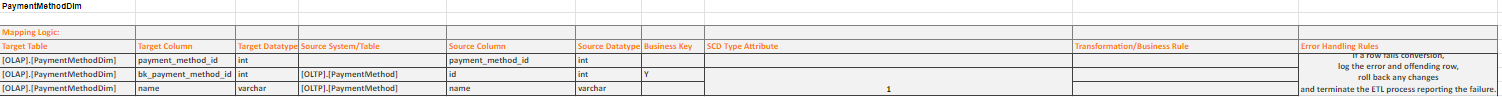


Рис. 2.3. STM-таблиця для PaymentMethodDim

Для таблиці OfficeDim дані взяті з таблиці offices, cities, countries OLTP БД. Таблиця не містить ніяких трансформацій, тому виконується пряме копіювання даних з БД.

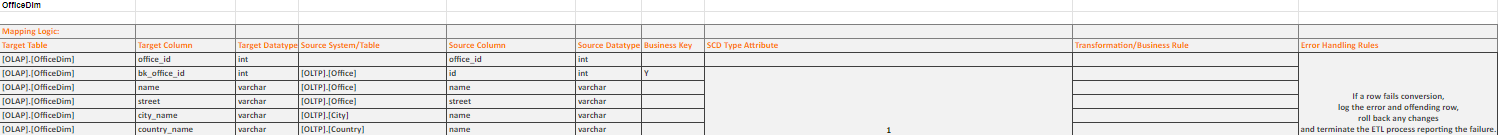


Рис. 2.4. STM-таблиця для OfficeDim

Для таблиці TeacherDim дані взяті з таблиці teachers OLTP БД. Таблиця не містить ніяких трансформацій, тому виконується пряме копіювання даних з БД.

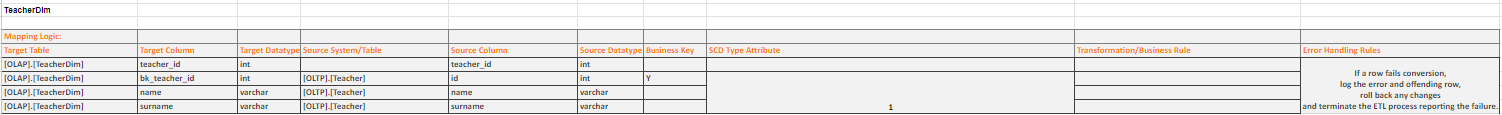


Рис. 2.5. STM-таблиця для TeacherDim

Для таблиці DateDim дані взяті з таблиці groups, student\_contracts, teacher\_contracts, teacher\_contract\_payments OLTP БД. Таблиця містить трансформації дат з вихідних таблиць БД для формування виміру в сховищі, а саме за допомогою команди EXTRACT витягує рік, місяць та день певної дати, записує їх у поля date\_year, date\_month, date\_day відповідно. Для формування переліку дат використовується підзапит з UNION для всіх дат, що є у сховищі.

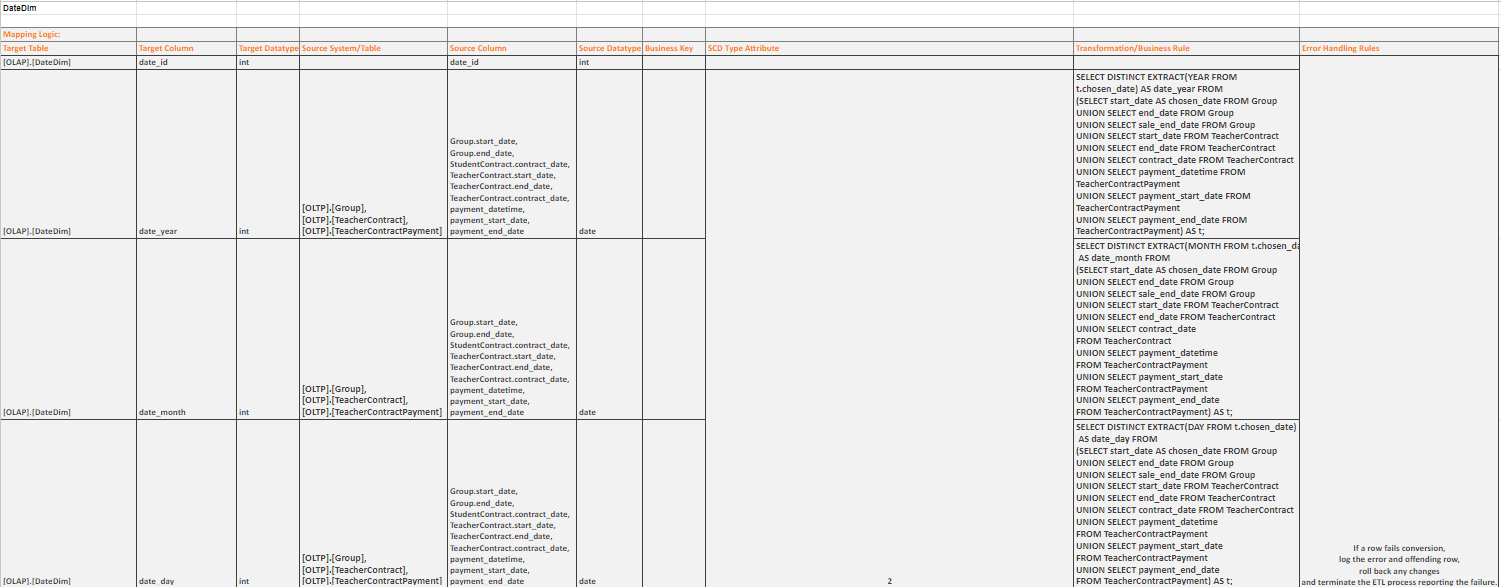


Рис. 2.6. STM-таблиця для DateDim

Для таблиці TeacherContractDim дані взяті з таблиці teacher\_contracts OLTP БД. Таблиця містить трансформації для співставлення start\_date, end\_date, contract\_date з БД та виміру дат, формування зовнішніх ключів, які будуть зв’язувати ці дати з відповідними рядками виміру дат.

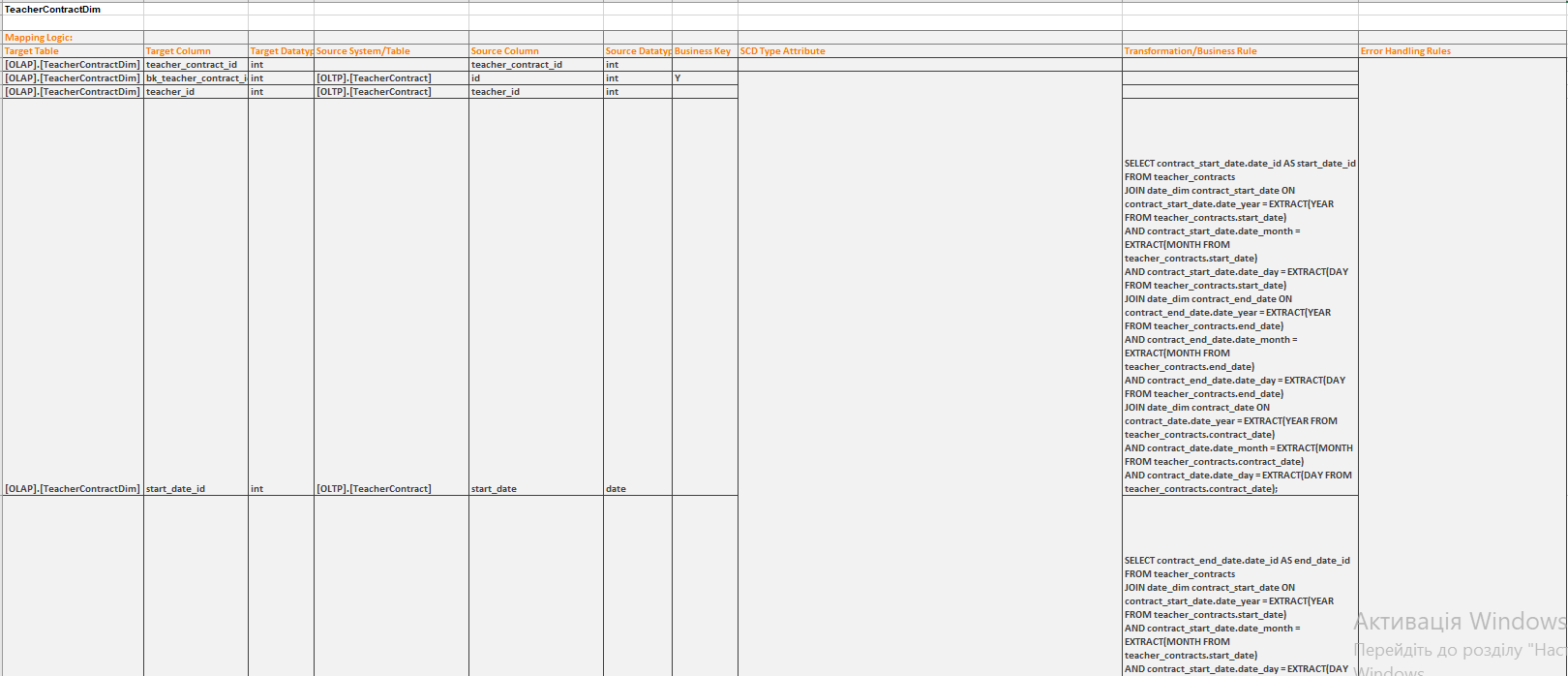


Рис. 2.7. STM-таблиця для TeacherContractDim (Частина 1)

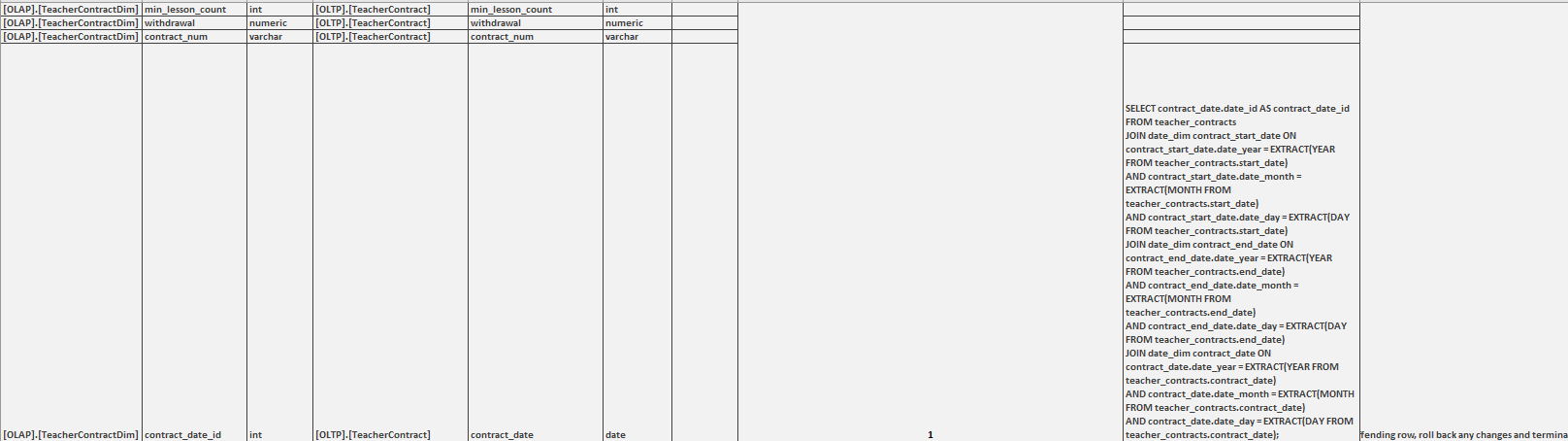


Рис. 2.8. STM-таблиця для TeacherContractDim (Частина 2)

Для таблиці TeacherContractPaymentFact дані взяті з таблиці teacher\_contract\_payments OLTP БД. Таблиця містить трансформації для формування метрики salary\_change\_percentage за допомогою колонки total\_sum та функції LAG для отримання попередньої зарплати викладача, а також трансформації для співставлення payment\_date, payment\_period\_end, payment\_period\_start з БД та виміру дат, формування зовнішніх ключів, які будуть зв’язувати ці дати з відповідними рядками виміру дат.

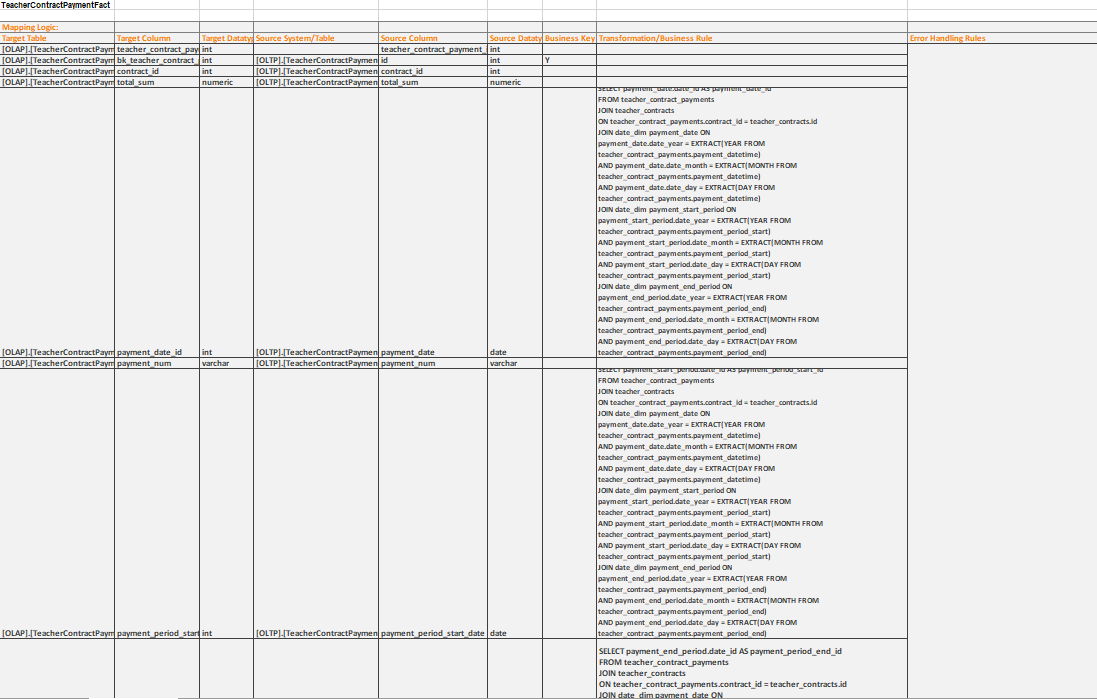


Рис. 2.9. STM-таблиця для TeacherContractPaymentFact (Частина 1)

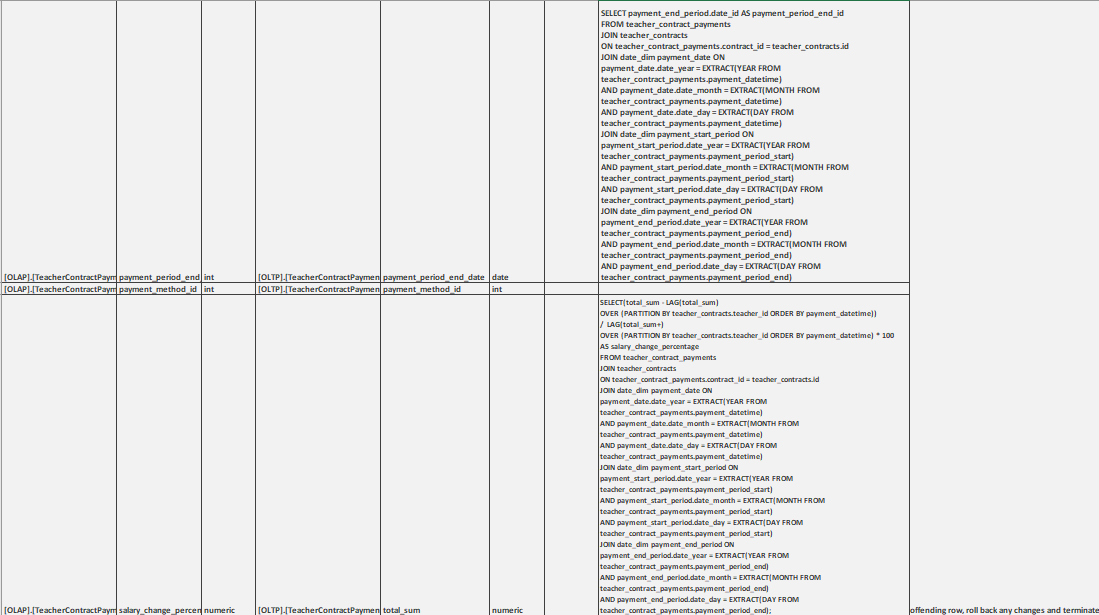


Рис. 2.10. STM-таблиця для TeacherContractPaymentFact (Частина 2)

Для таблиці GroupContractFact дані взяті з таблиці groups, student\_contracts OLTP БД. Таблиця містить трансформації для формування метрик income\_sum, student\_count, same\_course\_interval за допомогою колонок price таблиці courses, id таблиці student\_contracts, start\_date та end\_date таблиці groups, використання агрегатних функцій (SUM, COUNT) та використання функції LAG для отримання попереднього курсу, що проводив той самий викладач в тому самому офісі. Також містить трансформації для співставлення sale\_end\_date, end\_date, start\_date таблиці groups з БД та виміру дат, формування зовнішніх ключів, які будуть зв’язувати ці дати з відповідними рядками виміру дат.

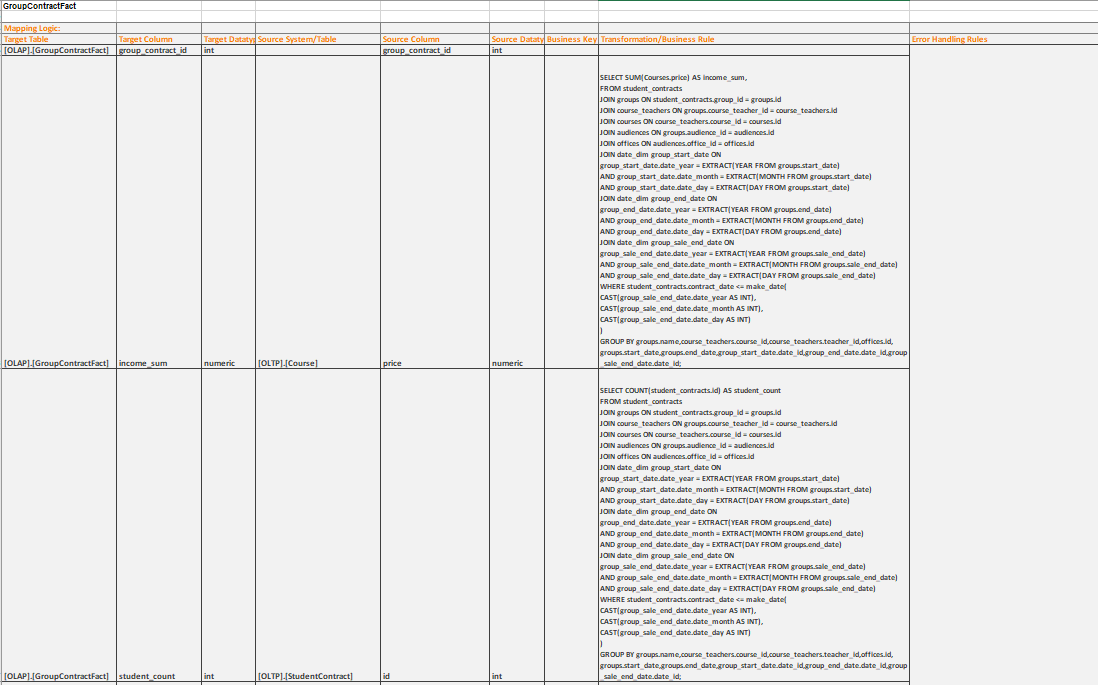


Рис. 2.11. STM-таблиця для GroupContractFact (Частина 1)



Рис. 2.12. STM-таблиця для GroupContractFact (Частина 2)

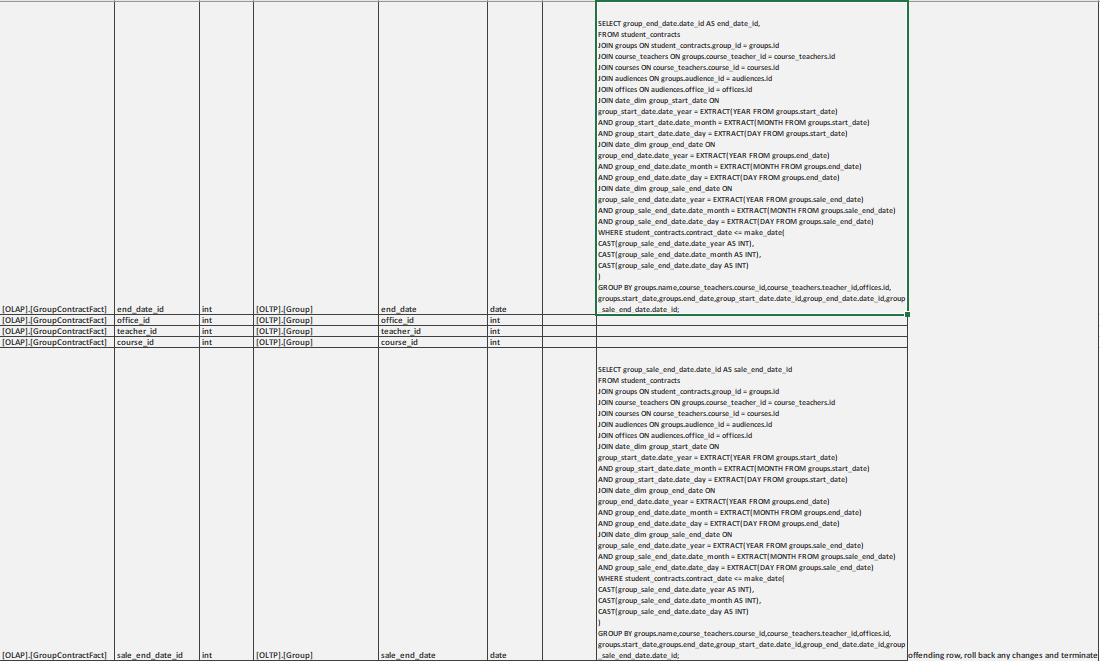


Рис. 2.13. STM-таблиця для GroupContractFact (Частина 3)

**2.3. Імплементація первинного завантаження даних**

Для реалізації первинного завантаження даних реалізовано скрипти заповнення БД метаданих, завантаження первинних даних з OLTP БД в стейджингову БД та з стейджингової БД в сховище даних.

Заповнення БД метаданих відбувається наступним чином: вибір за допомогою системних таблиць і функцій, таких як INFORMATION\_SCHEMA та DIAGNOSTICS, проходження по всіх рядках таблиці за допмогою FOR та додавання відповідних значень. Заповнення відбувається напівавтоматично, оскільки колонки типу фактів, опису метрик, типу вимірів та правил трансформації вимагають ручного заповнення.

*Приклад заповнення таблиці з БД метаданих:*

FOR table\_record IN SELECT \* FROM dblink('CompCourses-oltp-conn', 'SELECT table\_name FROM information\_schema.tables WHERE table\_schema = ''public'' AND table\_type = ''BASE TABLE''') AS t(table\_name text)

LOOP

primary\_key\_column := (SELECT column\_name FROM dblink('CompCourses-oltp-conn',

'SELECT kcu.column\_name

FROM information\_schema.table\_constraints tc

JOIN information\_schema.key\_column\_usage kcu

ON tc.constraint\_name = kcu.constraint\_name

WHERE tc.table\_name = ''' || table\_record.table\_name || '''

AND tc.constraint\_type = ''PRIMARY KEY''

LIMIT 1') AS t(column\_name TEXT));

EXECUTE 'INSERT INTO source\_table (source\_db\_id, source\_table\_name, key\_name) VALUES (1, $1, $2)'

USING table\_record.table\_name, primary\_key\_column;

END LOOP;

Повний скрипт заповнення таблиць БД метаданих знаходиться в додатку Г.

Для завантаження первинних даних з OLTP БД в стейджингову БД відбувається копіювання даних з OLTP БД тих таблиць, які присутні в стейджингу за допомогою команди INSERT INTO (column1,column2) SELECT, де вказується вибірка всіх даних з певної таблиці.

*Приклад завантаження даних таблиці з OLTP БД в стейджинг:*

INSERT INTO languages(id,name)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,name FROM languages')

AS language\_columns(id integer,name varchar);

Повний скрипт первинного завантаження даних з OLTP БД в стейджинг для наведено в додатку Д.

Для завантаження первинних даних з стейджингу в сховище відбувається вставка даних з стейджингу в сховище за допомогою команди INSERT INTO (column1,column2) SELECT. Після виконання кожної вставки даних застосовується оновлення значень для вставки в БД метаданих.

*Приклад оновлення значень для запису в БД метаданих:*

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''teacherdim''');

END IF;

Для обчислення метрик застосовано трансформації з використанням віконних та агрегатних функцій.

*Скрипт для обчислення метрики відсотку зміни зарплати викладача:*

SELECT

(total\_sum - LAG(total\_sum) OVER (PARTITION BY teacher\_contracts.teacher\_id ORDER BY payment\_datetime))

/ LAG(total\_sum) OVER (PARTITION BY teacher\_contracts.teacher\_id ORDER BY payment\_datetime) \* 100

AS salary\_change\_percentage

Для обрахування метрик суми прибутку та кількості студентів застосовуються агрегатні функції SUM та COUNT.

*Скрипт для обчислення метрики інтервалу між тим самим курсом, що провів той самий викладач в тому самому офісі:*

groups.start\_date - LAG(groups.end\_date) OVER

(PARTITION BY course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id ORDER BY groups.start\_date)

AS same\_course\_interval

Після заповнення даних в сховище відбувається додавання даних про успішну заливку в БД метаданих та очистка стейджингу за допомогою команди TRUNCATE.

*Скрипт для додавання даних про успішну заливку в БД метаданих:*

-- Insert data into the data\_load\_history table and capture the datetime for calculation

DECLARE @meta\_start\_time DATETIME = GETDATE();

DECLARE @row\_count INT = load\_rows;

DECLARE @dw\_table\_count INT = tables\_count;

DECLARE @db\_table\_count INT = tables\_count;

DECLARE @data\_load\_history\_id INT;

-- Calculate the duration in seconds since the start time

DECLARE @load\_time FLOAT = DATEDIFF(SECOND, @meta\_start\_time, GETDATE());

-- Insert statement using linked server

EXEC ('INSERT INTO CompCoursesMetadata.dbo.data\_load\_history

(load\_datetime, load\_time, load\_rows, affected\_table\_count, source\_table\_count)

VALUES (GETDATE(), ?, ?, ?, ?)',

@load\_time, @row\_count, @dw\_table\_count, @db\_table\_count) AT CompCoursesMetadata;

-- Retrieve the last inserted ID from data\_load\_history

EXEC ('SELECT @output = MAX(data\_load\_history\_id) FROM data\_load\_history',

@data\_load\_history\_id OUTPUT) AT CompCoursesMetadata;

-- Assuming insert\_statements is an array of SQL strings (handled in application code or script)

-- Iterate over insert\_statements and replace placeholder with the actual ID

-- Example of one such statement:

DECLARE @sql NVARCHAR(MAX);

-- Here's an example of how the dynamic SQL could be set up in the application:

SET @sql = 'INSERT INTO some\_table (data\_load\_history\_id, other\_column) VALUES (?, ?)';

-- You would replace the placeholders with actual values and execute:

EXEC sp\_executesql @sql, N'@data\_load\_history\_id INT, @other\_column\_value INT',

@data\_load\_history\_id, 456;

Повний скрипт первинного завантаження даних з стейджингу в сховище для інших таблиць наведено в додатку Д.

**2.4. Імплементація інкрементального завантаження даних**

Для реалізації первинного завантаження даних реалізовано скрипти інкрементального завантаження даних з OLTP БД в стейджингову БД та з стейджингової БД в сховище.

Для інкрементального завантаження даних з OLTP БД в стейджинг

Повний скрипт інкрементального завантаження даних з OLTP БД в стейджинг наведено в додатку Е.

Для інкрементального завантаження даних з стейджингу в сховище необхідно використати команду MERGE, в якій вказати таблицю, в яку зливати дані та джерело, з якого будуть братись дані для злиття. Для формування джерела пишеться підзапит, який витягує дані з таблиці стейджингу. Далі вказується, що робити, коли дані джерела та цілі співпали, а коли ні. Для виміру дат та таблиці кумулятивного факту використовується лише умова на додавання даних в сховище даних.

*Скрипт для злиття таблиці в сховищі:*

MERGE INTO teacherdim target

USING (

SELECT \* FROM dblink('CompCourses-staging',

'SELECT id,name,surname FROM teachers')

AS teacher\_columns(id integer,name varchar,surname varchar)

) AS source

ON source.id = target.bk\_teacher\_id

WHEN MATCHED THEN

UPDATE SET name = source.name,

surname = source.surname

WHEN NOT MATCHED THEN

INSERT (name,surname) VALUES(source.name,source.surname);

Після заповнення даних в сховище відбувається додавання даних про успішну заливку в БД метаданих та очистка стейджингу за допомогою команди TRUNCATE.

*Скрипт для додавання даних про успішну заливку в БД метаданих:*

PERFORM dblink\_exec('CompCourses-metadata',

format('INSERT INTO data\_load\_history(load\_datetime, load\_time, load\_rows, affected\_table\_count, source\_table\_count) VALUES (%L, EXTRACT(EPOCH FROM %L::timestamp - %L::timestamp), %L, %L, %L)',

clock\_timestamp(), clock\_timestamp(), meta\_start\_time, row\_count, dw\_table\_count, db\_table\_count));

SELECT \* INTO last\_id\_result FROM dblink('CompCourses-metadata', 'SELECT MAX(data\_load\_history\_id) as last\_id FROM data\_load\_history') AS t(last\_id INT);

data\_load\_history\_id := last\_id\_result.last\_id;

FOREACH generated\_sql IN ARRAY insert\_statements

LOOP

PERFORM dblink\_exec('CompCourses-metadata', format(generated\_sql, data\_load\_history\_id));

END LOOP;

Повний скрипт інкрементального завантаження даних з стейджингу в сховище наведено в додатку Е.

**РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ БІЗНЕС-АНАЛІТИКИ ДЛЯ МЕРЕЖІ ОФІСІВ З ПРОДАЖУ КОМП’ЮТЕРНИХ КУРСІВ**

**3.1. Реалізація архітектури ПЗ та доступу до даних**

**3.2. Реалізація функціональних характеристик системи**

**3.3. Опис роботи системи**

**ВИСНОВКИ**

**СПИСОК ЛІТЕРАТУРИ**

1. Інформація про базовий комп’ютерний курс [Електронний ресурс]. – Режим доступу: https://danco.com.ua/ua/courses/kursi-koristuvachiv-pk.
2. Інформація про просунуті теми курсів [Електронний ресурс]. – Режим доступу: https://lviv.logos-academy.com.
3. Аналіз попиту на IT-спеціалістів [Електронний ресурс]. – Режим доступу: https://forbes.ua/inside/bezumnyy-spros-prodolzhaetsya-zarplata-ukrainskikh-razrabotchikov-za-god-vyrosla-na-3040-kogo-ishchut-it-kompanii-20012022-3298 (2022).
4. Інформація про основні критерії компаній з продажу курсів [Електронний ресурс]. – Режим доступу: https://mate.academy.

**ДОДАТКИ**

**Додаток А. Скрипт створення бази даних**

CREATE DATABASE "CompCourses"

WITH

OWNER = postgres

ENCODING = 'UTF8'

LC\_COLLATE = 'Ukrainian\_Ukraine.1251'

LC\_CTYPE = 'Ukrainian\_Ukraine.1251'

LOCALE\_PROVIDER = 'libc'

TABLESPACE = pg\_default

CONNECTION LIMIT = -1

IS\_TEMPLATE = False;

CREATE TABLE IF NOT EXISTS public.categories

(

id integer NOT NULL DEFAULT nextval('categories\_id\_seq'::regclass),

name character varying(50) COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT categories\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.countries

(

id integer NOT NULL DEFAULT nextval('countries\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT countries\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.cities

(

id integer NOT NULL DEFAULT nextval('cities\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

country\_id integer,

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT cities\_pkey PRIMARY KEY (id),

CONSTRAINT cities\_country\_id\_fkey FOREIGN KEY (country\_id)

REFERENCES public.countries (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.days\_of\_week

(

id integer NOT NULL DEFAULT nextval('days\_of\_week\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT days\_of\_week\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.difficulty\_levels

(

id integer NOT NULL DEFAULT nextval('difficulty\_levels\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT difficulty\_levels\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.languages

(

id integer NOT NULL DEFAULT nextval('languages\_id\_seq'::regclass),

name character varying(50) COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT languages\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.lesson\_types

(

id integer NOT NULL DEFAULT nextval('lesson\_types\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT lesson\_types\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.offices

(

id integer NOT NULL DEFAULT nextval('offices\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

email character varying COLLATE pg\_catalog."default",

phone character varying COLLATE pg\_catalog."default",

city\_id integer,

street character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT offices\_pkey PRIMARY KEY (id),

CONSTRAINT offices\_city\_id\_fkey FOREIGN KEY (city\_id)

REFERENCES public.cities (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.audiences

(

id integer NOT NULL DEFAULT nextval('audiences\_id\_seq'::regclass),

area numeric(8,2),

seats\_count integer,

office\_id integer,

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

num character varying COLLATE pg\_catalog."default",

CONSTRAINT audiences\_pkey PRIMARY KEY (id),

CONSTRAINT audiences\_office\_id\_fkey FOREIGN KEY (office\_id)

REFERENCES public.offices (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.payment\_methods

(

id integer NOT NULL DEFAULT nextval('payment\_methods\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT payment\_methods\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.payment\_statuses

(

id integer NOT NULL DEFAULT nextval('payment\_statuses\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT payment\_statuses\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.requirements

(

id integer NOT NULL DEFAULT nextval('requirements\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT requirements\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.schedules

(

id integer NOT NULL DEFAULT nextval('schedules\_id\_seq'::regclass),

start\_time time without time zone,

end\_time time without time zone,

day\_id integer,

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT schedules\_pkey PRIMARY KEY (id),

CONSTRAINT schedules\_day\_id\_fkey FOREIGN KEY (day\_id)

REFERENCES public.days\_of\_week (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.softwares

(

id integer NOT NULL DEFAULT nextval('softwares\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT softwares\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.students

(

id integer NOT NULL DEFAULT nextval('students\_id\_seq'::regclass),

name character varying(100) COLLATE pg\_catalog."default",

email character varying COLLATE pg\_catalog."default",

birthday date,

surname character varying COLLATE pg\_catalog."default",

phone character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT students\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.teachers

(

id integer NOT NULL DEFAULT nextval('teachers\_id\_seq'::regclass),

name character varying(150) COLLATE pg\_catalog."default",

surname character varying(150) COLLATE pg\_catalog."default",

birthday date,

email character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT teachers\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.technologies

(

id integer NOT NULL DEFAULT nextval('technologies\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT technologies\_pkey PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS public.courses

(

id integer NOT NULL DEFAULT nextval('courses\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

description character varying COLLATE pg\_catalog."default",

price numeric(10,2),

category\_id integer,

duration integer,

difficulty\_level\_id integer,

lesson\_count integer,

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

language\_id integer,

CONSTRAINT courses\_pkey PRIMARY KEY (id),

CONSTRAINT courses\_category\_id\_fkey FOREIGN KEY (category\_id)

REFERENCES public.categories (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT courses\_difficulty\_level\_id\_fkey FOREIGN KEY (difficulty\_level\_id)

REFERENCES public.difficulty\_levels (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT courses\_language\_id\_fkey FOREIGN KEY (language\_id)

REFERENCES public.languages (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.course\_chapters

(

id integer NOT NULL DEFAULT nextval('course\_chapters\_id\_seq'::regclass),

name character varying(150) COLLATE pg\_catalog."default",

course\_id integer,

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT course\_chapters\_pkey PRIMARY KEY (id),

CONSTRAINT course\_chapters\_course\_id\_fkey FOREIGN KEY (course\_id)

REFERENCES public.courses (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.course\_requirements

(

course\_id integer,

requirement\_id integer,

CONSTRAINT course\_requirements\_course\_id\_fkey FOREIGN KEY (course\_id)

REFERENCES public.courses (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT course\_requirements\_requirement\_id\_fkey FOREIGN KEY (requirement\_id)

REFERENCES public.requirements (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.course\_schedules

(

course\_id integer,

schedule\_id integer,

CONSTRAINT course\_schedules\_course\_id\_fkey FOREIGN KEY (course\_id)

REFERENCES public.courses (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT course\_schedules\_schedule\_id\_fkey FOREIGN KEY (schedule\_id)

REFERENCES public.schedules (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.course\_softwares

(

course\_id integer,

software\_id integer,

CONSTRAINT course\_softwares\_course\_id\_fkey FOREIGN KEY (course\_id)

REFERENCES public.courses (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT course\_softwares\_software\_id\_fkey FOREIGN KEY (software\_id)

REFERENCES public.softwares (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.course\_teachers

(

id integer NOT NULL DEFAULT nextval('course\_teachers\_id\_seq'::regclass),

course\_id integer,

teacher\_id integer,

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT course\_teachers\_pkey PRIMARY KEY (id),

CONSTRAINT course\_teachers\_course\_id\_fkey FOREIGN KEY (course\_id)

REFERENCES public.courses (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT course\_teachers\_teacher\_id\_fkey FOREIGN KEY (teacher\_id)

REFERENCES public.teachers (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.course\_technologies

(

course\_id integer,

technology\_id integer,

CONSTRAINT course\_technologies\_course\_id\_fkey FOREIGN KEY (course\_id)

REFERENCES public.courses (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT course\_technologies\_technology\_id\_fkey FOREIGN KEY (technology\_id)

REFERENCES public.technologies (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.lessons

(

id integer NOT NULL DEFAULT nextval('lessons\_id\_seq'::regclass),

name character varying COLLATE pg\_catalog."default",

description character varying COLLATE pg\_catalog."default",

lesson\_type\_id integer,

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

course\_chapter\_id integer,

CONSTRAINT lessons\_pkey PRIMARY KEY (id),

CONSTRAINT lessons\_course\_chapter\_id\_fkey FOREIGN KEY (course\_chapter\_id)

REFERENCES public.course\_chapters (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT lessons\_lesson\_type\_id\_fkey FOREIGN KEY (lesson\_type\_id)

REFERENCES public.lesson\_types (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.groups

(

id integer NOT NULL DEFAULT nextval('office\_courses\_id\_seq'::regclass),

start\_date date,

end\_date date,

max\_student\_count integer,

audience\_id integer,

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

course\_teacher\_id integer,

sale\_end\_date date,

name character varying COLLATE pg\_catalog."default",

CONSTRAINT office\_courses\_pkey PRIMARY KEY (id),

CONSTRAINT groups\_audience\_id\_fkey FOREIGN KEY (audience\_id)

REFERENCES public.audiences (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT groups\_course\_teacher\_id\_fkey FOREIGN KEY (course\_teacher\_id)

REFERENCES public.course\_teachers (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.completed\_lessons

(

id integer NOT NULL DEFAULT nextval('completed\_lessons\_id\_seq'::regclass),

group\_id integer,

teacher\_id integer,

lesson\_id integer,

lesson\_date date,

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT completed\_lessons\_pkey PRIMARY KEY (id),

CONSTRAINT completed\_lessons\_group\_id\_fkey FOREIGN KEY (group\_id)

REFERENCES public.groups (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT completed\_lessons\_lesson\_id\_fkey FOREIGN KEY (lesson\_id)

REFERENCES public.lessons (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT completed\_lessons\_teacher\_id\_fkey FOREIGN KEY (teacher\_id)

REFERENCES public.teachers (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.student\_contracts

(

id integer NOT NULL DEFAULT nextval('applications\_id\_seq'::regclass),

student\_id integer,

contract\_date timestamp without time zone,

group\_id integer,

contract\_num character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT applications\_pkey PRIMARY KEY (id),

CONSTRAINT applications\_student\_id\_fkey FOREIGN KEY (student\_id)

REFERENCES public.students (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT group\_contracts\_group\_id\_fkey FOREIGN KEY (group\_id)

REFERENCES public.groups (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.student\_contract\_payments

(

id integer NOT NULL DEFAULT nextval('payments\_id\_seq'::regclass),

payment\_time timestamp without time zone,

amount numeric,

payment\_method\_id integer,

payment\_status\_id integer,

contract\_id integer,

payment\_num character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT payments\_pkey PRIMARY KEY (id),

CONSTRAINT payments\_contract\_id\_fkey FOREIGN KEY (contract\_id)

REFERENCES public.student\_contracts (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT payments\_payment\_method\_id\_fkey FOREIGN KEY (payment\_method\_id)

REFERENCES public.payment\_methods (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT payments\_payment\_status\_id\_fkey FOREIGN KEY (payment\_status\_id)

REFERENCES public.payment\_statuses (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.teacher\_contracts

(

id integer NOT NULL DEFAULT nextval('teacher\_contracts\_id\_seq'::regclass),

teacher\_id integer,

start\_date date,

end\_date date,

lesson\_cost numeric,

min\_lesson\_count integer,

withdrawal numeric,

contract\_num character varying COLLATE pg\_catalog."default",

contract\_date date,

description character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

CONSTRAINT teacher\_contracts\_pkey PRIMARY KEY (id),

CONSTRAINT teacher\_contracts\_teacher\_id\_fkey FOREIGN KEY (teacher\_id)

REFERENCES public.teachers (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.teacher\_contract\_payments

(

id integer NOT NULL DEFAULT nextval('teacher\_contract\_payments\_id\_seq'::regclass),

contract\_id integer,

total\_sum numeric(10,2),

payment\_datetime timestamp without time zone,

payment\_num character varying COLLATE pg\_catalog."default",

create\_date timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

update\_date timestamp with time zone,

payment\_period\_start date,

payment\_period\_end date,

payment\_method\_id integer,

payment\_status\_id integer,

CONSTRAINT teacher\_contract\_payments\_pkey PRIMARY KEY (id),

CONSTRAINT teacher\_contract\_payments\_contract\_id\_fkey FOREIGN KEY (contract\_id)

REFERENCES public.teacher\_contracts (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT teacher\_contract\_payments\_payment\_method\_id\_fkey FOREIGN KEY (payment\_method\_id)

REFERENCES public.payment\_methods (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT teacher\_contract\_payments\_payment\_status\_id\_fkey FOREIGN KEY (payment\_status\_id)

REFERENCES public.payment\_statuses (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

**Додаток Б. Скрипт завантаження історичних даних у базу даних**

COPY days\_of\_week(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\days\_of\_week.csv'

WITH (FORMAT csv, HEADER true);

COPY categories(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\categories.csv'

WITH (FORMAT csv, HEADER true);

COPY languages(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\languages.csv'

WITH (FORMAT csv, HEADER true);

COPY difficulty\_levels(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\difficulty\_levels.csv'

WITH (FORMAT csv, HEADER true);

COPY requirements(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\requirements.csv'

WITH (FORMAT csv, HEADER true);

COPY softwares(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\softwares.csv'

WITH (FORMAT csv, HEADER true);

COPY technologies(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\technologies.csv'

WITH (FORMAT csv, HEADER true);

COPY countries(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\countries.csv'

WITH (FORMAT csv, HEADER true);

COPY lesson\_types(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\lesson\_types.csv'

WITH (FORMAT csv, HEADER true);

COPY payment\_methods(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\payment\_methods.csv'

WITH (FORMAT csv, HEADER true);

COPY payment\_statuses(name)

FROM 'D:\Study\6 term\DW\Lab2\datasets\payment\_statuses.csv'

WITH (FORMAT csv, HEADER true);

COPY cities(name,country\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\cities.csv'

WITH (FORMAT csv, HEADER true);

COPY offices(name,email,phone,street,city\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\offices.csv'

WITH (FORMAT csv, HEADER true);

COPY audiences(num,area,seats\_count,office\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\audiences.csv'

WITH (FORMAT csv, HEADER true);

COPY teachers(name,surname,email,birthday)

FROM 'D:\Study\6 term\DW\Lab2\datasets\teachers.csv'

WITH (FORMAT csv, HEADER true);

COPY students(name,surname,email,birthday,phone)

FROM 'D:\Study\6 term\DW\Lab2\datasets\students.csv'

WITH (FORMAT csv, HEADER true);

COPY courses(name,description,price,category\_id,duration,difficulty\_level\_id,lesson\_count,language\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\courses.csv'

WITH (FORMAT csv,HEADER true);

COPY schedules(start\_time,end\_time,day\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\schedules.csv'

WITH (FORMAT csv, HEADER true);

COPY course\_chapters(name,course\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\course\_chapters.csv'

WITH (FORMAT csv, HEADER true);

COPY course\_requirements(course\_id,requirement\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\course\_requirements.csv'

WITH (FORMAT csv, HEADER true);

COPY course\_schedules(course\_id,schedule\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\course\_schedules.csv'

WITH (FORMAT csv, HEADER true);

COPY course\_softwares(course\_id,software\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\course\_softwares.csv'

WITH (FORMAT csv, HEADER true);

COPY course\_technologies(course\_id,technology\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\course\_technologies.csv'

WITH (FORMAT csv, HEADER true);

COPY course\_teachers(course\_id,teacher\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\course\_teachers.csv'

WITH (FORMAT csv, HEADER true);

COPY lessons(name,description,lesson\_type\_id,course\_chapter\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\lessons.csv'

WITH (FORMAT csv,HEADER true);

COPY groups(name,start\_date,end\_date,max\_student\_count,audience\_id,course\_teacher\_id,sale\_end\_date)

FROM 'D:\Study\6 term\DW\Lab2\datasets\groups.csv'

WITH (FORMAT csv, HEADER true);

COPY completed\_lessons(group\_id,teacher\_id,lesson\_id,lesson\_date)

FROM 'D:\Study\6 term\DW\Lab2\datasets\completed\_lessons.csv'

WITH (FORMAT csv, HEADER true);

COPY student\_contracts(contract\_num,student\_id,contract\_date,group\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\student\_contracts.csv'

WITH (FORMAT csv, HEADER true);

COPY student\_contract\_payments(payment\_num,payment\_time,payment\_method\_id,payment\_status\_id,contract\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\student\_contract\_payments.csv'

WITH (FORMAT csv, HEADER true);

COPY teacher\_contracts(contract\_num,teacher\_id,start\_date,end\_date,lesson\_cost,min\_lesson\_count,withdrawal,contract\_date,description)

FROM 'D:\Study\6 term\DW\Lab2\datasets\teacher\_contracts.csv'

WITH (FORMAT csv, HEADER true);

COPY teacher\_contract\_payments(payment\_num,payment\_datetime,contract\_id, payment\_period\_start,payment\_period\_end,payment\_method\_id,payment\_status\_id)

FROM 'D:\Study\6 term\DW\Lab2\datasets\teacher\_contract\_payments.csv'

WITH (FORMAT csv, HEADER true);

UPDATE student\_contract\_payments

SET amount = (

SELECT courses.price

FROM student\_contracts

JOIN groups ON groups.id = student\_contracts.group\_id

JOIN course\_teachers ON groups.course\_teacher\_id = course\_teachers.id

JOIN courses ON courses.id = course\_teachers.course\_id

WHERE student\_contracts.id = student\_contract\_payments.contract\_id

LIMIT 1

)

WHERE EXISTS (

SELECT 1

FROM student\_contracts

JOIN groups ON groups.id = student\_contracts.group\_id

JOIN course\_teachers ON groups.course\_teacher\_id = course\_teachers.id

JOIN courses ON courses.id = course\_teachers.course\_id

WHERE student\_contracts.id = student\_contract\_payments.contract\_id

);

UPDATE teacher\_contract\_payments

SET total\_sum = (

SELECT

CASE

WHEN COUNT(completed\_lessons.id) < min\_lesson\_count THEN COUNT(completed\_lessons.id)\*lesson\_cost - withdrawal

ELSE COUNT(completed\_lessons.id)\*lesson\_cost

END

FROM completed\_lessons

JOIN teacher\_contracts ON completed\_lessons.teacher\_id = teacher\_contracts.teacher\_id

WHERE completed\_lessons.lesson\_date BETWEEN teacher\_contract\_payments.payment\_period\_start AND teacher\_contract\_payments.payment\_period\_end

AND completed\_lessons.teacher\_id = teacher\_contracts.teacher\_id

AND teacher\_contract\_payments.contract\_id = teacher\_contracts.id

GROUP BY teacher\_contracts.teacher\_id,lesson\_cost,min\_lesson\_count,withdrawal

);

**Додаток В. Скрипт створення сховища даних**

CREATE DATABASE "CompCoursesDW"

WITH

OWNER = postgres

ENCODING = 'UTF8'

LC\_COLLATE = 'Ukrainian\_Ukraine.1251'

LC\_CTYPE = 'Ukrainian\_Ukraine.1251'

LOCALE\_PROVIDER = 'libc'

TABLESPACE = pg\_default

CONNECTION LIMIT = -1

IS\_TEMPLATE = False;

CREATE TABLE IF NOT EXISTS public.coursedim

(

course\_id integer NOT NULL DEFAULT nextval('coursedim\_course\_id\_seq'::regclass),

bk\_course\_id integer,

price numeric,

category\_name character varying COLLATE pg\_catalog."default",

difficulty\_level\_name character varying COLLATE pg\_catalog."default",

language\_name character varying COLLATE pg\_catalog."default",

name character varying COLLATE pg\_catalog."default",

CONSTRAINT coursedim\_pkey PRIMARY KEY (course\_id)

);

CREATE TABLE IF NOT EXISTS public.datedim

(

date\_id integer NOT NULL DEFAULT nextval('datedim\_date\_id\_seq'::regclass),

date\_year integer,

date\_month integer,

date\_day integer,

CONSTRAINT datedim\_pkey PRIMARY KEY (date\_id)

);

CREATE TABLE IF NOT EXISTS public.officedim

(

office\_id integer NOT NULL DEFAULT nextval('officedim\_office\_id\_seq'::regclass),

bk\_office\_id integer,

name character varying COLLATE pg\_catalog."default",

city\_name character varying COLLATE pg\_catalog."default",

country\_name character varying COLLATE pg\_catalog."default",

street character varying COLLATE pg\_catalog."default",

CONSTRAINT officedim\_pkey PRIMARY KEY (office\_id)

);

CREATE TABLE IF NOT EXISTS public.paymentmethoddim

(

payment\_method\_id integer NOT NULL DEFAULT nextval('paymentmethoddim\_payment\_method\_id\_seq'::regclass),

bk\_payment\_method\_id integer,

name character varying COLLATE pg\_catalog."default",

CONSTRAINT paymentmethoddim\_pkey PRIMARY KEY (payment\_method\_id)

);

CREATE TABLE IF NOT EXISTS public.teacherdim

(

teacher\_id integer NOT NULL DEFAULT nextval('teacherdim\_teacher\_id\_seq'::regclass),

bk\_teacher\_id integer,

name character varying COLLATE pg\_catalog."default",

surname character varying COLLATE pg\_catalog."default",

CONSTRAINT teacherdim\_pkey PRIMARY KEY (teacher\_id)

);

CREATE TABLE IF NOT EXISTS public.teacherdim

(

teacher\_id integer NOT NULL DEFAULT nextval('teacherdim\_teacher\_id\_seq'::regclass),

bk\_teacher\_id integer,

name character varying COLLATE pg\_catalog."default",

surname character varying COLLATE pg\_catalog."default",

CONSTRAINT teacherdim\_pkey PRIMARY KEY (teacher\_id)

);

CREATE TABLE IF NOT EXISTS public.teachercontractdim

(

teacher\_contract\_id integer NOT NULL DEFAULT nextval('teachercontractdim\_teacher\_contract\_id\_seq'::regclass),

bk\_teacher\_contract\_id integer,

teacher\_id integer,

start\_date\_id integer,

end\_date\_id integer,

lesson\_cost numeric,

min\_lesson\_count integer,

withdrawal numeric,

contract\_num character varying COLLATE pg\_catalog."default",

contract\_date\_id integer,

CONSTRAINT teachercontractdim\_pkey PRIMARY KEY (teacher\_contract\_id),

CONSTRAINT teachercontractdim\_contract\_date\_id\_fkey FOREIGN KEY (contract\_date\_id)

REFERENCES public.datedim (date\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT teachercontractdim\_end\_date\_id\_fkey FOREIGN KEY (end\_date\_id)

REFERENCES public.datedim (date\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT teachercontractdim\_start\_date\_id\_fkey FOREIGN KEY (start\_date\_id)

REFERENCES public.datedim (date\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT teachercontractdim\_teacher\_id\_fkey FOREIGN KEY (teacher\_id)

REFERENCES public.teacherdim (teacher\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.teachercontractpaymentfact

(

teacher\_contract\_payment\_id integer NOT NULL DEFAULT nextval('teachercontractpaymentfact\_teacher\_contract\_payment\_id\_seq'::regclass),

bk\_teacher\_contract\_payment\_id integer,

contract\_id integer,

total\_sum numeric,

payment\_date\_id integer,

payment\_num character varying COLLATE pg\_catalog."default",

payment\_period\_start\_date\_id integer,

payment\_period\_end\_date\_id integer,

payment\_method\_id integer,

salary\_change\_percentage numeric,

CONSTRAINT teachercontractpaymentfact\_pkey PRIMARY KEY (teacher\_contract\_payment\_id),

CONSTRAINT teachercontractpaymentfact\_contract\_id\_fkey FOREIGN KEY (contract\_id)

REFERENCES public.teachercontractdim (teacher\_contract\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT teachercontractpaymentfact\_payment\_date\_id\_fkey FOREIGN KEY (payment\_date\_id)

REFERENCES public.datedim (date\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT teachercontractpaymentfact\_payment\_method\_id\_fkey FOREIGN KEY (payment\_method\_id)

REFERENCES public.paymentmethoddim (payment\_method\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT teachercontractpaymentfact\_payment\_period\_end\_date\_id\_fkey FOREIGN KEY (payment\_period\_end\_date\_id)

REFERENCES public.datedim (date\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT teachercontractpaymentfact\_payment\_period\_start\_date\_id\_fkey FOREIGN KEY (payment\_period\_start\_date\_id)

REFERENCES public.datedim (date\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

CREATE TABLE IF NOT EXISTS public.groupcontractfact

(

group\_contract\_id integer NOT NULL DEFAULT nextval('studentcontractdim\_student\_contract\_id\_seq'::regclass),

income\_sum numeric,

student\_count integer,

same\_course\_interval integer,

group\_name character varying COLLATE pg\_catalog."default",

start\_date\_id integer,

end\_date\_id integer,

office\_id integer,

course\_id integer,

teacher\_id integer,

sale\_end\_date\_id integer,

CONSTRAINT studentcontractdim\_pkey PRIMARY KEY (group\_contract\_id),

CONSTRAINT groupcontractfact\_course\_id\_fkey FOREIGN KEY (course\_id)

REFERENCES public.coursedim (course\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT groupcontractfact\_end\_date\_id\_fkey FOREIGN KEY (end\_date\_id)

REFERENCES public.datedim (date\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT groupcontractfact\_office\_id\_fkey FOREIGN KEY (office\_id)

REFERENCES public.officedim (office\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT groupcontractfact\_sale\_end\_date\_id\_fkey FOREIGN KEY (sale\_end\_date\_id)

REFERENCES public.datedim (date\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT groupcontractfact\_start\_date\_id\_fkey FOREIGN KEY (start\_date\_id)

REFERENCES public.datedim (date\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT groupcontractfact\_teacher\_id\_fkey FOREIGN KEY (teacher\_id)

REFERENCES public.teacherdim (teacher\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

**Додаток Г. Скрипт заповнення БД метаданих**

CREATE OR REPLACE PROCEDURE metadata\_fill()

LANGUAGE plpgsql

AS $$

DECLARE

table\_record RECORD;

primary\_key\_column TEXT;

column\_record RECORD;

primary\_key\_name TEXT;

dw\_table\_id\_insert INT;

insert\_dw\_table\_id INT;

insert\_dimension\_id INT;

insert\_fact\_id INT;

insert\_dw\_attribute\_column\_id INT;

BEGIN

PERFORM dblink\_connect('CompCourses-oltp-conn', 'host=localhost user=postgres password=12345 dbname=CompCourses');

PERFORM dblink\_connect('CompCourses-olap-conn', 'host=localhost user=postgres password=12345 dbname=CompCoursesDW');

TRUNCATE TABLE

transformation,

fact\_metric,

dimension\_attributes,

fact,

dimension,

source\_column,

source\_table,

source\_db,

dw\_attribute\_column,

data\_load\_history,

dw\_table\_data\_load\_history,

dw\_table RESTART IDENTITY CASCADE;

INSERT INTO source\_db(source\_db\_name)

VALUES (

(SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT table\_catalog FROM information\_schema.tables') AS t(catalog\_name TEXT) LIMIT 1)

);

FOR table\_record IN SELECT \* FROM dblink('CompCourses-oltp-conn', 'SELECT table\_name FROM information\_schema.tables WHERE table\_schema = ''public'' AND table\_type = ''BASE TABLE''') AS t(table\_name text)

LOOP

primary\_key\_column := (SELECT column\_name FROM dblink('CompCourses-oltp-conn',

'SELECT kcu.column\_name

FROM information\_schema.table\_constraints tc

JOIN information\_schema.key\_column\_usage kcu

ON tc.constraint\_name = kcu.constraint\_name

WHERE tc.table\_name = ''' || table\_record.table\_name || '''

AND tc.constraint\_type = ''PRIMARY KEY''

LIMIT 1') AS t(column\_name TEXT));

EXECUTE 'INSERT INTO source\_table (source\_db\_id, source\_table\_name, key\_name) VALUES (1, $1, $2)'

USING table\_record.table\_name, primary\_key\_column;

END LOOP;

FOR column\_record IN SELECT \* FROM dblink('CompCourses-oltp-conn', 'SELECT table\_name, column\_name, data\_type FROM information\_schema.columns WHERE table\_schema = ''public''') AS t(table\_name text, column\_name text, data\_type text)

LOOP

EXECUTE 'INSERT INTO source\_column (source\_table\_id, source\_column\_name, data\_type) VALUES ((SELECT source\_table\_id FROM source\_table WHERE source\_table\_name = $1), $2, $3)'

USING column\_record.table\_name, column\_record.column\_name, column\_record.data\_type;

END LOOP;

FOR column\_record IN SELECT \* FROM dblink('CompCourses-olap-conn', 'SELECT table\_name FROM information\_schema.tables WHERE table\_schema = ''public''') AS t(table\_name text)

LOOP

EXECUTE 'INSERT INTO dw\_table (dw\_table\_name) VALUES ($1)'

USING column\_record.table\_name;

END LOOP;

FOR column\_record IN SELECT \* FROM dblink('CompCourses-olap-conn', 'SELECT table\_name, column\_name, data\_type FROM information\_schema.columns WHERE table\_schema = ''public''') AS t(table\_name text,column\_name text, data\_type text)

LOOP

EXECUTE 'INSERT INTO dw\_attribute\_column (dw\_attribute\_column\_name, dw\_attribute\_column\_datatype,dw\_attribute\_table\_id) VALUES ($1, $2, (SELECT dw\_table\_id FROM dw\_table WHERE dw\_table\_name = $3))'

USING column\_record.column\_name, column\_record.data\_type, column\_record.table\_name;

END LOOP;

FOR table\_record IN SELECT \* FROM dblink('CompCourses-olap-conn', 'SELECT table\_name FROM information\_schema.tables WHERE table\_schema = ''public'' AND table\_type = ''BASE TABLE'' AND table\_name LIKE ''%fact''') AS t(table\_name text)

LOOP

SELECT \* INTO primary\_key\_name FROM dblink('CompCourses-olap-conn',

'SELECT column\_name FROM information\_schema.table\_constraints tc

JOIN information\_schema.key\_column\_usage kcu ON tc.constraint\_name = kcu.constraint\_name AND tc.table\_schema = kcu.table\_schema

WHERE tc.table\_schema = ''public'' AND tc.table\_name = ''' || table\_record.table\_name || '''

AND tc.constraint\_type = ''PRIMARY KEY''

LIMIT 1') AS t(column\_name text);

SELECT dw\_table\_id INTO dw\_table\_id\_insert FROM dw\_table WHERE dw\_table\_name = table\_record.table\_name;

IF primary\_key\_name IS NOT NULL AND dw\_table\_id\_insert IS NOT NULL THEN

INSERT INTO fact(key\_name, dw\_table\_id) VALUES (primary\_key\_name, dw\_table\_id\_insert);

END IF;

END LOOP;

UPDATE fact

SET fact\_type = 'transactional'

WHERE key\_name LIKE '%payment%';

UPDATE fact

SET fact\_type = 'cumulative'

WHERE key\_name LIKE '%group%';

FOR table\_record IN SELECT \* FROM dblink('CompCourses-olap-conn', 'SELECT table\_name FROM information\_schema.tables WHERE table\_schema = ''public'' AND table\_type = ''BASE TABLE'' AND table\_name LIKE ''%dim''') AS t(table\_name text)

LOOP

SELECT \* INTO primary\_key\_name FROM dblink('CompCourses-olap-conn',

'SELECT column\_name FROM information\_schema.table\_constraints tc

JOIN information\_schema.key\_column\_usage kcu ON tc.constraint\_name = kcu.constraint\_name AND tc.table\_schema = kcu.table\_schema

WHERE tc.table\_schema = ''public'' AND tc.table\_name = ''' || table\_record.table\_name || '''

AND tc.constraint\_type = ''PRIMARY KEY''

LIMIT 1') AS t(column\_name text);

SELECT dw\_table\_id INTO insert\_dw\_table\_id FROM dw\_table WHERE dw\_table\_name = table\_record.table\_name;

IF primary\_key\_name IS NOT NULL AND insert\_dw\_table\_id IS NOT NULL THEN

INSERT INTO dimension(key\_name, dw\_table\_id,dimension\_name) VALUES (primary\_key\_name, insert\_dw\_table\_id,table\_record.table\_name);

END IF;

END LOOP;

UPDATE dimension

SET dimension\_type = 'SCD Type 1'

WHERE dimension\_name NOT LIKE '%date%';

UPDATE dimension

SET dimension\_type = 'SCD Type 2'

WHERE dimension\_name LIKE '%date%';

FOR column\_record IN SELECT \* FROM dblink('CompCourses-olap-conn', 'SELECT table\_name, column\_name FROM information\_schema.columns WHERE table\_schema = ''public'' AND table\_name LIKE ''%dim''') AS t(table\_name text, column\_name text)

LOOP

SELECT di.dimension\_id INTO insert\_dimension\_id FROM dimension AS di

JOIN dw\_table dw ON di.dw\_table\_id = dw.dw\_table\_id

WHERE dw.dw\_table\_name = column\_record.table\_name

LIMIT 1;

SELECT dw\_attribute\_column\_id INTO insert\_dw\_attribute\_column\_id FROM dw\_attribute\_column WHERE dw\_attribute\_column\_name = column\_record.column\_name LIMIT 1;

IF insert\_dimension\_id IS NOT NULL AND insert\_dw\_attribute\_column\_id IS NOT NULL THEN

INSERT INTO dimension\_attributes(dimension\_id, dw\_attribute\_column\_id) VALUES (insert\_dimension\_id, insert\_dw\_attribute\_column\_id);

END IF;

END LOOP;

FOR column\_record IN SELECT \* FROM dblink('CompCourses-olap-conn', 'SELECT table\_name, column\_name FROM information\_schema.columns WHERE table\_schema = ''public'' AND table\_name LIKE ''%fact''') AS t(table\_name text, column\_name text)

LOOP

SELECT fact\_id INTO insert\_fact\_id FROM fact AS f

JOIN dw\_table AS dw ON f.dw\_table\_id = dw.dw\_table\_id

WHERE dw.dw\_table\_name = column\_record.table\_name

LIMIT 1;

SELECT dw\_attribute\_column\_id INTO insert\_dw\_attribute\_column\_id FROM dw\_attribute\_column

LEFT JOIN source\_column ON source\_column.source\_column\_name = dw\_attribute\_column.dw\_attribute\_column\_name

WHERE dw\_attribute\_column\_name = column\_record.column\_name

AND source\_column.source\_column\_name IS NULL

AND dw\_attribute\_column\_name NOT LIKE '%id%' AND dw\_attribute\_column\_datatype NOT LIKE '%char%'

LIMIT 1;

IF insert\_fact\_id IS NOT NULL AND insert\_dw\_attribute\_column\_id IS NOT NULL THEN

INSERT INTO fact\_metric(fact\_id, dw\_attribute\_column\_id) VALUES (insert\_fact\_id, insert\_dw\_attribute\_column\_id);

END IF;

END LOOP;

UPDATE fact\_metric

SET fact\_metric\_description = 'percentage of change in teachers salary per month, is a non-additive fact, as it is not summed up by measurements'

WHERE dw\_attribute\_column\_id =

(SELECT fact\_metric.dw\_attribute\_column\_id

FROM fact\_metric

JOIN dw\_attribute\_column ON fact\_metric.dw\_attribute\_column\_id = dw\_attribute\_column.dw\_attribute\_column\_id

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%salary%'

);

UPDATE fact\_metric

SET fact\_metric\_description = 'the amount of sales revenue, is an additive fact, as it is summed across all dimensions'

WHERE dw\_attribute\_column\_id =

(SELECT fact\_metric.dw\_attribute\_column\_id

FROM fact\_metric

JOIN dw\_attribute\_column ON fact\_metric.dw\_attribute\_column\_id = dw\_attribute\_column.dw\_attribute\_column\_id

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%income%'

);

UPDATE fact\_metric

SET fact\_metric\_description = 'the number of students in a group, is an additive fact, as it is summed up across all dimensions'

WHERE dw\_attribute\_column\_id =

(SELECT fact\_metric.dw\_attribute\_column\_id

FROM fact\_metric

JOIN dw\_attribute\_column ON fact\_metric.dw\_attribute\_column\_id = dw\_attribute\_column.dw\_attribute\_column\_id

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%student\_count%'

);

UPDATE fact\_metric

SET fact\_metric\_description = 'the interval between the same course taught by the same teacher in the same office is a non-additive fact, as it is not summed across dimensions'

WHERE dw\_attribute\_column\_id =

(SELECT fact\_metric.dw\_attribute\_column\_id

FROM fact\_metric

JOIN dw\_attribute\_column ON fact\_metric.dw\_attribute\_column\_id = dw\_attribute\_column.dw\_attribute\_column\_id

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%same\_course%'

);

INSERT INTO transformation(dw\_attribute\_column\_id,source\_column\_id)

SELECT dw\_attribute\_column\_id,null AS source\_column\_id FROM fact\_metric

UNION

SELECT dw\_attribute\_column\_id,source\_column\_id

FROM dw\_attribute\_column

JOIN source\_column ON dw\_attribute\_column.dw\_attribute\_column\_name = CONCAT(source\_column.source\_column\_name,'\_id')

OR dw\_attribute\_column.dw\_attribute\_column\_name = CONCAT(source\_column.source\_column\_name,'\_date\_id')

JOIN source\_table ON source\_column.source\_table\_id = source\_table.source\_table\_id

JOIN dw\_table ON dw\_attribute\_column.dw\_attribute\_table\_id = dw\_table.dw\_table\_id

WHERE dw\_table.dw\_table\_name LIKE CONCAT('%',substr(source\_table.source\_table\_name,1,4),'%')

UNION

SELECT dw\_attribute\_column\_id,(SELECT source\_column\_id FROM source\_column

JOIN source\_table ON source\_column.source\_table\_id = source\_table.source\_table\_id

WHERE source\_column.source\_column\_name LIKE '%contract\_date%' AND source\_table.source\_table\_name LIKE '%contract%' LIMIT 1)

FROM dw\_attribute\_column

JOIN dw\_table ON dw\_attribute\_column.dw\_attribute\_table\_id = dw\_table.dw\_table\_id

WHERE dw\_table.dw\_table\_name = 'datedim' AND dw\_attribute\_column.dw\_attribute\_column\_name NOT LIKE '%id%';

UPDATE transformation

SET source\_column\_id = (

SELECT source\_column\_id

FROM source\_column

WHERE source\_column\_name LIKE '%total\_sum%'

),

transformation\_rule = 'SELECT(total\_sum - LAG(total\_sum)

OVER (PARTITION BY teacher\_contracts.teacher\_id ORDER BY payment\_datetime))

/ LAG(total\_sum+)

OVER (PARTITION BY teacher\_contracts.teacher\_id ORDER BY payment\_datetime) \* 100

AS salary\_change\_percentage

FROM teacher\_contract\_payments

JOIN teacher\_contracts

ON teacher\_contract\_payments.contract\_id = teacher\_contracts.id

JOIN date\_dim payment\_date ON

payment\_date.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_datetime)

JOIN date\_dim payment\_start\_period ON

payment\_start\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_start)

JOIN date\_dim payment\_end\_period ON

payment\_end\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_end);'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%salary\_change%'

);

UPDATE transformation

SET source\_column\_id = (

SELECT source\_column\_id

FROM source\_column

WHERE source\_column\_name LIKE '%price%'

),

transformation\_rule = 'SELECT SUM(Courses.price) AS income\_sum,

FROM student\_contracts

JOIN groups ON student\_contracts.group\_id = groups.id

JOIN course\_teachers ON groups.course\_teacher\_id = course\_teachers.id

JOIN courses ON course\_teachers.course\_id = courses.id

JOIN audiences ON groups.audience\_id = audiences.id

JOIN offices ON audiences.office\_id = offices.id

JOIN date\_dim group\_start\_date ON

group\_start\_date.date\_year = EXTRACT(YEAR FROM groups.start\_date)

AND group\_start\_date.date\_month = EXTRACT(MONTH FROM groups.start\_date)

AND group\_start\_date.date\_day = EXTRACT(DAY FROM groups.start\_date)

JOIN date\_dim group\_end\_date ON

group\_end\_date.date\_year = EXTRACT(YEAR FROM groups.end\_date)

AND group\_end\_date.date\_month = EXTRACT(MONTH FROM groups.end\_date)

AND group\_end\_date.date\_day = EXTRACT(DAY FROM groups.end\_date)

JOIN date\_dim group\_sale\_end\_date ON

group\_sale\_end\_date.date\_year = EXTRACT(YEAR FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_month = EXTRACT(MONTH FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_day = EXTRACT(DAY FROM groups.sale\_end\_date)

WHERE student\_contracts.contract\_date <= make\_date(

CAST(group\_sale\_end\_date.date\_year AS INT),

CAST(group\_sale\_end\_date.date\_month AS INT),

CAST(group\_sale\_end\_date.date\_day AS INT)

)

GROUP BY groups.name,course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id,

groups.start\_date,groups.end\_date,group\_start\_date.date\_id,group\_end\_date.date\_id,group\_sale\_end\_date.date\_id;'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%income%'

);

UPDATE transformation

SET source\_column\_id = (

SELECT source\_column\_id

FROM source\_column

JOIN source\_table ON source\_column.source\_table\_id = source\_table.source\_table\_id AND source\_table.source\_table\_name LIKE 'student\_contracts'

LIMIT 1

),

transformation\_rule = 'SELECT COUNT(student\_contracts.id) AS student\_count

FROM student\_contracts

JOIN groups ON student\_contracts.group\_id = groups.id

JOIN course\_teachers ON groups.course\_teacher\_id = course\_teachers.id

JOIN courses ON course\_teachers.course\_id = courses.id

JOIN audiences ON groups.audience\_id = audiences.id

JOIN offices ON audiences.office\_id = offices.id

JOIN date\_dim group\_start\_date ON

group\_start\_date.date\_year = EXTRACT(YEAR FROM groups.start\_date)

AND group\_start\_date.date\_month = EXTRACT(MONTH FROM groups.start\_date)

AND group\_start\_date.date\_day = EXTRACT(DAY FROM groups.start\_date)

JOIN date\_dim group\_end\_date ON

group\_end\_date.date\_year = EXTRACT(YEAR FROM groups.end\_date)

AND group\_end\_date.date\_month = EXTRACT(MONTH FROM groups.end\_date)

AND group\_end\_date.date\_day = EXTRACT(DAY FROM groups.end\_date)

JOIN date\_dim group\_sale\_end\_date ON

group\_sale\_end\_date.date\_year = EXTRACT(YEAR FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_month = EXTRACT(MONTH FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_day = EXTRACT(DAY FROM groups.sale\_end\_date)

WHERE student\_contracts.contract\_date <= make\_date(

CAST(group\_sale\_end\_date.date\_year AS INT),

CAST(group\_sale\_end\_date.date\_month AS INT),

CAST(group\_sale\_end\_date.date\_day AS INT)

)

GROUP BY groups.name,course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id,

groups.start\_date,groups.end\_date,group\_start\_date.date\_id,group\_end\_date.date\_id,group\_sale\_end\_date.date\_id;'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%student\_count%'

);

UPDATE transformation

SET source\_column\_id = (

SELECT source\_column\_id

FROM source\_column

JOIN source\_table ON source\_column.source\_table\_id = source\_table.source\_table\_id AND source\_table.source\_table\_name LIKE 'groups'

LIMIT 1

),

transformation\_rule = 'SELECT groups.start\_date - LAG(groups.end\_date) OVER

(PARTITION BY course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id ORDER BY groups.start\_date) AS same\_course\_interval,

FROM student\_contracts

JOIN groups ON student\_contracts.group\_id = groups.id

JOIN course\_teachers ON groups.course\_teacher\_id = course\_teachers.id

JOIN courses ON course\_teachers.course\_id = courses.id

JOIN audiences ON groups.audience\_id = audiences.id

JOIN offices ON audiences.office\_id = offices.id

JOIN date\_dim group\_start\_date ON

group\_start\_date.date\_year = EXTRACT(YEAR FROM groups.start\_date)

AND group\_start\_date.date\_month = EXTRACT(MONTH FROM groups.start\_date)

AND group\_start\_date.date\_day = EXTRACT(DAY FROM groups.start\_date)

JOIN date\_dim group\_end\_date ON

group\_end\_date.date\_year = EXTRACT(YEAR FROM groups.end\_date)

AND group\_end\_date.date\_month = EXTRACT(MONTH FROM groups.end\_date)

AND group\_end\_date.date\_day = EXTRACT(DAY FROM groups.end\_date)

JOIN date\_dim group\_sale\_end\_date ON

group\_sale\_end\_date.date\_year = EXTRACT(YEAR FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_month = EXTRACT(MONTH FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_day = EXTRACT(DAY FROM groups.sale\_end\_date)

WHERE student\_contracts.contract\_date <= make\_date(

CAST(group\_sale\_end\_date.date\_year AS INT),

CAST(group\_sale\_end\_date.date\_month AS INT),

CAST(group\_sale\_end\_date.date\_day AS INT)

)

GROUP BY groups.name,course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id,

groups.start\_date,groups.end\_date,group\_start\_date.date\_id,group\_end\_date.date\_id,group\_sale\_end\_date.date\_id;'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%same\_course%'

);

UPDATE transformation

SET transformation\_rule = 'SELECT group\_sale\_end\_date.date\_id AS sale\_end\_date\_id

FROM student\_contracts

JOIN groups ON student\_contracts.group\_id = groups.id

JOIN course\_teachers ON groups.course\_teacher\_id = course\_teachers.id

JOIN courses ON course\_teachers.course\_id = courses.id

JOIN audiences ON groups.audience\_id = audiences.id

JOIN offices ON audiences.office\_id = offices.id

JOIN date\_dim group\_start\_date ON

group\_start\_date.date\_year = EXTRACT(YEAR FROM groups.start\_date)

AND group\_start\_date.date\_month = EXTRACT(MONTH FROM groups.start\_date)

AND group\_start\_date.date\_day = EXTRACT(DAY FROM groups.start\_date)

JOIN date\_dim group\_end\_date ON

group\_end\_date.date\_year = EXTRACT(YEAR FROM groups.end\_date)

AND group\_end\_date.date\_month = EXTRACT(MONTH FROM groups.end\_date)

AND group\_end\_date.date\_day = EXTRACT(DAY FROM groups.end\_date)

JOIN date\_dim group\_sale\_end\_date ON

group\_sale\_end\_date.date\_year = EXTRACT(YEAR FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_month = EXTRACT(MONTH FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_day = EXTRACT(DAY FROM groups.sale\_end\_date)

WHERE student\_contracts.contract\_date <= make\_date(

CAST(group\_sale\_end\_date.date\_year AS INT),

CAST(group\_sale\_end\_date.date\_month AS INT),

CAST(group\_sale\_end\_date.date\_day AS INT)

)

GROUP BY groups.name,course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id,

groups.start\_date,groups.end\_date,group\_start\_date.date\_id,group\_end\_date.date\_id,group\_sale\_end\_date.date\_id;'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%sale\_end\_date%'

);

UPDATE transformation

SET transformation\_rule = 'SELECT group\_end\_date.date\_id AS end\_date\_id,

FROM student\_contracts

JOIN groups ON student\_contracts.group\_id = groups.id

JOIN course\_teachers ON groups.course\_teacher\_id = course\_teachers.id

JOIN courses ON course\_teachers.course\_id = courses.id

JOIN audiences ON groups.audience\_id = audiences.id

JOIN offices ON audiences.office\_id = offices.id

JOIN date\_dim group\_start\_date ON

group\_start\_date.date\_year = EXTRACT(YEAR FROM groups.start\_date)

AND group\_start\_date.date\_month = EXTRACT(MONTH FROM groups.start\_date)

AND group\_start\_date.date\_day = EXTRACT(DAY FROM groups.start\_date)

JOIN date\_dim group\_end\_date ON

group\_end\_date.date\_year = EXTRACT(YEAR FROM groups.end\_date)

AND group\_end\_date.date\_month = EXTRACT(MONTH FROM groups.end\_date)

AND group\_end\_date.date\_day = EXTRACT(DAY FROM groups.end\_date)

JOIN date\_dim group\_sale\_end\_date ON

group\_sale\_end\_date.date\_year = EXTRACT(YEAR FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_month = EXTRACT(MONTH FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_day = EXTRACT(DAY FROM groups.sale\_end\_date)

WHERE student\_contracts.contract\_date <= make\_date(

CAST(group\_sale\_end\_date.date\_year AS INT),

CAST(group\_sale\_end\_date.date\_month AS INT),

CAST(group\_sale\_end\_date.date\_day AS INT)

)

GROUP BY groups.name,course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id,

groups.start\_date,groups.end\_date,group\_start\_date.date\_id,group\_end\_date.date\_id,group\_sale\_end\_date.date\_id;'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

JOIN dw\_table ON dw\_table.dw\_table\_id = dw\_attribute\_column.dw\_attribute\_table\_id

WHERE dw\_attribute\_column.dw\_attribute\_column\_name = 'end\_date\_id' AND dw\_table.dw\_table\_name LIKE '%group%'

);

UPDATE transformation

SET transformation\_rule = 'SELECT group\_start\_date.date\_id AS start\_date\_id,

FROM student\_contracts

JOIN groups ON student\_contracts.group\_id = groups.id

JOIN course\_teachers ON groups.course\_teacher\_id = course\_teachers.id

JOIN courses ON course\_teachers.course\_id = courses.id

JOIN audiences ON groups.audience\_id = audiences.id

JOIN offices ON audiences.office\_id = offices.id

JOIN date\_dim group\_start\_date ON

group\_start\_date.date\_year = EXTRACT(YEAR FROM groups.start\_date)

AND group\_start\_date.date\_month = EXTRACT(MONTH FROM groups.start\_date)

AND group\_start\_date.date\_day = EXTRACT(DAY FROM groups.start\_date)

JOIN date\_dim group\_end\_date ON

group\_end\_date.date\_year = EXTRACT(YEAR FROM groups.end\_date)

AND group\_end\_date.date\_month = EXTRACT(MONTH FROM groups.end\_date)

AND group\_end\_date.date\_day = EXTRACT(DAY FROM groups.end\_date)

JOIN date\_dim group\_sale\_end\_date ON

group\_sale\_end\_date.date\_year = EXTRACT(YEAR FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_month = EXTRACT(MONTH FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_day = EXTRACT(DAY FROM groups.sale\_end\_date)

WHERE student\_contracts.contract\_date <= make\_date(

CAST(group\_sale\_end\_date.date\_year AS INT),

CAST(group\_sale\_end\_date.date\_month AS INT),

CAST(group\_sale\_end\_date.date\_day AS INT)

)

GROUP BY groups.name,course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id,

groups.start\_date,groups.end\_date,group\_start\_date.date\_id,group\_end\_date.date\_id,group\_sale\_end\_date.date\_id;'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

JOIN dw\_table ON dw\_table.dw\_table\_id = dw\_attribute\_column.dw\_attribute\_table\_id

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%start\_date\_id%' AND dw\_table.dw\_table\_name LIKE '%group%'

);

UPDATE transformation

SET transformation\_rule = 'SELECT payment\_end\_period.date\_id AS payment\_period\_end\_id

FROM teacher\_contract\_payments

JOIN teacher\_contracts

ON teacher\_contract\_payments.contract\_id = teacher\_contracts.id

JOIN date\_dim payment\_date ON

payment\_date.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_datetime)

JOIN date\_dim payment\_start\_period ON

payment\_start\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_start)

JOIN date\_dim payment\_end\_period ON

payment\_end\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_end)'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%payment\_period\_end%'

);

UPDATE transformation

SET transformation\_rule = 'SELECT payment\_start\_period.date\_id AS payment\_period\_start\_id

FROM teacher\_contract\_payments

JOIN teacher\_contracts

ON teacher\_contract\_payments.contract\_id = teacher\_contracts.id

JOIN date\_dim payment\_date ON

payment\_date.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_datetime)

JOIN date\_dim payment\_start\_period ON

payment\_start\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_start)

JOIN date\_dim payment\_end\_period ON

payment\_end\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_end)'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%payment\_period\_start%'

);

UPDATE transformation

SET transformation\_rule = 'SELECT payment\_date.date\_id AS payment\_date\_id

FROM teacher\_contract\_payments

JOIN teacher\_contracts

ON teacher\_contract\_payments.contract\_id = teacher\_contracts.id

JOIN date\_dim payment\_date ON

payment\_date.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_datetime)

JOIN date\_dim payment\_start\_period ON

payment\_start\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_start)

JOIN date\_dim payment\_end\_period ON

payment\_end\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_end)'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%payment\_date%'

);

UPDATE transformation

SET transformation\_rule = 'SELECT contract\_end\_date.date\_id AS end\_date\_id

FROM teacher\_contracts

JOIN date\_dim contract\_start\_date ON

contract\_start\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.start\_date)

AND contract\_start\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.start\_date)

AND contract\_start\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.start\_date)

JOIN date\_dim contract\_end\_date ON

contract\_end\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.end\_date)

AND contract\_end\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.end\_date)

AND contract\_end\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.end\_date)

JOIN date\_dim contract\_date ON

contract\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.contract\_date)

AND contract\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.contract\_date)

AND contract\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.contract\_date);'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

JOIN dw\_table ON dw\_table.dw\_table\_id = dw\_attribute\_column.dw\_attribute\_table\_id

WHERE dw\_attribute\_column.dw\_attribute\_column\_name = 'end\_date\_id' AND dw\_table.dw\_table\_name LIKE '%teachercontract%'

);

UPDATE transformation

SET transformation\_rule = 'SELECT contract\_start\_date.date\_id AS start\_date\_id

FROM teacher\_contracts

JOIN date\_dim contract\_start\_date ON

contract\_start\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.start\_date)

AND contract\_start\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.start\_date)

AND contract\_start\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.start\_date)

JOIN date\_dim contract\_end\_date ON

contract\_end\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.end\_date)

AND contract\_end\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.end\_date)

AND contract\_end\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.end\_date)

JOIN date\_dim contract\_date ON

contract\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.contract\_date)

AND contract\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.contract\_date)

AND contract\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.contract\_date);'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

JOIN dw\_table ON dw\_table.dw\_table\_id = dw\_attribute\_column.dw\_attribute\_table\_id

WHERE dw\_attribute\_column.dw\_attribute\_column\_name = 'start\_date\_id' AND dw\_table.dw\_table\_name LIKE '%teachercontract%'

);

UPDATE transformation

SET transformation\_rule = 'SELECT contract\_date.date\_id AS contract\_date\_id

FROM teacher\_contracts

JOIN date\_dim contract\_start\_date ON

contract\_start\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.start\_date)

AND contract\_start\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.start\_date)

AND contract\_start\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.start\_date)

JOIN date\_dim contract\_end\_date ON

contract\_end\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.end\_date)

AND contract\_end\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.end\_date)

AND contract\_end\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.end\_date)

JOIN date\_dim contract\_date ON

contract\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.contract\_date)

AND contract\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.contract\_date)

AND contract\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.contract\_date);'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name LIKE '%contract\_date%'

);

UPDATE transformation

SET transformation\_rule = 'SELECT DISTINCT EXTRACT(YEAR FROM t.chosen\_date) AS date\_year FROM

(SELECT start\_date AS chosen\_date FROM Group

UNION SELECT end\_date FROM Group

UNION SELECT sale\_end\_date FROM Group

UNION SELECT start\_date FROM TeacherContract

UNION SELECT end\_date FROM TeacherContract

UNION SELECT contract\_date FROM TeacherContract

UNION SELECT date(payment\_datetime) FROM TeacherContractPayment

UNION SELECT payment\_start\_date FROM TeacherContractPayment

UNION SELECT payment\_end\_date FROM TeacherContractPayment) AS t;'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name = 'date\_year'

);

UPDATE transformation

SET transformation\_rule = 'SELECT DISTINCT EXTRACT(MONTH FROM t.chosen\_date) AS date\_month FROM

(SELECT start\_date AS chosen\_date FROM Group

UNION SELECT end\_date FROM Group

UNION SELECT sale\_end\_date FROM Group

UNION SELECT start\_date FROM TeacherContract

UNION SELECT end\_date FROM TeacherContract

UNION SELECT contract\_date FROM TeacherContract

UNION SELECT date(payment\_datetime) FROM TeacherContractPayment

UNION SELECT payment\_start\_date FROM TeacherContractPayment

UNION SELECT payment\_end\_date FROM TeacherContractPayment) AS t;'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name = 'date\_month'

);

UPDATE transformation

SET transformation\_rule = 'SELECT DISTINCT EXTRACT(DAY FROM t.chosen\_date) AS date\_day FROM

(SELECT start\_date AS chosen\_date FROM Group

UNION SELECT end\_date FROM Group

UNION SELECT sale\_end\_date FROM Group

UNION SELECT start\_date FROM TeacherContract

UNION SELECT end\_date FROM TeacherContract

UNION SELECT contract\_date FROM TeacherContract

UNION SELECT date(payment\_datetime) FROM TeacherContractPayment

UNION SELECT payment\_start\_date FROM TeacherContractPayment

UNION SELECT payment\_end\_date FROM TeacherContractPayment) AS t;'

WHERE dw\_attribute\_column\_id = (

SELECT dw\_attribute\_column\_id

FROM dw\_attribute\_column

WHERE dw\_attribute\_column.dw\_attribute\_column\_name = 'date\_day'

);

PERFORM dblink\_disconnect('CompCourses-oltp-conn');

PERFORM dblink\_disconnect('CompCourses-olap-conn');

END;

$$;

**Додаток Д. Скрипт первинної заливки даних у сховище даних**

CREATE EXTENSION IF NOT EXISTS dblink;

CREATE PROCEDURE oltp\_to\_staging\_primary()

LANGUAGE SQL

AS $$

SELECT dblink\_connect('CompCourses-oltp-conn','host=localhost user=postgres password=12345 dbname=CompCourses');

TRUNCATE audiences,categories,cities,countries,course\_teachers,courses,difficulty\_levels, groups,languages,offices,payment\_methods,student\_contracts,teacher\_contract\_payments,

teacher\_contracts,teachers;

INSERT INTO categories(id,name)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,name FROM categories')

AS category\_columns(id integer,name varchar);

INSERT INTO audiences(id,office\_id)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,office\_id FROM audiences')

AS audience\_columns(id integer,office\_id integer);

INSERT INTO cities(id,name,country\_id)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,name,country\_id FROM cities')

AS city\_columns(id integer,name varchar,country\_id integer);

INSERT INTO countries(id,name)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,name FROM countries')

AS country\_columns(id integer,name varchar);

INSERT INTO course\_teachers(id,course\_id,teacher\_id)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,course\_id,teacher\_id FROM course\_teachers')

AS course\_teacher\_columns(id integer,course\_id integer,teacher\_id integer);

INSERT INTO courses(id,name,price,category\_id,difficulty\_level\_id,language\_id)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,name,price,category\_id,difficulty\_level\_id,language\_id FROM courses')

AS course\_columns(id integer,name varchar,price numeric(10,2),category\_id integer,difficulty\_level\_id integer,language\_id integer);

INSERT INTO difficulty\_levels(id,name)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,name FROM difficulty\_levels')

AS difficulty\_level\_columns(id integer,name varchar);

INSERT INTO languages(id,name)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,name FROM languages')

AS language\_columns(id integer,name varchar);

INSERT INTO payment\_methods(id,name)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,name FROM payment\_methods')

AS payment\_method\_columns(id integer,name varchar);

INSERT INTO offices(id,name,city\_id,street)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,name,city\_id,street FROM offices')

AS office\_columns(id integer,name varchar,city\_id integer,street varchar);

INSERT INTO teachers(id,name,surname)

SELECT \* FROM dblink('CompCourses-oltp-conn','SELECT id,name,surname FROM teachers')

AS teacher\_columns(id integer,name varchar,surname varchar);

INSERT INTO teacher\_contracts(id,teacher\_id,start\_date,end\_date,lesson\_cost,min\_lesson\_count,withdrawal,contract\_num,contract\_date)

SELECT \* FROM dblink('CompCourses-oltp-conn',

'SELECT id,teacher\_id,start\_date,end\_date,lesson\_cost,min\_lesson\_count,withdrawal,contract\_num,contract\_date

FROM teacher\_contracts')

AS teacher\_contract\_columns(id integer,teacher\_id integer,start\_date date,end\_date date,lesson\_cost numeric(10,2),

min\_lesson\_count integer,withdrawal numeric(10,2),contract\_num varchar,contract\_date date);

INSERT INTO teacher\_contract\_payments(id,contract\_id,total\_sum,payment\_datetime,payment\_num,payment\_period\_start,payment\_period\_end,payment\_method\_id)

SELECT \* FROM dblink('CompCourses-oltp-conn',

'SELECT id,contract\_id,total\_sum,payment\_datetime,payment\_num,payment\_period\_start,payment\_period\_end,payment\_method\_id

FROM teacher\_contract\_payments')

AS teache\_contract\_payment\_columns(id integer,contract\_id integer,total\_sum numeric(10,2),payment\_datetime timestamp with time zone,

payment\_num varchar,payment\_period\_start date,payment\_period\_end date,payment\_method\_id integer);

INSERT INTO groups(id,start\_date,end\_date,audience\_id,course\_teacher\_id,sale\_end\_date,name)

SELECT \* FROM dblink('CompCourses-oltp-conn',

'SELECT id,start\_date,end\_date,audience\_id,course\_teacher\_id,sale\_end\_date,name

FROM groups WHERE sale\_end\_date < CURRENT\_DATE')

AS group\_columns(id integer,start\_date date,end\_date date,audience\_id integer,course\_teacher\_id integer,sale\_end\_date date,name varchar);

INSERT INTO student\_contracts(id,contract\_date,group\_id)

SELECT \* FROM dblink('CompCourses-oltp-conn',

'SELECT student\_contracts.id,contract\_date,group\_id FROM student\_contracts

JOIN groups ON student\_contracts.group\_id = groups.id

WHERE contract\_date < CURRENT\_DATE AND sale\_end\_date < CURRENT\_DATE')

AS student\_contract\_columns(id integer,contract\_date date,group\_id integer);

SELECT dblink\_disconnect('CompCourses-oltp-conn');

$$;

CREATE EXTENSION IF NOT EXISTS dblink;

CREATE OR REPLACE PROCEDURE staging\_to\_olap\_primary()

LANGUAGE plpgsql

AS $$

DECLARE

meta\_start\_time TIMESTAMP := clock\_timestamp();

db\_table\_count INT := 0;

dw\_table\_count INT := 0;

row\_count BIGINT := 0;

temp\_count INT := 0;

insert\_statements TEXT[] := ARRAY[]::TEXT[];

data\_load\_history\_id INT;

generated\_sql TEXT;

last\_id\_result RECORD;

BEGIN

PERFORM dblink\_connect('CompCourses-staging','host=localhost user=postgres password=12345 dbname=CompCoursesStaging');

PERFORM dblink\_connect('CompCourses-metadata','host=localhost user=postgres password=12345 dbname=CompCoursesMetadata');

TRUNCATE coursedim,datedim,groupcontractfact,officedim,

paymentmethoddim,teachercontractdim,teachercontractpaymentfact,teacherdim RESTART IDENTITY CASCADE;

INSERT INTO teacherdim(bk\_teacher\_id,name,surname)

SELECT \* FROM dblink('CompCourses-staging',

'SELECT id,name,surname FROM teachers')

AS teacher\_columns(bk\_teacher\_id integer,name varchar,surname varchar);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''teacherdim''');

END IF;

INSERT INTO paymentmethoddim(bk\_payment\_method\_id,name)

SELECT \* FROM dblink('CompCourses-staging',

'SELECT id,name FROM payment\_methods')

AS pm\_columns(bk\_payment\_method\_id integer,name varchar);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''paymentmethoddim''');

END IF;

INSERT INTO officedim(bk\_office\_id,name,city\_name,country\_name,street)

SELECT \* FROM dblink('CompCourses-staging',

'SELECT offices.id,offices.name AS name,cities.name AS city\_name,countries.name AS country\_name,

offices.street

FROM offices

JOIN cities ON offices.city\_id = cities.id

JOIN countries ON cities.country\_id = countries.id')

AS office\_columns(bk\_office\_id integer,name varchar,city\_name varchar,country\_name varchar,street varchar);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 3;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''officedim''');

END IF;

INSERT INTO coursedim(bk\_course\_id,price,category\_name,difficulty\_level\_name,language\_name,name)

SELECT \* FROM dblink('CompCourses-staging',

'SELECT courses.id,price,categories.name AS category\_name,

difficulty\_levels.name AS difficulty\_level\_name,

languages.name AS language\_name,courses.name AS name

FROM courses

JOIN categories ON courses.category\_id = categories.id

JOIN difficulty\_levels ON difficulty\_levels.id = courses.difficulty\_level\_id

JOIN languages ON languages.id = courses.language\_id')

AS course\_columns(bk\_course\_id integer,price numeric(10,2),category\_name varchar,difficulty\_level\_name varchar,

language\_name varchar,name varchar);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 4;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''coursedim''');

END IF;

INSERT INTO datedim(date\_year,date\_month,date\_day)

SELECT \* FROM dblink('CompCourses-staging',

'SELECT DISTINCT EXTRACT(YEAR FROM t.chosen\_date) AS date\_year,

EXTRACT(MONTH FROM t.chosen\_date) AS date\_month,

EXTRACT(DAY FROM t.chosen\_date) AS date\_day

FROM

(SELECT start\_date AS chosen\_date FROM groups

UNION SELECT end\_date FROM groups

UNION SELECT sale\_end\_date FROM groups

UNION SELECT start\_date FROM teacher\_contracts

UNION SELECT end\_date FROM teacher\_contracts

UNION SELECT contract\_date FROM teacher\_contracts

UNION SELECT date(payment\_datetime) FROM teacher\_contract\_payments

UNION SELECT payment\_period\_start FROM teacher\_contract\_payments

UNION SELECT payment\_period\_end FROM teacher\_contract\_payments) AS t')

AS date\_columns(date\_year integer,date\_month integer,date\_day integer);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 3;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''datedim''');

END IF;

INSERT INTO teachercontractdim(bk\_teacher\_contract\_id,teacher\_id,start\_date\_id,end\_date\_id,

lesson\_cost,min\_lesson\_count,withdrawal,contract\_num,contract\_date\_id)

SELECT teacher\_contracts.id,

teacher\_contracts.teacher\_id,

contract\_start\_date.date\_id AS start\_date\_id,

contract\_end\_date.date\_id AS end\_date\_id,

teacher\_contracts.lesson\_cost,

teacher\_contracts.min\_lesson\_count,

teacher\_contracts.withdrawal,

teacher\_contracts.contract\_num,

contract\_date.date\_id AS contract\_date\_id

FROM dblink('CompCourses-staging',

'SELECT \* FROM teacher\_contracts')

AS teacher\_contracts(id integer,teacher\_id integer,start\_date date,end\_date date,lesson\_cost numeric(10,2),

min\_lesson\_count integer,withdrawal numeric(10,2),contract\_num varchar,contract\_date date)

JOIN datedim contract\_start\_date ON

contract\_start\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.start\_date)

AND contract\_start\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.start\_date)

AND contract\_start\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.start\_date)

JOIN datedim contract\_end\_date ON

contract\_end\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.end\_date)

AND contract\_end\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.end\_date)

AND contract\_end\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.end\_date)

JOIN datedim contract\_date ON

contract\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.contract\_date)

AND contract\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.contract\_date)

AND contract\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.contract\_date);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''teachercontractdim''');

END IF;

INSERT INTO teachercontractpaymentfact(bk\_teacher\_contract\_payment\_id,contract\_id,total\_sum,payment\_date\_id,

payment\_num,payment\_period\_start\_date\_id,payment\_period\_end\_date\_id,

payment\_method\_id,salary\_change\_percentage)

SELECT teacher\_contract\_payments.id,

teacher\_contract\_payments.contract\_id,

teacher\_contract\_payments.total\_sum,

payment\_date.date\_id AS payment\_date\_id,

teacher\_contract\_payments.payment\_num,

payment\_start\_period.date\_id AS payment\_period\_start\_date\_id,

payment\_end\_period.date\_id AS payment\_period\_end\_date\_id,

teacher\_contract\_payments.payment\_method\_id,

(total\_sum + 1 - LAG(total\_sum+1) OVER (PARTITION BY teacher\_contracts.teacher\_id ORDER BY payment\_datetime))

/ LAG(total\_sum+1) OVER (PARTITION BY teacher\_contracts.teacher\_id ORDER BY payment\_datetime) \* 100

AS salary\_change\_percentage

FROM dblink('CompCourses-staging',

'SELECT \* FROM teacher\_contract\_payments')

AS teacher\_contract\_payments(id integer,contract\_id integer,total\_sum numeric(10,2),payment\_datetime timestamp with time zone,

payment\_num varchar,payment\_period\_start date,payment\_period\_end date,payment\_method\_id integer)

JOIN

dblink('CompCourses-staging',

'SELECT \* FROM teacher\_contracts')

AS teacher\_contracts(id integer,teacher\_id integer,start\_date date,end\_date date,lesson\_cost numeric(10,2),

min\_lesson\_count integer,withdrawal numeric(10,2),contract\_num varchar,contract\_date date)

ON teacher\_contracts.id = teacher\_contract\_payments.contract\_id

JOIN datedim payment\_date ON

payment\_date.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_datetime)

JOIN datedim payment\_start\_period ON

payment\_start\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_start)

JOIN datedim payment\_end\_period ON

payment\_end\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_end);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''teachercontractpaymentfact''');

END IF;

INSERT INTO groupcontractfact(income\_sum,student\_count,same\_course\_interval,group\_name,start\_date\_id,

end\_date\_id,office\_id,course\_id,teacher\_id,sale\_end\_date\_id)

SELECT SUM(Courses.price) AS income\_sum,

COUNT(student\_contracts.id) AS student\_count,

groups.start\_date - LAG(groups.end\_date) OVER

(PARTITION BY course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id ORDER BY groups.start\_date)

AS same\_course\_interval,

groups.name AS group\_name,

group\_start\_date.date\_id AS start\_date\_id,

group\_end\_date.date\_id AS end\_date\_id,

offices.id AS office\_id,

course\_teachers.course\_id AS course\_id,

course\_teachers.teacher\_id AS teacher\_id,

group\_sale\_end\_date.date\_id AS sale\_end\_date\_id

FROM dblink('CompCourses-staging',

'SELECT \* FROM student\_contracts')

AS student\_contracts(id integer,contract\_date date,group\_id integer)

JOIN dblink('CompCourses-staging',

'SELECT \* FROM groups')

AS groups(id integer,start\_date date,end\_date date,audience\_id integer,course\_teacher\_id integer,sale\_end\_date date,name varchar)

ON student\_contracts.group\_id = groups.id

JOIN dblink('CompCourses-staging',

'SELECT \* FROM course\_teachers')

AS course\_teachers(id integer,course\_id integer,teacher\_id integer)

ON groups.course\_teacher\_id = course\_teachers.id

JOIN dblink('CompCourses-staging',

'SELECT id,price FROM courses')

AS courses(id integer,price numeric(10,2))

ON course\_teachers.course\_id = courses.id

JOIN dblink('CompCourses-staging',

'SELECT \* FROM audiences')

AS audiences(id integer,office\_id integer)

ON groups.audience\_id = audiences.id

JOIN dblink('CompCourses-staging',

'SELECT \* FROM offices')

AS offices(id integer,name varchar,city\_id integer,street varchar)

ON audiences.office\_id = offices.id

JOIN datedim group\_start\_date ON

group\_start\_date.date\_year = EXTRACT(YEAR FROM groups.start\_date)

AND group\_start\_date.date\_month = EXTRACT(MONTH FROM groups.start\_date)

AND group\_start\_date.date\_day = EXTRACT(DAY FROM groups.start\_date)

JOIN datedim group\_end\_date ON

group\_end\_date.date\_year = EXTRACT(YEAR FROM groups.end\_date)

AND group\_end\_date.date\_month = EXTRACT(MONTH FROM groups.end\_date)

AND group\_end\_date.date\_day = EXTRACT(DAY FROM groups.end\_date)

JOIN datedim group\_sale\_end\_date ON

group\_sale\_end\_date.date\_year = EXTRACT(YEAR FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_month = EXTRACT(MONTH FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_day = EXTRACT(DAY FROM groups.sale\_end\_date)

WHERE student\_contracts.contract\_date <= make\_date(

CAST(group\_sale\_end\_date.date\_year AS INT),

CAST(group\_sale\_end\_date.date\_month AS INT),

CAST(group\_sale\_end\_date.date\_day AS INT)

)

GROUP BY groups.name,course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id,

groups.start\_date,groups.end\_date,group\_start\_date.date\_id,group\_end\_date.date\_id,group\_sale\_end\_date.date\_id;

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''groupcontractfact''');

END IF;

PERFORM dblink\_exec('CompCourses-metadata',

format('INSERT INTO data\_load\_history(load\_datetime, load\_time, load\_rows, affected\_table\_count, source\_table\_count) VALUES (%L, EXTRACT(EPOCH FROM %L::timestamp - %L::timestamp), %L, %L, %L)',

clock\_timestamp(), clock\_timestamp(), meta\_start\_time, row\_count, dw\_table\_count, db\_table\_count));

SELECT \* INTO last\_id\_result FROM dblink('CompCourses-metadata', 'SELECT MAX(data\_load\_history\_id) as last\_id FROM data\_load\_history') AS t(last\_id INT);

data\_load\_history\_id := last\_id\_result.last\_id;

FOREACH generated\_sql IN ARRAY insert\_statements

LOOP

PERFORM dblink\_exec('CompCourses-metadata', format(generated\_sql, data\_load\_history\_id));

END LOOP;

PERFORM dblink\_exec('CompCourses-staging',

'TRUNCATE audiences,categories,cities,countries,course\_teachers,courses,difficulty\_levels,

groups,languages,offices,payment\_methods,student\_contracts,teacher\_contract\_payments,

teacher\_contracts,teachers');

PERFORM dblink\_disconnect('CompCourses-staging');

PERFORM dblink\_disconnect('CompCourses-metadata');

END;

$$;

**Додаток Е. Скрипт інкрементальної заливки у сховище даних**

CREATE EXTENSION IF NOT EXISTS dblink;

CREATE PROCEDURE oltp\_to\_staging\_inc()

LANGUAGE plpgsql

AS $$

DECLARE last\_load\_date timestamp;

BEGIN

PERFORM dblink\_connect('CompCourses-oltp-conn','host=localhost user=postgres password=12345 dbname=CompCourses');

PERFORM dblink\_connect('CompCourses-metadata-conn','host=localhost user=postgres password=12345 dbname=CompCoursesMetadata');

last\_load\_date:= (SELECT load\_date FROM dblink('CompCourses-metadata-conn','SELECT MAX(load\_datetime) FROM data\_load\_history') AS last\_date\_columns(load\_date timestamp));

TRUNCATE audiences,categories,cities,countries,course\_teachers,courses,difficulty\_levels, groups,languages,offices,payment\_methods,student\_contracts,teacher\_contract\_payments,

teacher\_contracts,teachers;

INSERT INTO categories(id,name)

SELECT id,name FROM dblink('CompCourses-oltp-conn','SELECT id,name,create\_date,update\_date FROM categories')

AS category\_columns(id integer,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT c.id,c.name

FROM dblink('CompCourses-oltp-conn','SELECT id,name,create\_date,update\_date FROM categories') AS c(id integer,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn','SELECT id,category\_id,create\_date,update\_date FROM courses') AS courses\_columns(id integer,category\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone) ON courses\_columns.category\_id = c.id

WHERE courses\_columns.create\_date > last\_load\_date OR courses\_columns.update\_date > last\_load\_date;

INSERT INTO audiences(id,office\_id)

SELECT id,office\_id FROM dblink('CompCourses-oltp-conn','SELECT id,office\_id,create\_date,update\_date FROM audiences')

AS audience\_columns(id integer,office\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT a.id,a.office\_id

FROM dblink('CompCourses-oltp-conn','SELECT id,office\_id,create\_date,update\_date FROM audiences') AS a(id integer,office\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn',

'SELECT id,audience\_id,sale\_end\_date,create\_date,update\_date

FROM groups')

AS group\_columns(id integer,audience\_id integer,sale\_end\_date date,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON a.id = group\_columns.audience\_id

WHERE group\_columns.sale\_end\_date BETWEEN last\_load\_date AND CURRENT\_DATE AND (group\_columns.create\_date > last\_load\_date OR group\_columns.update\_date > last\_load\_date);

INSERT INTO cities(id,name,country\_id)

SELECT id,name,country\_id FROM dblink('CompCourses-oltp-conn','SELECT id,name,country\_id,create\_date,update\_date FROM cities')

AS city\_columns(id integer,name varchar,country\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT c.id,c.name,c.country\_id

FROM dblink('CompCourses-oltp-conn','SELECT id,name,country\_id,create\_date,update\_date FROM cities') AS c (id integer,name varchar,country\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn','SELECT id,city\_id,create\_date,update\_date FROM offices') AS offices\_columns(id integer,city\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON offices\_columns.city\_id = c.id

WHERE offices\_columns.create\_date > last\_load\_date OR offices\_columns.update\_date > last\_load\_date;

INSERT INTO countries(id,name)

SELECT id,name FROM dblink('CompCourses-oltp-conn','SELECT id,name,create\_date,update\_date FROM countries')

AS country\_columns(id integer,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT c.id,c.name

FROM dblink('CompCourses-oltp-conn','SELECT id,name,create\_date,update\_date FROM countries') AS c (id integer,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn','SELECT id,country\_id,create\_date,update\_date FROM cities') AS city\_columns(id integer,country\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON city\_columns.country\_id = c.id

WHERE city\_columns.create\_date > last\_load\_date OR city\_columns.update\_date > last\_load\_date;

INSERT INTO course\_teachers(id,course\_id,teacher\_id)

SELECT id,course\_id,teacher\_id FROM dblink('CompCourses-oltp-conn','SELECT id,course\_id,teacher\_id,create\_date,update\_date FROM course\_teachers')

AS course\_teacher\_columns(id integer,course\_id integer,teacher\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT ct.id,ct.course\_id,ct.teacher\_id

FROM dblink('CompCourses-oltp-conn','SELECT id,course\_id,teacher\_id,create\_date,update\_date FROM course\_teachers')

AS ct(id integer,course\_id integer,teacher\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn',

'SELECT id,course\_teacher\_id,sale\_end\_date,create\_date,update\_date

FROM groups')

AS group\_columns(id integer,course\_teacher\_id integer,sale\_end\_date date,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON ct.id = group\_columns.course\_teacher\_id

WHERE group\_columns.sale\_end\_date BETWEEN last\_load\_date AND CURRENT\_DATE AND (group\_columns.create\_date > last\_load\_date OR group\_columns.update\_date > last\_load\_date);

INSERT INTO courses(id,name,price,category\_id,difficulty\_level\_id,language\_id)

SELECT id,name,price,category\_id,difficulty\_level\_id,language\_id FROM dblink('CompCourses-oltp-conn','SELECT id,name,price,category\_id,difficulty\_level\_id,language\_id,create\_date,update\_date FROM courses')

AS course\_columns(id integer,name varchar,price numeric(10,2),category\_id integer,difficulty\_level\_id integer,language\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT c.id,c.name,c.price,c.category\_id,c.difficulty\_level\_id,c.language\_id FROM dblink('CompCourses-oltp-conn','SELECT id,name,price,category\_id,difficulty\_level\_id,language\_id,create\_date,update\_date FROM courses')

AS c(id integer,name varchar,price numeric(10,2),category\_id integer,difficulty\_level\_id integer,language\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn','SELECT id,course\_id,create\_date,update\_date FROM course\_teachers')

AS course\_teacher\_columns(id integer,course\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON course\_teacher\_columns.course\_id = c.id

WHERE course\_teacher\_columns.create\_date > last\_load\_date OR course\_teacher\_columns.update\_date > last\_load\_date;

INSERT INTO difficulty\_levels(id,name)

SELECT id,name FROM dblink('CompCourses-oltp-conn','SELECT id,name,create\_date,update\_date FROM difficulty\_levels')

AS difficulty\_level\_columns(id integer,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT dl.id,dl.name FROM dblink('CompCourses-oltp-conn','SELECT id,name,create\_date,update\_date FROM difficulty\_levels') AS dl(id integer,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn','SELECT id,difficulty\_level\_id,create\_date,update\_date FROM courses') AS courses\_columns(id integer,difficulty\_level\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone) ON courses\_columns.difficulty\_level\_id = dl.id

WHERE courses\_columns.create\_date > last\_load\_date OR courses\_columns.update\_date > last\_load\_date;

INSERT INTO languages(id,name)

SELECT id,name FROM dblink('CompCourses-oltp-conn','SELECT id,name,create\_date,update\_date FROM languages')

AS language\_columns(id integer,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT l.id,l.name FROM dblink('CompCourses-oltp-conn','SELECT id,name,create\_date,update\_date FROM languages')

AS l(id integer,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn','SELECT id,language\_id,create\_date,update\_date FROM courses') AS courses\_columns(id integer,language\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone) ON courses\_columns.language\_id = l.id

WHERE courses\_columns.create\_date > last\_load\_date OR courses\_columns.update\_date > last\_load\_date;

INSERT INTO payment\_methods(id,name)

SELECT id,name FROM dblink('CompCourses-oltp-conn','SELECT id,name,create\_date,update\_date FROM payment\_methods')

AS payment\_method\_columns(id integer,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT pm.id,pm.name FROM dblink('CompCourses-oltp-conn','SELECT id,name,create\_date,update\_date FROM payment\_methods')

AS pm(id integer,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn',

'SELECT id,payment\_method\_id,create\_date,update\_date

FROM teacher\_contract\_payments')

AS teacher\_contract\_payment\_columns(id integer,payment\_method\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON pm.id = teacher\_contract\_payment\_columns.payment\_method\_id

WHERE teacher\_contract\_payment\_columns.create\_date > last\_load\_date OR teacher\_contract\_payment\_columns.update\_date > last\_load\_date;

INSERT INTO offices(id,name,city\_id,street)

SELECT id,name,city\_id,street FROM dblink('CompCourses-oltp-conn','SELECT id,name,city\_id,street,create\_date,update\_date FROM offices')

AS office\_columns(id integer,name varchar,city\_id integer,street varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT o.id,o.name,o.city\_id,o.street

FROM dblink('CompCourses-oltp-conn','SELECT id,name,city\_id,street,create\_date,update\_date FROM offices')

AS o(id integer,name varchar,city\_id integer,street varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn','SELECT id,office\_id,create\_date,update\_date FROM audiences')

AS audience\_columns(id integer,office\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON o.id = audience\_columns.office\_id

WHERE audience\_columns.create\_date > last\_load\_date OR audience\_columns.update\_date > last\_load\_date;

INSERT INTO teachers(id,name,surname)

SELECT id,name,surname FROM dblink('CompCourses-oltp-conn','SELECT id,name,surname,create\_date,update\_date FROM teachers')

AS teacher\_columns(id integer,name varchar,surname varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT t.id,t.name,t.surname FROM dblink('CompCourses-oltp-conn','SELECT id,name,surname,create\_date,update\_date FROM teachers')

AS t(id integer,name varchar,surname varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn','SELECT id,teacher\_id,create\_date,update\_date FROM course\_teachers')

AS course\_teacher\_columns(id integer,teacher\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON t.id = course\_teacher\_columns.teacher\_id

WHERE course\_teacher\_columns.create\_date > last\_load\_date OR course\_teacher\_columns.update\_date > last\_load\_date

UNION

SELECT t.id,t.name,t.surname FROM dblink('CompCourses-oltp-conn','SELECT id,name,surname,create\_date,update\_date FROM teachers')

AS t(id integer,name varchar,surname varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn',

'SELECT id,teacher\_id,create\_date,update\_date

FROM teacher\_contracts')

AS teacher\_contract\_columns(id integer,teacher\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON t.id = teacher\_contract\_columns.teacher\_id

WHERE teacher\_contract\_columns.create\_date > last\_load\_date OR teacher\_contract\_columns.update\_date > last\_load\_date;

INSERT INTO teacher\_contracts(id,teacher\_id,start\_date,end\_date,lesson\_cost,min\_lesson\_count,withdrawal,contract\_num,contract\_date)

SELECT id,teacher\_id,start\_date,end\_date,lesson\_cost,min\_lesson\_count,withdrawal,contract\_num,contract\_date FROM dblink('CompCourses-oltp-conn',

'SELECT id,teacher\_id,start\_date,end\_date,lesson\_cost,min\_lesson\_count,withdrawal,contract\_num,contract\_date,create\_date,update\_date

FROM teacher\_contracts')

AS teacher\_contract\_columns(id integer,teacher\_id integer,start\_date date,end\_date date,lesson\_cost numeric(10,2),

min\_lesson\_count integer,withdrawal numeric(10,2),contract\_num varchar,contract\_date date,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date

UNION

SELECT tc.id,tc.teacher\_id,tc.start\_date,tc.end\_date,tc.lesson\_cost,tc.min\_lesson\_count,tc.withdrawal,tc.contract\_num,tc.contract\_date FROM dblink('CompCourses-oltp-conn',

'SELECT id,teacher\_id,start\_date,end\_date,lesson\_cost,min\_lesson\_count,withdrawal,contract\_num,contract\_date,create\_date,update\_date

FROM teacher\_contracts')

AS tc(id integer,teacher\_id integer,start\_date date,end\_date date,lesson\_cost numeric(10,2),

min\_lesson\_count integer,withdrawal numeric(10,2),contract\_num varchar,contract\_date date,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn',

'SELECT id,contract\_id,create\_date,update\_date

FROM teacher\_contract\_payments')

AS teacher\_contract\_payment\_columns(id integer,contract\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON tc.id = teacher\_contract\_payment\_columns.contract\_id

WHERE teacher\_contract\_payment\_columns.create\_date > last\_load\_date OR teacher\_contract\_payment\_columns.update\_date > last\_load\_date;

INSERT INTO teacher\_contract\_payments(id,contract\_id,total\_sum,payment\_datetime,payment\_num,payment\_period\_start,payment\_period\_end,payment\_method\_id)

SELECT id,contract\_id,total\_sum,payment\_datetime,payment\_num,payment\_period\_start,payment\_period\_end,payment\_method\_id FROM dblink('CompCourses-oltp-conn',

'SELECT id,contract\_id,total\_sum,payment\_datetime,payment\_num,payment\_period\_start,payment\_period\_end,payment\_method\_id,create\_date,update\_date

FROM teacher\_contract\_payments')

AS teacher\_contract\_payment\_columns(id integer,contract\_id integer,total\_sum numeric(10,2),payment\_datetime timestamp with time zone,

payment\_num varchar,payment\_period\_start date,payment\_period\_end date,payment\_method\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE create\_date > last\_load\_date OR update\_date > last\_load\_date;

INSERT INTO groups(id,start\_date,end\_date,audience\_id,course\_teacher\_id,sale\_end\_date,name)

SELECT id,start\_date,end\_date,audience\_id,course\_teacher\_id,sale\_end\_date,name FROM dblink('CompCourses-oltp-conn',

'SELECT id,start\_date,end\_date,audience\_id,course\_teacher\_id,sale\_end\_date,name,create\_date,update\_date

FROM groups')

AS group\_columns(id integer,start\_date date,end\_date date,audience\_id integer,course\_teacher\_id integer,sale\_end\_date date,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

WHERE group\_columns.sale\_end\_date BETWEEN last\_load\_date AND CURRENT\_DATE AND (group\_columns.create\_date > last\_load\_date OR group\_columns.update\_date > last\_load\_date)

UNION

SELECT g.id,g.start\_date,g.end\_date,g.audience\_id,g.course\_teacher\_id,g.sale\_end\_date,g.name FROM dblink('CompCourses-oltp-conn',

'SELECT id,start\_date,end\_date,audience\_id,course\_teacher\_id,sale\_end\_date,name,create\_date,update\_date

FROM groups')

AS g(id integer,start\_date date,end\_date date,audience\_id integer,course\_teacher\_id integer,sale\_end\_date date,name varchar,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn',

'SELECT id,contract\_date,group\_id,create\_date,update\_date FROM student\_contracts')

AS student\_contract\_columns(id integer,contract\_date date,group\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON student\_contract\_columns.group\_id = g.id

WHERE (student\_contract\_columns.contract\_date BETWEEN last\_load\_date AND CURRENT\_DATE AND g.sale\_end\_date BETWEEN last\_load\_date AND CURRENT\_DATE) AND ((student\_contract\_columns.create\_date > last\_load\_date OR student\_contract\_columns.update\_date > last\_load\_date) AND (g.create\_date > last\_load\_date OR g.update\_date > last\_load\_date));

INSERT INTO student\_contracts(id,contract\_date,group\_id)

SELECT student\_contract\_columns.id,contract\_date,group\_id FROM dblink('CompCourses-oltp-conn',

'SELECT student\_contracts.id,contract\_date,group\_id,create\_date,update\_date FROM student\_contracts')

AS student\_contract\_columns(id integer,contract\_date date,group\_id integer,create\_date timestamp with time zone,update\_date timestamp with time zone)

JOIN dblink('CompCourses-oltp-conn',

'SELECT id,sale\_end\_date,name,create\_date,update\_date

FROM groups')

AS group\_columns(id integer,sale\_end\_date date,create\_date timestamp with time zone,update\_date timestamp with time zone)

ON student\_contract\_columns.group\_id = group\_columns.id

WHERE (student\_contract\_columns.contract\_date BETWEEN last\_load\_date AND CURRENT\_DATE AND group\_columns.sale\_end\_date BETWEEN last\_load\_date AND CURRENT\_DATE) AND ((student\_contract\_columns.create\_date > last\_load\_date OR student\_contract\_columns.update\_date > last\_load\_date) AND (group\_columns.create\_date > last\_load\_date OR group\_columns.update\_date > last\_load\_date));

PERFORM dblink\_disconnect('CompCourses-oltp-conn');

PERFORM dblink\_disconnect('CompCourses-metadata-conn');

END;

$$;

CREATE EXTENSION IF NOT EXISTS dblink;

CREATE OR REPLACE PROCEDURE staging\_to\_olap\_inc()

LANGUAGE plpgsql

AS $$

DECLARE

meta\_start\_time TIMESTAMP := clock\_timestamp();

db\_table\_count INT := 0;

dw\_table\_count INT := 0;

row\_count BIGINT := 0;

temp\_count INT := 0;

insert\_statements TEXT[] := ARRAY[]::TEXT[];

data\_load\_history\_id INT;

generated\_sql TEXT;

last\_id\_result RECORD;

BEGIN

PERFORM dblink\_connect('CompCourses-staging','host=localhost user=postgres password=12345 dbname=CompCoursesStaging');

PERFORM dblink\_connect('CompCourses-metadata','host=localhost user=postgres password=12345 dbname=CompCoursesMetadata');

INSERT INTO datedim(date\_year,date\_month,date\_day)

SELECT \* FROM dblink('CompCourses-staging',

'SELECT DISTINCT EXTRACT(YEAR FROM t.chosen\_date) AS date\_year,

EXTRACT(MONTH FROM t.chosen\_date) AS date\_month,

EXTRACT(DAY FROM t.chosen\_date) AS date\_day

FROM

(SELECT start\_date AS chosen\_date FROM groups

UNION SELECT end\_date FROM groups

UNION SELECT sale\_end\_date FROM groups

UNION SELECT start\_date FROM teacher\_contracts

UNION SELECT end\_date FROM teacher\_contracts

UNION SELECT contract\_date FROM teacher\_contracts

UNION SELECT date(payment\_datetime) FROM teacher\_contract\_payments

UNION SELECT payment\_period\_start FROM teacher\_contract\_payments

UNION SELECT payment\_period\_end FROM teacher\_contract\_payments) AS t')

AS date\_columns(date\_year integer,date\_month integer,date\_day integer);

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 3;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''datedim''');

END IF;

MERGE INTO paymentmethoddim target

USING (

SELECT \* FROM dblink('CompCourses-staging',

'SELECT id,name FROM payment\_methods')

AS pm\_columns(id integer,name varchar)

) AS source

ON source.id = target.bk\_payment\_method\_id

WHEN MATCHED THEN

UPDATE SET name = source.name

WHEN NOT MATCHED THEN

INSERT (name) VALUES(source.name);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''paymentmethoddim''');

END IF;

MERGE INTO teacherdim target

USING (

SELECT \* FROM dblink('CompCourses-staging',

'SELECT id,name,surname FROM teachers')

AS teacher\_columns(id integer,name varchar,surname varchar)

) AS source

ON source.id = target.bk\_teacher\_id

WHEN MATCHED THEN

UPDATE SET name = source.name,

surname = source.surname

WHEN NOT MATCHED THEN

INSERT (name,surname) VALUES(source.name,source.surname);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''teacherdim''');

END IF;

MERGE INTO officedim target

USING (

SELECT \* FROM dblink('CompCourses-staging',

'SELECT offices.id,offices.name AS name,cities.name AS city\_name,countries.name AS country\_name,

offices.street

FROM offices

JOIN cities ON offices.city\_id = cities.id

JOIN countries ON cities.country\_id = countries.id')

AS office\_columns(id integer,name varchar,city\_name varchar,country\_name varchar,street varchar)

) AS source

ON source.id = target.bk\_office\_id

WHEN MATCHED THEN

UPDATE SET name = source.name,

city\_name = source.city\_name,

country\_name = source.country\_name,

street = source.street

WHEN NOT MATCHED THEN

INSERT (name,city\_name,country\_name,street)

VALUES(source.name,source.city\_name,source.country\_name,source.street);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 3;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''officedim''');

END IF;

MERGE INTO coursedim target

USING (

SELECT \* FROM dblink('CompCourses-staging',

'SELECT courses.id,price,categories.name AS category\_name,

difficulty\_levels.name AS difficulty\_level\_name,

languages.name AS language\_name,courses.name AS name

FROM courses

JOIN categories ON courses.category\_id = categories.id

JOIN difficulty\_levels ON difficulty\_levels.id = courses.difficulty\_level\_id

JOIN languages ON languages.id = courses.language\_id')

AS course\_columns(id integer,price numeric(10,2),category\_name varchar,difficulty\_level\_name varchar, language\_name varchar,name varchar)

) AS source

ON source.id = target.bk\_course\_id

WHEN MATCHED THEN

UPDATE SET price = source.price,

category\_name = source.category\_name,

difficulty\_level\_name = source.difficulty\_level\_name,

language\_name = source.language\_name,

name = source.name

WHEN NOT MATCHED THEN

INSERT (price,category\_name,difficulty\_level\_name,language\_name,name)

VALUES (source.price,source.category\_name,source.difficulty\_level\_name,source.language\_name,source.name);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 4;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''coursedim''');

END IF;

MERGE INTO teachercontractdim target

USING (

SELECT teacher\_contracts.id,

teacher\_contracts.teacher\_id,

contract\_start\_date.date\_id AS start\_date\_id,

contract\_end\_date.date\_id AS end\_date\_id,

teacher\_contracts.lesson\_cost,

teacher\_contracts.min\_lesson\_count,

teacher\_contracts.withdrawal,

teacher\_contracts.contract\_num,

contract\_date.date\_id AS contract\_date\_id

FROM dblink('CompCourses-staging',

'SELECT \* FROM teacher\_contracts')

AS teacher\_contracts(id integer,teacher\_id integer,start\_date date,end\_date date,lesson\_cost numeric(10,2),

min\_lesson\_count integer,withdrawal numeric(10,2),contract\_num varchar,contract\_date date)

JOIN datedim contract\_start\_date ON

contract\_start\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.start\_date)

AND contract\_start\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.start\_date)

AND contract\_start\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.start\_date)

JOIN datedim contract\_end\_date ON

contract\_end\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.end\_date)

AND contract\_end\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.end\_date)

AND contract\_end\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.end\_date)

JOIN datedim contract\_date ON

contract\_date.date\_year = EXTRACT(YEAR FROM teacher\_contracts.contract\_date)

AND contract\_date.date\_month = EXTRACT(MONTH FROM teacher\_contracts.contract\_date)

AND contract\_date.date\_day = EXTRACT(DAY FROM teacher\_contracts.contract\_date)

) AS source

ON source.id = target.bk\_teacher\_contract\_id

WHEN MATCHED THEN

UPDATE SET teacher\_id = source.teacher\_id,

start\_date\_id = source.start\_date\_id,

end\_date\_id = source.end\_date\_id,

lesson\_cost = source.lesson\_cost,

min\_lesson\_count = source.min\_lesson\_count,

withdrawal = source.withdrawal,

contract\_num = source.contract\_num,

contract\_date\_id = source.contract\_date\_id

WHEN NOT MATCHED THEN

INSERT (teacher\_id,start\_date\_id,end\_date\_id,lesson\_cost,min\_lesson\_count,withdrawal,contract\_num,contract\_date\_id)

VALUES(source.teacher\_id,source.start\_date\_id,source.end\_date\_id,source.lesson\_cost,source.min\_lesson\_count,source.withdrawal,source.contract\_num,source.contract\_date\_id);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''teachercontractdim''');

END IF;

MERGE INTO teachercontractpaymentfact target

USING (

SELECT teacher\_contract\_payments.id,

teacher\_contract\_payments.contract\_id,

teacher\_contract\_payments.total\_sum,

payment\_date.date\_id AS payment\_date\_id,

teacher\_contract\_payments.payment\_num,

payment\_start\_period.date\_id AS payment\_period\_start\_date\_id,

payment\_end\_period.date\_id AS payment\_period\_end\_date\_id,

teacher\_contract\_payments.payment\_method\_id,

(total\_sum + 1 - LAG(total\_sum+1) OVER (PARTITION BY teacher\_contracts.teacher\_id ORDER BY payment\_datetime))

/ LAG(total\_sum+1) OVER (PARTITION BY teacher\_contracts.teacher\_id ORDER BY payment\_datetime) \* 100

AS salary\_change\_percentage

FROM dblink('CompCourses-staging',

'SELECT \* FROM teacher\_contract\_payments')

AS teacher\_contract\_payments(id integer,contract\_id integer,total\_sum numeric(10,2),payment\_datetime timestamp with time zone,

payment\_num varchar,payment\_period\_start date,payment\_period\_end date,payment\_method\_id integer)

JOIN

dblink('CompCourses-staging',

'SELECT \* FROM teacher\_contracts')

AS teacher\_contracts(id integer,teacher\_id integer,start\_date date,end\_date date,lesson\_cost numeric(10,2),

min\_lesson\_count integer,withdrawal numeric(10,2),contract\_num varchar,contract\_date date)

ON teacher\_contracts.id = teacher\_contract\_payments.contract\_id

JOIN datedim payment\_date ON

payment\_date.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_datetime)

AND payment\_date.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_datetime)

JOIN datedim payment\_start\_period ON

payment\_start\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_start)

AND payment\_start\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_start)

JOIN datedim payment\_end\_period ON

payment\_end\_period.date\_year = EXTRACT(YEAR FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_month = EXTRACT(MONTH FROM teacher\_contract\_payments.payment\_period\_end)

AND payment\_end\_period.date\_day = EXTRACT(DAY FROM teacher\_contract\_payments.payment\_period\_end)

) AS source

ON source.id = target.bk\_teacher\_contract\_payment\_id

WHEN MATCHED THEN

UPDATE SET contract\_id = source.contract\_id,

total\_sum = source.total\_sum,

payment\_date\_id = source.payment\_date\_id,

payment\_num = source.payment\_num,

payment\_period\_start\_date\_id = source.payment\_period\_start\_date\_id,

payment\_period\_end\_date\_id = source.payment\_period\_end\_date\_id,

payment\_method\_id = source.payment\_method\_id,

salary\_change\_percentage = source.salary\_change\_percentage

WHEN NOT MATCHED THEN

INSERT (contract\_id,total\_sum,payment\_date\_id,payment\_num,payment\_period\_start\_date\_id,payment\_period\_end\_date\_id,payment\_method\_id,

salary\_change\_percentage)

VALUES(source.contract\_id,source.total\_sum,source.payment\_date\_id,source.payment\_num,source.payment\_period\_start\_date\_id,source.payment\_period\_end\_date\_id,source.payment\_method\_id,source.salary\_change\_percentage);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''teachercontractpaymentfact''');

END IF;

MERGE INTO groupcontractfact target

USING (

SELECT SUM(Courses.price) AS income\_sum,

COUNT(student\_contracts.id) AS student\_count,

groups.start\_date - LAG(groups.end\_date) OVER

(PARTITION BY course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id ORDER BY groups.start\_date)

AS same\_course\_interval,

groups.name AS group\_name,

group\_start\_date.date\_id AS start\_date\_id,

group\_end\_date.date\_id AS end\_date\_id,

offices.id AS office\_id,

course\_teachers.course\_id AS course\_id,

course\_teachers.teacher\_id AS teacher\_id,

group\_sale\_end\_date.date\_id AS sale\_end\_date\_id

FROM dblink('CompCourses-staging',

'SELECT \* FROM student\_contracts')

AS student\_contracts(id integer,contract\_date date,group\_id integer)

JOIN dblink('CompCourses-staging',

'SELECT \* FROM groups')

AS groups(id integer,start\_date date,end\_date date,audience\_id integer,course\_teacher\_id integer,sale\_end\_date date,name varchar)

ON student\_contracts.group\_id = groups.id

JOIN dblink('CompCourses-staging',

'SELECT \* FROM course\_teachers')

AS course\_teachers(id integer,course\_id integer,teacher\_id integer)

ON groups.course\_teacher\_id = course\_teachers.id

JOIN dblink('CompCourses-staging',

'SELECT id,price FROM courses')

AS courses(id integer,price numeric(10,2))

ON course\_teachers.course\_id = courses.id

JOIN dblink('CompCourses-staging',

'SELECT \* FROM audiences')

AS audiences(id integer,office\_id integer)

ON groups.audience\_id = audiences.id

JOIN dblink('CompCourses-staging',

'SELECT \* FROM offices')

AS offices(id integer,name varchar,city\_id integer,street varchar)

ON audiences.office\_id = offices.id

JOIN datedim group\_start\_date ON

group\_start\_date.date\_year = EXTRACT(YEAR FROM groups.start\_date)

AND group\_start\_date.date\_month = EXTRACT(MONTH FROM groups.start\_date)

AND group\_start\_date.date\_day = EXTRACT(DAY FROM groups.start\_date)

JOIN datedim group\_end\_date ON

group\_end\_date.date\_year = EXTRACT(YEAR FROM groups.end\_date)

AND group\_end\_date.date\_month = EXTRACT(MONTH FROM groups.end\_date)

AND group\_end\_date.date\_day = EXTRACT(DAY FROM groups.end\_date)

JOIN datedim group\_sale\_end\_date ON

group\_sale\_end\_date.date\_year = EXTRACT(YEAR FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_month = EXTRACT(MONTH FROM groups.sale\_end\_date)

AND group\_sale\_end\_date.date\_day = EXTRACT(DAY FROM groups.sale\_end\_date)

WHERE student\_contracts.contract\_date <= make\_date(

CAST(group\_sale\_end\_date.date\_year AS INT),

CAST(group\_sale\_end\_date.date\_month AS INT),

CAST(group\_sale\_end\_date.date\_day AS INT)

)

GROUP BY groups.name,course\_teachers.course\_id,course\_teachers.teacher\_id,offices.id,

groups.start\_date,groups.end\_date,group\_start\_date.date\_id,group\_end\_date.date\_id,group\_sale\_end\_date.date\_id

) AS source

ON source.course\_id = target.course\_id

AND source.teacher\_id = target.teacher\_id

AND source.office\_id = target.office\_id

AND source.sale\_end\_date\_id = target.sale\_end\_date\_id

WHEN NOT MATCHED THEN

INSERT (income\_sum,student\_count,same\_course\_interval,group\_name,start\_date\_id,end\_date\_id,office\_id,course\_id,teacher\_id,sale\_end\_date\_id)

VALUES (source.income\_sum,source.student\_count,source.same\_course\_interval,source.group\_name,source.start\_date\_id,source.end\_date\_id,source.office\_id,source.course\_id,source.teacher\_id,source.sale\_end\_date\_id);

GET DIAGNOSTICS temp\_count = ROW\_COUNT;

IF temp\_count > 0 THEN

row\_count := row\_count + temp\_count;

dw\_table\_count := dw\_table\_count + 1;

db\_table\_count := db\_table\_count + 1;

insert\_statements := insert\_statements || format('INSERT INTO dw\_table\_data\_load\_history(dw\_table\_id, data\_load\_history\_id) SELECT dw\_table\_id, %%s FROM dw\_table WHERE dw\_table\_name=''groupcontractfact''');

END IF;

PERFORM dblink\_exec('CompCourses-metadata',

format('INSERT INTO data\_load\_history(load\_datetime, load\_time, load\_rows, affected\_table\_count, source\_table\_count) VALUES (%L, EXTRACT(EPOCH FROM %L::timestamp - %L::timestamp), %L, %L, %L)',

clock\_timestamp(), clock\_timestamp(), meta\_start\_time, row\_count, dw\_table\_count, db\_table\_count));

SELECT \* INTO last\_id\_result FROM dblink('CompCourses-metadata', 'SELECT MAX(data\_load\_history\_id) as last\_id FROM data\_load\_history') AS t(last\_id INT);

data\_load\_history\_id := last\_id\_result.last\_id;

FOREACH generated\_sql IN ARRAY insert\_statements

LOOP

PERFORM dblink\_exec('CompCourses-metadata', format(generated\_sql, data\_load\_history\_id));

END LOOP;

PERFORM dblink\_exec('CompCourses-staging',

'TRUNCATE audiences,categories,cities,countries,course\_teachers,courses,difficulty\_levels,

groups,languages,offices,payment\_methods,student\_contracts,teacher\_contract\_payments,

teacher\_contracts,teachers');

PERFORM dblink\_disconnect('CompCourses-staging');

PERFORM dblink\_disconnect('CompCourses-metadata');

END;

$$;