Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет_____Компьютерної інженерії та управління_____
(повна назва)

Кафедра_____Автоматизації проектування обчислювальної техніки _____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти_____другий(магістерський)_____

_____Система обліку та управління підприємства_____
_____з використанням Fog-технології_____
(тема)

Виконав:
студент _6_ курсу, групи__СКСм-19-2_____
_____Федота О.В._____
(прізвище, ініціали)

Спеціальність _____
_____123_Комп'ютерна інженерія_____
(код і повна назва спеціальності)
Тип програми____ освітньо-професійна ____
(освітньо-професійна або освітньо-наукова)
Освітня програма_Спеціалізовані комп'ютерні системи_____
(повна назва освітньої програми)

Керівник___проф. Немченко В.П._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ ___Чумаченко С.В_____
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий(магістерський)

Спеціальність 123 Комп'ютерна інженерія
(код і повна назва)

Тип програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри_____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Федоті Олександру Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система обліку та управління підприємства з використанням Fog-технології

затверджена наказом університету від 26 березня 2021 р. № 385Ст

2. Термін подання студентом роботи до екзаменаційної комісії ____ _____ 20__ р.

3. Вихідні дані до роботи

Мова програмування T-SQL

Мова програмування HSQL

Кластерне рішення на Hadoop Hive

Середовище Cisco Packet Tracer

СКБД SQL Server

4. Перелік питань, що потрібно опрацювати в роботі

Аналіз методів організації сховищ даних

Огляд існуючих рішень

Вибір програмного забезпечення

Розробка компонентів системи

Алгоритм роботи системи

Розробка програмного забезпечення

Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій)____14 слайдів_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

Дата видачі завдання __01__ __лютого___2021 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	01.02.2021 – 10.02.2021	
2	Аналіз предметної області	10.02.2021 – 28.02.2021	
3	Постановка задачі	28.02.2021 – 15.03.2021	
4	Розробка вимог до системи	15.03.2021 – 15.04.2021	
5	Проектування компонентів мережі	15.04.2021 – 25.04.2021	
6	Реалізація системи сховища даних	25.04.2021 – 05.05.2021	
7	Створення моделі Fog-мережі	05.05.2021 – 07.05-2021	
8	Проведення дослідження	07.05.2021 – 10.05.2021	
9	Оформлення пояснювальної записки	10.05.2021 – 14.05.2021	
10	Оформлення графічного матеріалу	14.05.2021 – 15.05.2021	
11	Перевірка виконаного проекту керівником	15.05.2021 – 17.05.2021	
12	Захист проекту	27.05.2021	

Студент _____
(підпис)

Керівник роботи _____ __проф. Немченко В.П.___
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 83с., 3 додатки, 32 рис., 1 табл., 22 джерела за переліком посилань

СХОВИЩЕ ДАНИХ, ТУМАННІ ОБЧИСЛЕННЯ, АНАЛІТИЧНА СИСТЕМА, ДАТЧИКИ, ВЕЛИКІ ДАНІ, ІНТЕРНЕТ РЕЧЕЙ, ETL, OLAP

Метою дослідницької роботи є дослідження ефективності використання сховища даних, як компонента системи обліку та управління підприємством, з використанням технологій туманних обчислень і великих даних. Система використовує датчики як одне з джерел отримання поточної інформації для первинної обробки на локальному сервері і завантаження в корпоративне сховище даних.

В результаті реалізації забезпечена функціональність системи, що може використовувати різні джерела даних, одне з яких датчики, встановлені на підприємстві. Подальша обробка даних з датчиків відбувається локально, для запобігання потрапляння неважливої у даний час інформації до аналітичної системи, після чого дані з усіх джерел, включаючи FOG-рівень, проходять обробку в єдиному середовищі і подальше завантаження в цільові сутності.

ABSTRACT

Explanatory note: 83p., 3 appendices, 32 fig., 1 table., 22 sources according to the list of references

DATA STORAGE, FOG CALCULATION, ANALYTICAL SYSTEM, SENSORS, BIG DATA, INTERNET OF THINGS, ETL, OLAP

The purpose of the research is to study the efficiency of using the data warehouse as a component of the enterprise accounting and management system, using the technologies of fog-computing and big data. The system uses sensors as one of the sources of current information for initial processing on a local server and downloading to the enterprise data warehouse.

As a result of the implementation, the functionality of the system is provided, which can use various data sources, one of which is the sensors installed at the enterprise. Further processing of data from sensors is local, to prevent the ingress of currently irrelevant information into the analytical system, after which data from all sources, including the FOG-level, are processed in a single environment and then loaded into the target entities.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

Big Data - Технології, що пов’язані з роботою з великими обсягами даних;

IoT - Internet of Things - Інтернет речей;

DWH - Data Warehouse - Сховище даних;

ETL - Extract, Transform, Load - Підхід до роботи з даними в сховищі даних;

ELT - Extract, Load, Transform - Підхід до роботи з даними в сховищі даних;

OLAP - On-line Analytical Processing - підхід до роботи з даними;

OLTP - On-line Transactional Processing - підхід до роботи з даними;

ERP - Enterprise Resource Planning - системи обліку та керування ресурсами для підприємств;

FOG-computing - Туманні обчислення;

DBMS - Database Management System - Система керування базами даних;

HDFS - Hadoop Distributed File System - Файлова система Хадуп на кластерних рішеннях;

BI - Business Intelligence;

Data mart - Вітрина даних;

ERP - Enterprise Resource Planning System - Система обліку та управління підприємством

Kerberos - протокол передачі даних через незахищені мережі, що забезпечує захищену ідентифікацію користувачів;

XML - мова размітки;

Flat File - Джерело даних, що являє собою файл з даними.

ВСТУП

Розгортання мережі з використанням Інтернету речей генерує великі обсяги даних в процесі роботи, які потребують обробки та аналізу в режимі реального часу. Сучасні системи IoT не дозволяють обробляти дані з низькою затримкою та високошвидкісною обробкою і вимагають завантаження обробки даних у хмару. Хоча хмарні обчислення оптимізують використання ресурсів, вони не забезпечують ефективного рішення для розміщення додатків великих даних. Доступ до даних із хмари передбачає великий мережевий трафік, великі затримки, як правило, непридатні для реального часу або критичних рішень, а також більші ризики безпеки. В якості альтернативи, обчислення туману постають перспективною технологією, що дозволяє запобігти цим незручностям. В них пропонується використовувати пристрої, що відносяться до ‘Fog-computing’ для забезпечення кращих обчислювальних можливостей і, отже, зменшення мережевого трафіку, різкого скорочення затримок, одночасно покращуючи безпеку.

Дослідницька робота присвячена розробці компонентів системи обліку та управління підприємством, що забезпечують туманну обробку даних, і подальше завантаження їх разом з інформацією з джерел, що не пов’язані з туманними обчисленнями в цільове сховище даних. При цьому невід’ємними компонентами такої системи є сервер для первинної обробки даних отриманих з датчиків та сховище даних. Під терміном “корпоративне сховище даних“ мається на увазі систему, що забезпечує вивантаження вихідних даних, їх очистку та обробку, приведення до необхідного вигляду, та завантаження в цільову систему.

1 АНАЛІЗ МЕТОДІВ ОРГАНІЗАЦІЇ ХМАРНИХ СИСТЕМ ОБЛІКУ ТА УПРАВЛІННЯ

1.1. Аналіз актуальності розвитку “Інтернету речей” в хмарних системах

Розвиток Big Data та досягнення технологій зробили значний перехід до високої функціональності пристроїв IoT. Популярність пристроїв IoT призвела до більш простих методів збору, аналізу та розповсюдження Великих Даних швидкими темпами. Згідно зі звітом Statista, у всьому світі буде понад 75 мільярдів пристроїв IoT до 2025 року. Існують такі методи, як квантові обчислення, хмарні обчислення, обчислення межі/туману. Незважаючи на те, що квантові обчислення мають світлу перспективу, їм потрібно пройти довгий шлях, тим часом хмарні обчислення вже є популярним аналітичним методом серед розробників та вчених з питань обробки даних.

В останні роки вже традиційно управління розумними ресурсами на підприємствах здійснюється за допомогою хмарного рішення, де датчики та пристрої підключені для забезпечення централізованого та багатого набору відкритих даних. Перевагами хмарних фреймворків є їх повсюдність, а також (майже) необмежена потужність ресурсів.

Наприкінці 2015 року ряд лідерів IoT запустили консорціум OpenFog. Мета: прискорити впровадження технологій туману шляхом розробки відкритої архітектури [1]. Партнери-засновники включали Cisco, ARM, Dell, Intel, Microsoft та Princeton University. Згідно з дослідженнями, опублікованими наприкінці жовтня 2017 року з нагоди Всесвітнього конгресу щодо туманних обчислень, до 2022 року очікується, що ринок туманних обчислень у світі перевищить 18 млрд доларів. Консорціум OpenFog, який вважає туман необхідним для IoT, 5G та вбудованого штучного інтелекту, замовив 451 дослідження, щоб глибше заглибитися в основні ринки обчислень туману та мереж, порівняти хмару та місцеві витрати та розбити ринок на різні сегменти

(обладнання, програми та сервіси окремо від туману). Найбільші очікувані ринки збуту - транспорт, промисловість, енергетика/ комунальне господарство та охорона здоров'я. Очікується, що доходи від хмарного виробництва збільшаться на 147 відсотків до 2022 року, а туманні обчислення надходитимуть на існуючі пристрої та програмне забезпечення, що працює з новими цільовими вузлами туману. Туман як послуга (FaaS) повинен подвоїти своє зростання між 2020 і 2022 роками.

Зібрані BigData мають величезні обсяги, вони часто збираються з одного однорідного джерела або різних гетерогенних джерел. Ці джерела можуть бути датчиками, такими як акустичні датчики, біосенсори, нанородільні датчики тощо. В ці великі дані вбудована корисна інформація та цінні знання. Є різні дані, які збираються і аналізуються:

а) точні дані;

б) неточні дані;

в) невизначені дані.

Для збору тез BigData використовується процес, який називається Data Mining. Видобуток даних виявляє неявні, раніше невідомі та корисні дані. Видобуток даних на пристроях IoT допомагає BigData виявити шаблони програм IoT для розумних міст, університетів та підприємств.

1.2. Дослідження актуальних проблем IoT-систем для підприємства

Під час проектування, розробки і експлуатації систем, що використовують хмарні обчислення і великі дані(Big Data) можна виділити ті, наявність яких під час проектування постає в незалежності від обраних технологій і варіанти та шляхи вирішення цих проблем є актуальними і важливими на сьогоднішній день. Серед таких питань можна виділити наступні:

а) хмара може бути фізично розташована у віддаленому центрі обробки даних, тому може не бути можливим обслуговування IoT з розумною

затримкою та пропускною здатністю;

б) переміщення великих обсягів даних по вузлах віртуалізованої обчислювальної платформи може спричинити значні накладні витрати з точки зору часу, пропускної здатності, споживання енергії та вартості;

в) сучасні хмарні рішення не мають можливості розмістити аналітичні механізми для ефективної обробки великих даних.

г) існуючі платформи розробки IoT вертикально фрагментовані. Таким чином, новатори IoT повинні переміщатися між різнорідними апаратними та програмними послугами, які не завжди добре інтегруються між собою.

Для вирішення цих проблем аналітика даних може бути виконана на етапі туманних обчислень(або Fog) - поблизу місця, де генеруються дані - для зменшення обсягу даних під час комунікацій та накладних витрат. Спочатку пропонувалося використовувати параметри обчислювальної техніки для використання своїх обчислювальних ресурсів для локального зберігання, попередньої обробки даних, щоб зменшити навантаження або перевантаження мережі, а також забезпечити локальний аналіз даних, щоб призвело до швидкого процесу прийняття рішень, засновуючись на даних. Однак, крайні обчислювальні пристрої мають дуже обмежені ресурси(рис. 1.1), що призводить до обмеження ресурсів і збільшує затримку обробки. Тому з’явилася нова парадигма, відома як Fog Computing, яка інтегрує крайні пристрої, з одного боку, із хмарними ресурсами, з іншого, щоб також подолати всі обмеження Edge Computing [2, 3].

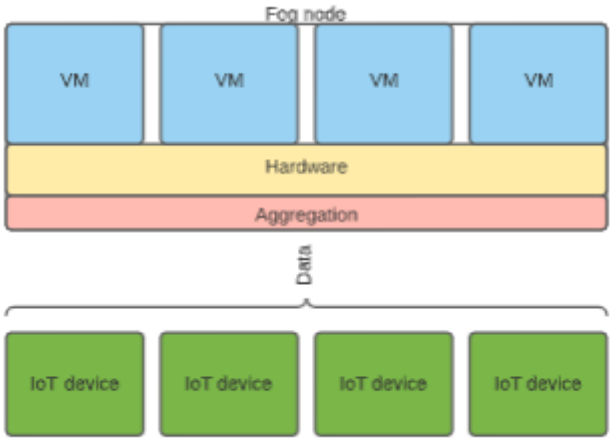

Рисунок 1.1 – структура Fog-вузла

Таким чином, обчислення туману дозволяють уникнути обмеження ресурсів на краю, використовуючи хмарні ресурси, а також координуючи географічно розподілені крайні пристрої.

1.3. Організація сучасних хмарних систем для роботи з “Великими Даними”

Більшість провідних постачальників великих даних зараз нараховують сотні клієнтів. Big Data - це вже не область Інтернету та медіа-компаній з великими веб-властивостями; компанії майже в кожній галузі працюють з великими обсягами даних. Сюди входять енергетика, фармацевтика, комунальні послуги, телекомунікації, страхування, роздрібна торгівля, фінансові послуги та уряд. Наприклад, клієнт фінансових послуг використовує Hadoop для підвищення точності своїх моделей шахрайства, працюючи з набагато більшими обсягами даних про транзакції.

Сьогодні на ринку існує 2 типи великих даних. Існує програмне забезпечення з відкритим кодом, що зосереджено переважно на Hadoop, яке виключає попередні витрати на ліцензування для управління та обробки великих обсягів даних. І тоді з’являються нові аналітичні інструменти, включаючи побутові прилади та сховища(рис. 1.2), які забезпечують значно

вищі показники, ніж реляційні бази даних загального призначення.

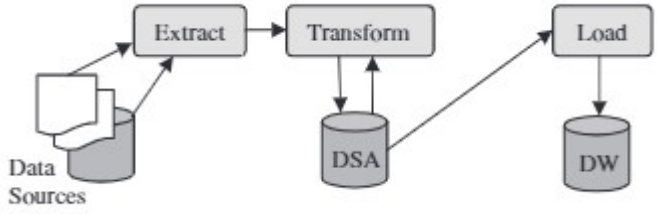

Рисунок 1.2 – Схема обробки даних в сучасному СД

Обидва набори програмного забезпечення для великих даних забезпечують більшу рентабельність інвестицій, ніж попередні покоління технологій управління даними, але дуже різними способами [4].

Hadoop - це розподілена файлова система з відкритим кодом, доступна через Apache Software Foundation, яка здатна зберігати та обробляти великі обсяги даних паралельно через сітку товарних серверів. Hadoop походить від великих Інтернет-провайдерів, таких як Google та Yahoo, яким потрібен економічний спосіб створення індексів пошуку. Вони знали, що традиційні реляційні бази даних будуть надмірно дорогими і технічно громіздкими, тому вони придумали недорогу альтернативу, яку створили самі і, врешті-решт, надали Фонду програмного забезпечення Apache, щоб інші могли отримати користь від їх нововведень. Сьогодні багато компаній впроваджують програмне забезпечення Hadoop від Apache, а також сторонніх постачальників, таких як IBM, Cloudera, Hortonworks та EMC. Розробники розглядають Hadoop як економічно ефективний спосіб обійти великі обсяги даних, з якими раніше вони ніколи не могли багато чого зробити.

Багато компаній використовують Hadoop для зберігання, обробки та аналізу великих обсягів даних журналів веб-серверів, щоб вони могли краще відчути поведінку своїх веб-клієнтів при перегляді та покупках. Раніше компанії передавали аналітичні дані своїх потоків кліків на аутсорсинг або просто дозволяли їм потрапляти до т.з. “staging area”, оскільки у них не було способу обробити їх своєчасно та ефективно. Компанії також звертаються до

Hadoop для обробки більш структурованих даних для вдосконалення аналітичних моделей.

Окрім того, що програмне забезпечення для великих даних є вільним у впровадженні, це ще й те, що воно є надзвичайно варіативним. Воно може обробляти будь-який тип даних. На відміну від сховища даних або традиційної реляційної бази даних, Hadoop не вимагає від адміністраторів моделювання чи перетворення даних перед їх завантаженням. У Hadoop ми не визначаємо структуру даних. Це значно зменшує витрати на підготовку даних до аналізу порівняно з тим, що відбувається в сховищі даних. Більшість експертів стверджують, що від 60% до 80% вартості побудови сховища даних, яка може обернутися десятками мільйонів доларів, передбачає вилучення, перетворення та завантаження даних. Hadoop практично виключає ці витрати. Як наслідок, багато компаній використовують Hadoop як загальнодоступну проміжну область та архівують всі свої дані. Отже, телекомунікаційна компанія може зберігати 12 місяців записів про деталі дзвінків, замість того щоб агрегувати ці дані у сховищі даних і переносити деталі в офлайн-сховище [5]. За допомогою Hadoop вони можуть зберігати всі свої дані в Інтернеті та елімінувати вартість архівних систем даних. Вони також можуть дозволити досвідченим користувачам запитувати дані Hadoop безпосередньо, якщо вони хочуть отримати доступ до необроблених даних або не можуть дочекатися завантаження агрегатів у сховище даних. приховані витрати. Перед запитом даних Hadoop розробник повинен зрозуміти структуру даних та всі їх аномалії. З чистим, добре зрозумілим, однорідним набором даних це не складно. Але більшість корпоративних даних не відповідають такому опис [6, 8]. Тож розробник Hadoop в кінцевому підсумку відіграє роль розробника зберігання даних під час запиту, вилучаючи дані та переконуючись, що їх формат та вміст відповідають їхнім очікуванням. Для запуску програмного забезпечення для великих даних все одно потрібно купувати, встановлювати та керувати товарними серверами (якщо ви не запускаєте середовище великих даних у хмарі, скажімо через Amazon Web Services). Хоча кожен сервер може

коштувати не так дорого, ціна зростає, але що дорожче, це досвід та програмне забезпечення, необхідні для адміністрування Hadoop та управління мережами товарних серверів.

Інший тип великих даних передує варіантам Hadoop та NoSQL на кілька років. Ця версія великих даних є меншим «рухом», ніж розширення існуючої технології реляційних баз даних, оптимізованої для обробки запитів. Ці аналітичні платформи охоплюють цілий ряд технологій, починаючи від приладів та стовпчастих баз даних і масової паралельної обробки даних. Спільною темою серед них є те, що більшість середовищ лише для читання забезпечують виняткову цінову ефективність порівняно з реляційними базами даних загального призначення, спочатку розробленими для запуску додатків для обробки транзакцій. початок 1980-х. Sybase також був першим попередником, поставивши першу стовпчасту базу даних в середині 1990-х. IBM Netezza підняла нинішній ринок у високі темпи в 2003 році, коли представила популярний аналітичний прилад, і незабаром за ним послідували десятки стартапів. Визнаючи можливість, усі великі імена програмного та апаратного забезпечення Oracle, IBM, Hewlett-Packard та SAP - згодом вийшли на ринок, або шляхом створення, або придбання технології, щоб надати цільові аналітичні системи новим та існуючим клієнтам. Хоча ціновий етикет цих систем часто перевищує 1 мільйон доларів, клієнти виявляють, що виняткова цінова ефективність забезпечує значну ділову цінність як у матеріальній, так і в нематеріальній формі. Наприклад, заснована в штаті Вірджинія компанія XO Communications повернула 3 мільйони доларів втраченого доходу за допомогою нового додатка для забезпечення доходів, який він побудував на аналітичному приладі, ще до того, як заплатив за систему. Згодом він побудував або переніс десяток заявок для роботи на новій спеціально побудованій системі, що засвідчує її цінність [10]. Однак необхідно буде здійснити розумний зв'язок між двома системами даних. З одного боку, це може бути поєднання даних із сховища даних на рівні аналітичних даних (наприклад, шляхом прив'язки потокових даних до замовника або номера контракту зі сховища даних).

Таблиця 1.1 – Класифікація платформ для роботи з даними

NoSql	**Hadoop**	**Аналітичні платформи**	**OLTP-бази даних**
Cassandra, MongoDB, Aster Data	Cloudera, IBM Hortonworks	Netezza, Vertica, Teradata	Oracle, DB2, SQL Server
Графові системи, що визначають відношення між сутностями	Архів даних переважно для неструктурованих даних	Корпоративні сховища, призначені для заміни MySQL/SQL Server у компаніях, що швидко розвиваються	Транзакційні системи
Пошукові бази даних для роботи з структурованими даними	Аналітична система, коли необхідно проаналізувати “сирі” дані		Корпоративні центри сховищ даних

Але це передбачає, що IT-компанія здійснює аналітичну обробку даних IoT. Часто використовується сценарій, в якому результати аналізу великих даних надаються зовнішнім постачальником послуг страховій компанії. У будь-якому випадку, завданням буде зв’язати аналітичні результати (наприклад, бальні значення) з інформацією, що зберігається у сховищі даних [11].

На високому рівні сьогодні є чотири категорії аналітичних систем обробки(Табл. 1.1):

а) транзакційні СКБД;

б) аналітичні платформи;

в) Hadoop системи;

г) NoSql бази даних.

Транзакційні СУБД були оригінально розроблені для підтримки програм обробки транзакцій, хоча більшість з них були модернізовані різними типами індексів і шляхами об'єднання, щоб зробити їх більш приємними для аналітичної обробки. Існує два типи транзакційних СКБД: корпоративні та відомчі [7]. Традиційні корпоративні СКБД, такі як IBM, Oracle та Sybase, найкраще підходять як концентратори для зберігання даних, які живлять різноманітні нижчі системи, спрямовані на кінцевих споживачів, але не обробляють безпосередньо трафік запитів. Незважаючи на те, що ці системи модернізовані з аналітичними можливостями, вони часто зменшують результати продуктивності та масштабованості, коли використовуються для обробки запитів разом з іншими робочими навантаженнями, і, що не менш важливо, це їх дороге оновлення та заміна. Ряд компаній використовують Microsoft SQL Server або MySQL як вітрини даних, що подаються корпоративним сховищем даних, або як самостійні сховища даних для бізнес-підрозділу або малого та середнього бізнесу (SMB). Як і їхні корпоративні брати, ці системи також часто б'ються об стіну, коли використання, обсяги даних або складність запитів швидко зростають. Швидко зростаючий бізнес-підрозділ або SMB часто замінює ці транзакційні СКБД аналітичними програмами (див. Нижче), які забезпечують такий самий або більший рівень простоти та простоти управління, як SQL Server або MySQL.

Аналітичні платформи представляють першу хвилю систем великих даних. Це спеціально побудовані системи на базі SQL, розроблені для забезпечення чудових показників ціни на аналітичні робочі навантаження порівняно з транзакційними СКБД. Існує багато типів аналітичних платформ. Більшість із них використовуються як замінники для зберігання даних або самостійні аналітичні системи. Бази даних з масовою паралельною обробкою (MPP) із потужними змішаними програмами робочого навантаження роблять хороші сховища корпоративних даних для аналітично налаштованих організацій. Teradata був першим на ринку з такою системою, але зараз у неї

багато конкурентів, включаючи EMC Greenplum та паралельне сховище даних від Microsoft, які є відносними новинками порівняно з 30-річною Teradata. Ці спеціально створені аналітичні системи є інтегрованою комбінацією апаратно-програмного забезпечення, налаштованої на аналітичні робочі навантаження. Аналітичні прилади бувають різних форм, розмірів та конфігурацій. Деякі, такі як IBM Netezza, EMC Greenplum та Oracle Exadata, є більш загальноприйнятими аналітичними машинами, які можуть служити заміною для більшості сховищ даних. Інші, такі як Teradata, орієнтовані на конкретні аналітичні робочі навантаження і можуть забезпечувати надзвичайно швидку продуктивність або керувати надвеликими обсягами даних. Якщо метою є продуктивність, немає нічого кращого, ніж система, яка дозволяє зберігати всі ваші дані в пам'яті. Ці системи незабаром стануть більш звичними завдяки SAP, який робить ставку на свою діяльність на HANA, базі даних в пам'яті для транзакційних та аналітичних обробок, і обгрунтовує потребу в системах в пам'яті(рис. 1.3). Ще одним претендентом у цьому просторі є Kognitio. Багато СКБД краще використовують пам'ять для кешування результатів та обробки запитів. Стовпчасті бази даних, такі як Sybase IQ від SAP, Vertica від Hewlett-Packard, ParAccel, Info-bright, Exasol, Calpont та Sand, забезпечують швидку продуктивність для багатьох типів запитів через те, як ці системи зберігають та стискають дані стовпцями замість рядків. Зберігання та обробка стовпців швидко стає функцією СКБД, а не окремою категорією продуктів [9]. Hadoop - це проект програмного забезпечення з відкритим вихідним кодом, який виконується в рамках Apache Software Foundation для обробки додатків, що вимагають великих обсягів даних, у розподіленому середовищі з вбудованою паралельністю та відмовою. Найважливішими частинами Hadoop є розподілена файлова система Hadoop (HDFS), яка зберігає дані у файлах на кластері серверів, і MapReduce, програма програмування для створення паралельних додатків, які працюють на HDFS.

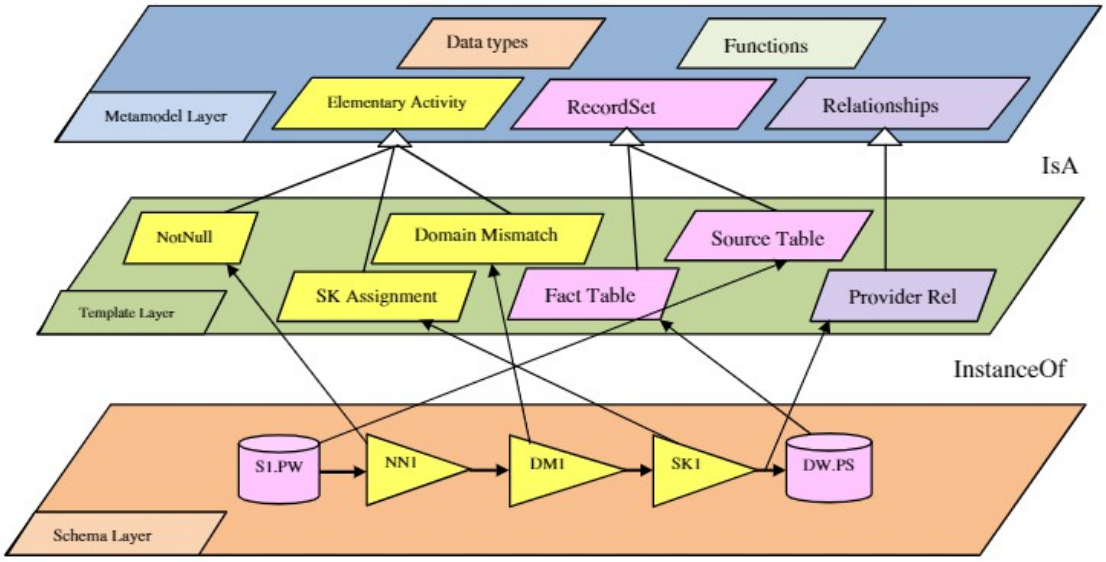

Рисунок 1.3 – Логічна схема процесів СД

Спільнота з відкритим кодом створює безліч додаткових компонентів, щоб перетворити Hadoop на середовище обробки даних корпоративного рівня. Збір цих компонентів називається розподілом Hadoop. Серед провідних постачальників дистрибутивів Hadoop - Cloudera, IBM, EMC, Amazon, Hortonworks та MapR. На сьогоднішній день в більшості додатків клієнтів Hadoop служить проміжним майданчиком та онлайн архівом для неструктурованих та напівструктурованих даних, а також виконує роль аналітичної пісочниці для науковців даних, які надсилають запити до файлів Hadoop безпосередньо перед тим, як дані агрегуються або завантажуються у сховище даних. Але це може змінитися. Hadoop відіграватиме дедалі важливішу роль в аналітичній екосистемі більшості компаній, або співпрацюючи з призовим DW, або виконуючи більшість обов'язків однієї. NoSQL - скорочення "не тільки SQL" - це ім'я, що дається широкому набору баз даних, єдиним спільним потоком яких є те, що вони не потребують SQL для обробки даних, хоча деякі підтримують як SQL, так і не SQL-форми обробки даних [13]. Існує багато типів баз даних NoSQL, і список щомісяця збільшується. Ці спеціалізовані системи побудовані з використанням власних компонентів та компонентів з відкритим кодом, або їх поєднання. У більшості

випадків вони призначені для подолання обмежень традиційних RDBM для обробки неструктурованих та напівструктурованих даних.

1.4. Структура сучасної корпоративної аналітичної системи

Минули часи, коли вся аналітична обробка вперше проходить через сховище даних або вітрину даних (або їх менш відомих братів-розповсюджувачів або тіньової системи даних). Тепер дані надходять до користувачів через безліч корпоративних структур даних, кожна з урахуванням типу вмісту, який він містить, і типу користувача, який хоче їх споживати. Розглянемо архітектуру аналітичної екосистеми, в якій використовуються технології великих даних. У цьому ієрархічному сценарії F2C передбачаємо Fog-вузли на найнижчому шарі, безпосередньо підключені до крайових пристроїв, призначені для агрегування та управління пропускною спроможністю крайових пристроїв - зберігання, зондування, обчислювальної техніки та мережі(рис. 1.4). До цього бачення існує безліч концепцій, підходів та ідей, які можна оцінити на їх придатність, щоб допомогти визначити вузли туману в майбутніх системах [14].

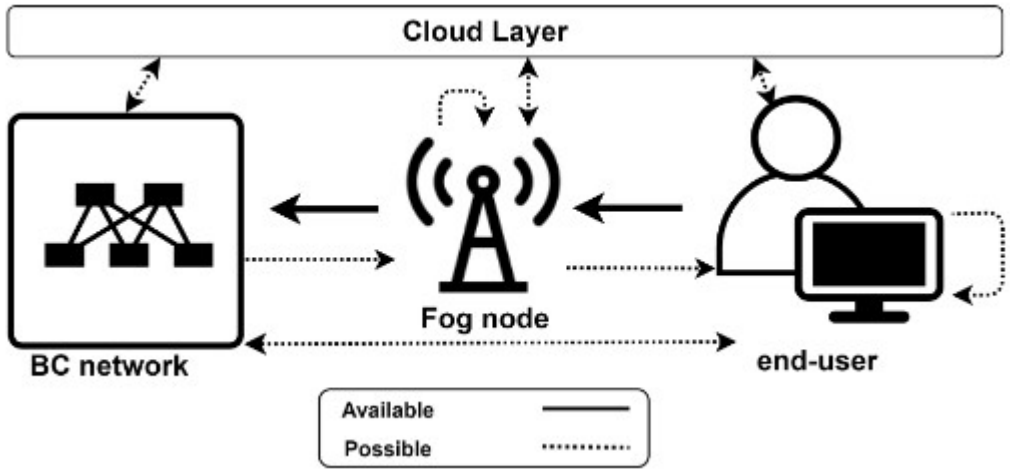

Рисунок 1.4 – Зображення взаємодії Fog-вузла з іншими компонентами

Основною метою даної роботи є систематичне вирішення питань, що

стосуються взаємодії Fog-компонентів з системою. При порівнянні з хмарними обчисленнями немає необхідності у формальному визначенні хмарного вузла, головним чином тому, що хмарні обчислення самі по собі децентралізовані, на відміну від Fog.

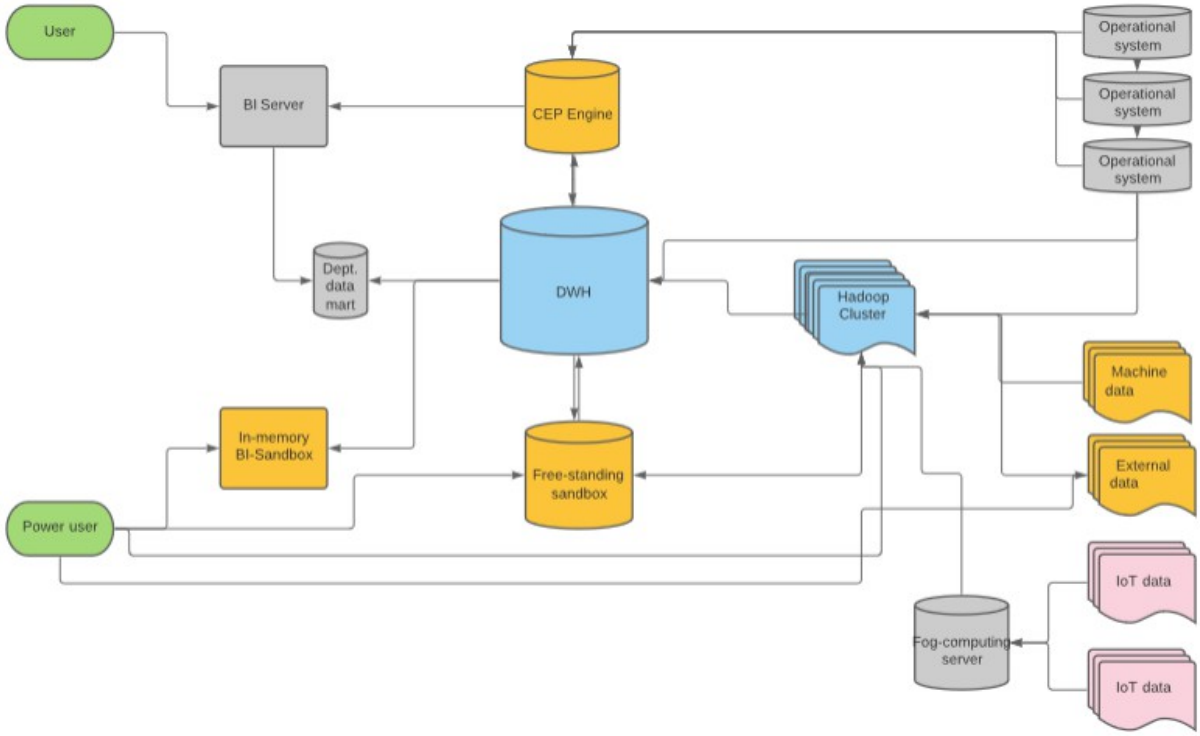

Рисунок 1.5 – Модель аналітичної екосистеми

Більшість вихідних даних зараз протікає через Hadoop, який в першу чергу виступає в якості проміжної області та онлайн-архіву(рис. 1.5). Це особливо актуально для напівструктурованих даних, таких як файли журналів та машинно генеровані дані, а також для деяких структурованих даних, які компанії не можуть ефективно зберігати та обробляти в движках SQL. З Hadoop дані надходять у центр зберігання даних, який часто розподіляє дані до нижніх систем, таких як дані, оперативні сховища даних та аналітичні пісочниці різних типів, де користувачі можуть запитувати дані за допомогою знайомих інструментів звітування та аналізу на основі SQL [14, 15]. Сьогодні аналітики аналізують необроблені дані всередині Hadoop, пишучи програми MapReduce на Java та інших мовах. Користувачі можуть здійснювати запити та обробляти

дані Hadoop за допомогою інструментів інтеграції даних та запитів на основі SQL. Революція великих даних полягає не лише в аналізі великих обсягів та нових джерел даних, а й у збалансуванні вирівнювання та консистенції даних Таким чином, аналітична екосистема має потоки даних "зверху вниз" та "знизу вгору", які відповідають усім бізнес-вимогам щодо звітування та аналізу. Тут вихідні дані обробляються, уточнюються та штампуються заздалегідь визначеною структурою даних, як правило, розмірною моделлю, а потім споживаються користувачами за допомогою інструментів звітування та аналізу на основі SQL. У цьому контексті задача роботи полягає у створенні семантичних моделей, щоб бізнес-користувачі могли отримувати відповіді на відомі питання, а керівники могли відстежувати ефективність заздалегідь визначених показників.

1.5. Актуальні підходи до питань безпеки сховищ даних

Дані на серверах розробки часто є конфіденційними, оскільки вони беруться із виробничих вихідних систем. Щоб полегшити розгортання, зробимо машини для розробки "середовищем безпеки подібними до виробничих систем". Захист мережі Інтернет - це в основному орієнтована на UNIX система, яка була розроблена в 1960-х роках як спосіб неформального спілкування університетів та дослідників. Інтернет використовує протокол управління передачею/Інтернет-протокол (TCP/ IP). TCP регулює, як дві машини встановлюють і підтримують зв’язок, і вимагає розмови вперед-назад [12]. Атака відмови в обслуговуванні може бути успішно створена, коли машина ініціює багато з'єднань, але ніколи не завершує їх. Приймальна машина може засмітитися незавершеними сеансами, які повинні повільно закінчитися, перш ніж бути випущеними. Частина протоколу IP визначає, як машини в Інтернеті знаходять одне одного. Це може розглядатися як конверт, передбачувана адреса та адреса повернення, але не додане привітання чи повідомлення. IP в якомусь сенсі ненадійний; саме по собі це не гарантує, що

повідомлення доставлене або що воно яким-небудь чином є законним. IP просто надає адреси та конверт. Повідомлення повинні розглядатися з певним рівнем підозри; IP-адресу джерела може "підробити" кожен, хто має фізичне підключення до магістральної мережі Інтернет. Іншими словами, ви не маєте реальних гарантій того, що зв’язок надходить від відомого хоста, просто подивившись на адресу джерела IP. Подібним чином, пакет зв'язку із законного джерела може бути перенаправлений у будь-яке місце призначення, скопіювавши пакет і замінивши IP-адресу призначення. Незважаючи на те, що все це звучить загрозливо, важливо не потрапляти в бік слабких сторін IP. Ви можете досягти всієї надійності та безпеки, які вам потрібні, використовуючи TCP/IP, але ви робите це, забезпечуючи належні мережеві компоненти на місці та захищаючи вміст конверту за допомогою шифрування, а не шляхом обробки адрес зовні.

Більшість інтрамереж всередині організацій використовують той самий протокол TCP/IP, що і Інтернет. Такі мережі мають складний набір пристроїв та функцій, які допомагають управляти потоком пакетів навколо організації та до Інтернету, а також обмежують несанкціонований доступ до інтрамережі та внутрішніх інформаційних ресурсів [17]. Ці пристрої та функції включають маршрутизатори, брандмауери та сервер каталогів.

У кожному місці, де одна мережа підключається до іншої, повинен бути пристрій, який називається маршрутизатором, який передає пакети між двома мережами. Маршрутизатор прослуховує кожен пакет TCP/IP у кожній мережі і переглядає адреси призначення. Якщо пакет в одній мережі адресовано комп’ютеру в іншій мережі, то завдання маршрутизатора полягає в тому, щоб пропустити цей пакет. Якщо нічого іншого, маршрутизатори роблять корисну роботу, виділяючи "локальний" трафік у кожній мережі. Якби цього не сталося, у віддалених мережах було б занадто багато непотрібного шуму. Маршрутизатори також знають, що робити, якщо вони бачать пакет, адресований зовнішній мережі. Очевидно, що, оскільки маршрутизатор переглядає кожну адресу призначення в кожному пакеті, він також може

шукати адресу джерела. Таким чином, маршрутизатор може служити фільтром пакетів. Маршрутизатор фільтрування пакетів, який також називають брандмауером фільтрації пакетів, може надавати такі корисні функції:

1. відхиляти спроби підключення від невідомих хостів;
2. відхиляти зовнішні спроби, які підробляють як інсайдери;
3. приймати лише трафік, що надходить від однієї машини;
4. припиняти та запобігати забруднення мережі.

Хорошою загальною конфігурацією безпеки для підключеного до Інтернету середовища DW/BI є так звана конфігурація брандмауера екранованої підмережі. Брандмауер екранованої підмережі насправді має два фільтри пакетів. Перший фільтр діє як основний брандмауер і передає майже весь вхідний мережевий трафік на спеціальний внутрішній хост, який називається бастіон-сервером. Бастіонний сервер діє як потужне, контрольоване вузьке місце для всіх комунікацій, що надходять до організації ззовні. За винятком певних спеціально кваліфікованих сторонніх осіб, усі комунікації повинні проходити через програми-проксі, що працюють на окремому, т.з. “бастіонному” сервері. Однак брандмауер фільтрації пакетів може обережно дозволяти вибраним зовнішнім хостам обходити сервер бастіонів і безпосередньо отримувати доступ до інших внутрішніх серверів. Ці "надійні пакети" не потребують проксі-сервера; їм дозволено розмовляти з реальним сервером. Якщо ця конфігурація поєднана з потужною системою автентифікації та шифрування, яку надає сервер каталогів, організація може отримати найкраще з обох світів. Потужні віддалені програми від надійних та автентифікованих клієнтів отримують повний доступ до внутрішніх ресурсів, тоді як звичайні спроби доступу повинні запускати гантлет бастіонного сервера [18].

Шифрування - це потужна технологія, яка при правильному використанні може в значній мірі захистити вміст наших комунікацій. Шифрування в широкому розумінні - це зміна повідомлення із секретним кодом, щоб його могли прочитати лише люди, які знають код. У певному сенсі це все, що

менеджер безпеки DW/BI повинен знати про шифрування. Існує невдала тенденція для любителів шифрування описувати математичну основу схем кодування. Математика шифрування, настільки ж приваблива, як і для нас, техніків, в основному не має відношення до проблеми використання шифрування. Менеджери безпеки DW / BI повинні розуміти використання півтора десятка форм шифрування. Жодна схема шифрування не вирішить будь-яку проблему безпеки. Схеми шифрування варіюються від автоматичних та анонімних і майже не турбують менеджера безпеки DW/BI, до того, що вони добре помітні для всіх користувачів сховища даних і потенційно обтяжливий тягар, якщо їх неправильно реалізувати.

Більшість систем DW/BI використовують двокомпонентний механізм захисту. Перша частина - це автентифікація користувачів - щоб підтвердити, що користувачі є такими, якими вони є, і дозволити їм увійти в систему. Служби автентифікації є або частиною мережі і прив'язані до операційної системи, або сервісу низького рівня, такого як Kerberos. У системі автентифікації налаштовуються користувачі та групи та призначаються користувачі одній або декільком групам [18]. Другим елементом архітектури безпеки є захист даних. Зазвичай це завдання покладається на команду BI, оскільки вона стоїть перед бізнес-користувачами та реалізує функціональність інструментів доступу до даних та баз даних.

Основним питанням безпеки системи DW/BI є запити. Будь-яка письмова діяльність, така як розробка прогнозів та бюджетів, повинна надійно управлятися додатком.

Найбезпечніша система DW/BI - це система, в якій немає даних. Кожного разу, коли ми визначаємо елемент чутливих даних, ми повинні запитати себе, чи справді це потрібно в сховищі даних. Найчастіше відповідь позитивна. Національний ідентифікаційний номер, такий як номер соціального страхування в США, є привабливим унікальним ідентифікатором особи [6]. Однак національні ідентифікаційні номери дійсно слід тримати в таємниці. Можливо замаскувати ідентифікаційний номер під час процесу ETL, створивши

сурогатний ключ для ідентифікації особи. Необхідно контролювати доступ до дуже конфіденційної інформації, наприклад, компенсації працівникам або номерів кредитних карток клієнтів. Якщо неможливо уникнути додавання цих атрибутів до своєї системи DW/BI(рис. 1.6), необхідно їх зашифрувати. Більшість механізмів баз даних можуть шифрувати дані у стовпці. Чим більше обмежених даних, тим складніше надати спеціальний доступ до системи DW/BI. Необхідно подивитись на приклад матриці чутливості даних і визначити, як реалізувати відкритий доступ на сукупних рівнях та обмежений доступ на детальних рівнях.

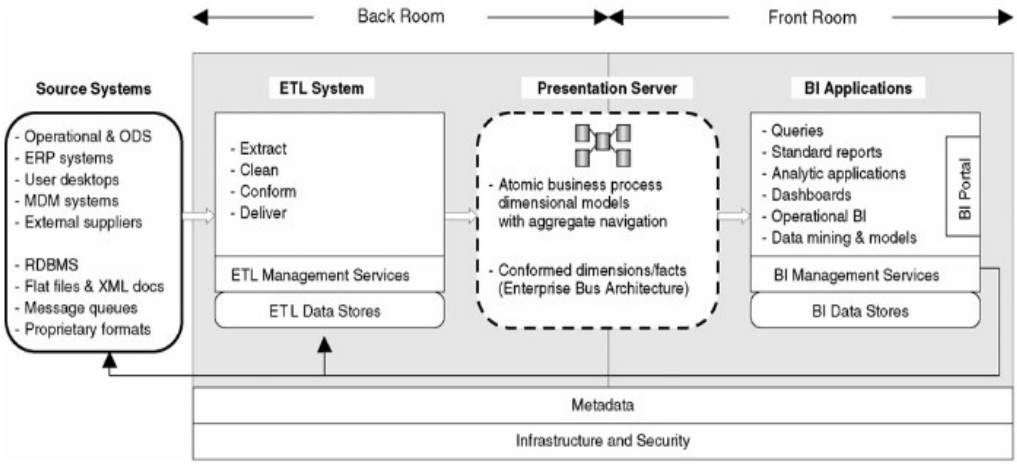

Рисунок 1.6 – Модель архітектури DWH найвищого рівня

Найпростіший спосіб це зробити - за допомогою програми BI, наприклад звіту; визначити звіти на сукупному рівні, які може бачити кожен, і обмежити доступ до звітів, що містять конфіденційну інформацію. Якщо ми дозволяємо спеціальний доступ до базових даних, можливо, доведеться застосувати складні правила доступу до бази даних. Це важко у реляційній базі даних, оскільки реляційний механізм сам по собі не має достатньо метаданих для створення багаторівневих ролей безпеки [13]. Чим більше спеціального доступу потрібно вашим користувачам до даних, які частково обмежені, тим вагомішим є складний BI-орієнтований сервер презентацій, такий як база даних OLAP. Більшість інструментів клієнтського доступу містять деякі функції безпеки.

Потрібно вивчити функції безпеки будь-якого клієнтського інструменту, перш ніж вирішити покладатися на ці можливості. Якщо користувач має привілеї входу до базової бази даних, захист повинен застосовуватися до об'єктів бази даних. В іншому випадку користувачі можуть просто використовувати Excel або будь-який інший клієнтський інструмент для створення спеціального з'єднання, входу в систему та перегляду обмежених даних. Багато інструментів BI можна - або навіть потрібно - налаштувати за допомогою єдиного адміністративного ідентифікатора для доступу до сервера баз даних. Якщо інструмент BI керує окремими логінами користувачів і використовує єдиний адміністративний ідентифікатор для доступу до сервера, тоді весь доступ користувачів повинен бути обмежений цим інструментом (якщо ви також не захистите користувачів у базі даних, як ми описуємо під спеціальним доступом далі в цьому розділі).). І повинно бути надзвичайно обережними, щоб не порушити цей адміністративний ідентифікатор. Більшість реалізацій мають різні вимоги до безпеки. Деякі звіти доступні для будь-кого в компанії, деякі мають обмежений доступ до повного звіту, а деякі звіти дають різні результати залежно від того, хто створює звіт. Окрім звітування, також потрібно буде підтримувати спеціальний доступ.

Спеціальний доступ означає, що користувачеві дозволяється залишати межі попередньо визначених звітів для створення нових запитів, звітів та аналізів. До цього моменту можна було уникнути надання користувачам прав доступу до бази даних, оскільки сервер звітів або додатки BI виконують автентифікацію та авторизацію. Але користувачі, яким потрібен спеціальний доступ, очевидно, потребують прав доступу. У базі даних, що містить конфіденційну інформацію, це означає обробку дозволів на рівні бази даних. У реляційній базі даних легко визначити дозволи на таблиці, подання та стовпці в таблиці або поданні, але визначення рівня безпеки рядка передбачає створення додаткових подань або політик безпеки на рівні рядків [15]. Наприклад, загальним підходом до обмеження доступу користувача до одного або кількох відділів є створення окремої таблиці, яка має принаймні два стовпці:

ідентифікатор бізнес-користувача та список ключів, таких як ідентифікатор відділу, для яких користувачеві дозволено побачити. За допомогою методу перегляду ви створюєте подання, яке приєднує таблицю фактів до таблиці безпеки в стовпці ключів, фільтруючи ідентифікатор користувача. Кожен бізнес-користувач, який запитує це подання, бачить лише рядки для ключів, перелічених для них у таблиці безпеки. Політика безпеки рівня рядка визначається подібним чином, але вона додається до речення WHERE кожного запиту щодо таблиці фактів. Проблема такого підходу полягає в тому, що важко забезпечити доступ до детальних даних, проте забезпечити відкритий доступ до агрегованих даних. Для багатого, але захищеного спеціального доступу більшість систем DW / BI реалізують аналітичний сервер. Додатковою інформацією щодо безпеки, якою можуть керувати ці інструменти, є значна частина їх цінності в середовищі, яке підтримує захищений спеціальний доступ.

1.6. Області використання Fog-обчислень в інформаційних мережах підприємства

Обчислення туману - це парадигма, яка поширює хмару на проміжні мережеві пристрої з обчислювальними та накопичувальними можливостями. Це дозволяє виконувати програми, розташовані ближче до граничних пристроїв та кінцевих споживачів, розподіляючи послуги на цих проміжних пристроях(рис.1.7). Розміщення цих служб впливає на роботу архітектури туману. Архітектура обчислювального середовища Fog, представлена на малюнку 3, передбачає ієрархічне розташування вузлів Fog по всій мережі між датчиками та хмарою в основі мережі. В архітектурі датчики IoT розміщуються в самому нижньому шарі архітектури та розподіляються в різних географічних місцях, відчуваючи навколишнє середовище та випромінюючи спостережувані значення на верхні шари через шлюзи для подальшої обробки та фільтрації. Подібним чином IoT Actuators працюють на найнижчому шарі архітектури і відповідають за управління механізмом чи системою [19].

Потік даних IoT складається з послідовності незмінних значень, випромінюваних датчиками. У архітектурі будь-який елемент мережі, який здатний розміщувати модулі додатків, називається Fog Device або пристрій туману. Туманні пристрої, які підключають датчики до Інтернету, зазвичай називають шлюзами.

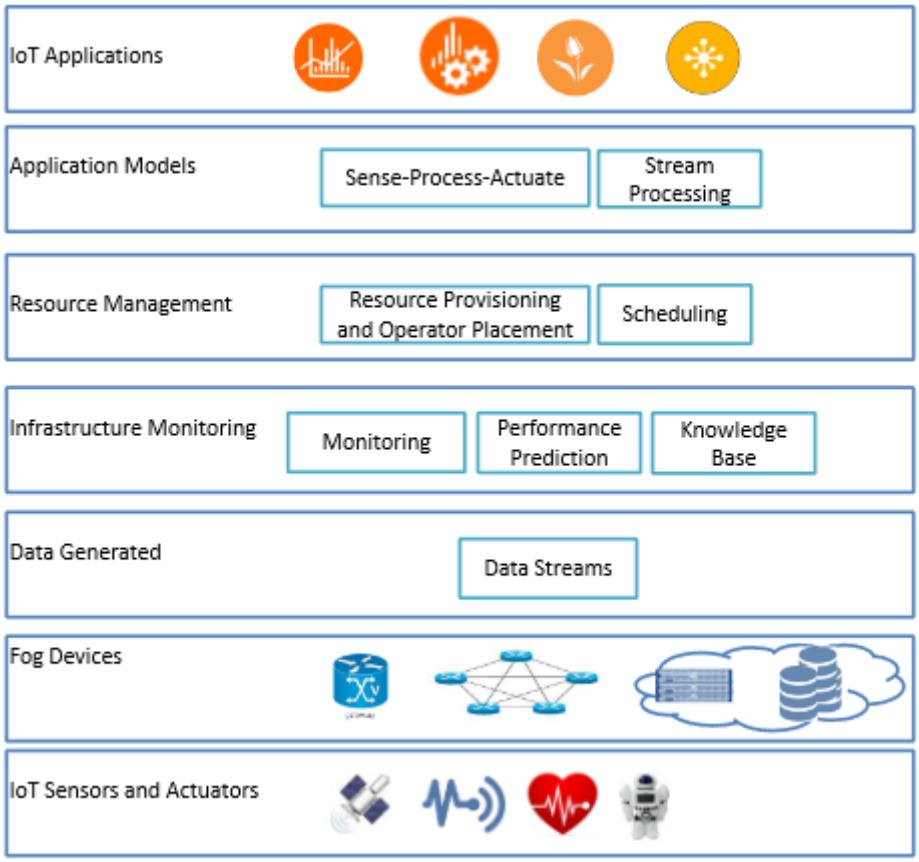

Рисунок 1.7 – Етапи роботи системи з Fog-пристроями

Пристрої Fog також включають хмарні ресурси, які надаються на замовлення з географічно розподілених центрів обробки даних. Крім того, архітектура визначає три основні служби та дві моделі додатків для середовищ Fog та IoT, які описані нижче. пускачі, протитуманні пристрої та елементи мережі. Вони відстежують програми та послуги, розгорнуті в інфраструктурі, відстежуючи їх ефективність та стан. Компоненти моніторингу надають цю інформацію іншим службам за потребою. Управління ресурсами є основним компонентом архітектури і складається з компонентів, які узгоджено керують

ресурсами таким чином, щоб обмеження якості QoS на рівні програми були дотримані, а втрата ресурсів зведена до мінімуму. З цією метою компоненти Розміщення та Планувальник відіграють важливу роль, відстежуючи стан доступних ресурсів (інформація, що надана Моніторинговою службою), щоб визначити найкращих кандидатів для розміщення модуля програми. проблема полягає у використанні ресурсів вузлів IoT, враховуючи обмеження на споживання енергії. На відміну від хмарних центрів обробки даних, обчислення Fog охоплюють велику кількість пристроїв з неоднорідним енергоспоживанням, що ускладнює досягнення управління енергією. Отже, оцінка впливу застосувань та політики управління ресурсами на споживання енергії має вирішальне значення перед розгортанням у виробничих середовищах. Тому потрібен компонент моніторингу потужності в архітектурі, який відповідає за моніторинг та звіт про споживання енергії пристроями Fog в моделюванні. Моделі застосування (програмування): Додатки, розроблені для розгортання в Fog, базуються на моделі розподіленого потоку даних (DDF). Ця модель дозволяє представляти додаток у вигляді спрямованого графіка, вершини яких представляють модулі програми та спрямовані ребра, що показують потік даних між модулями [20]. Архітектура підтримує дві моделі, що використовуються для програм IoT:

а) модель Sense-Process-Actuate: інформація, зібрана датчиками, передається у вигляді потоків даних, на що впливають програми, що працюють на пристроях Fog, а отримані команди надсилаються на виконавчі механізми;

б) модель обробки потоку: модель обробки потоку має мережу прикладних модулів, що працюють на пристроях Fog, які безперервно обробляють потоки даних, що випромінюються від датчиків.

Інформація, видобута з вхідних потоків, зберігається в центрах обробки даних для широкомасштабної та довгострокової аналітики.

2 ЕТАПИ РОЗРОБКИ

2.1 Мета і технічна задача дослідження

Метою дослідницької роботи є проектування та реалізація ефективного сховища даних для системи обліку та управління підприємством, що отримує вихідні дані з різних джерел, включаючи датчики, встановлені на підприємстві.

Система отримує різні за походженням вхідні дані, які потрібно привести до стандартного вигляду, визначеного відповідно до вимог і придатного для подальшої роботи з даними на етапі аналізу даних.

Завдання дослідження:

1. виконати аналіз існуючих комерційних рішень на ринку DWH;
2. визначити загальні вимоги до системи відповідно до специфіки підприємства;
3. визначити етапи проектування;
4. спроектувати окремі компоненти та сутності сховища даних;
5. спроектувати компоненти для реалізації туманних обчислень, що використовують локальний сервер, для первинної обробки даних;
6. об’єднати готові компоненти в межах однієї системи для забезпечення функціональності ETL-процесів;
7. провести аналіз ефективності роботи системи;
8. визначити можливість та перспективи використання системи що складається спроектованих компонентів.

Проаналізувавши задачі, було визначено, що вхідними даними є плоскі файли та інформація, отримана з датчиків та первинно оброблена на етапі туманних обчислень. Вихідними - дані стандартизованого вигляду, придатні до подальшого використання в OLAP-системах.

2.2. Визначення підходів проектування

Розмірне моделювання є логічним методом проектування для структурування даних, щоб воно було інтуїтивно зрозумілим для бізнес-користувачів і забезпечувало швидку продуктивність запитів. Простота є основною вимогою, оскільки вона гарантує, що користувачі можуть легко зрозуміти бази даних, а також дозволяє програмному забезпеченню ефективно орієнтуватися в базах даних.

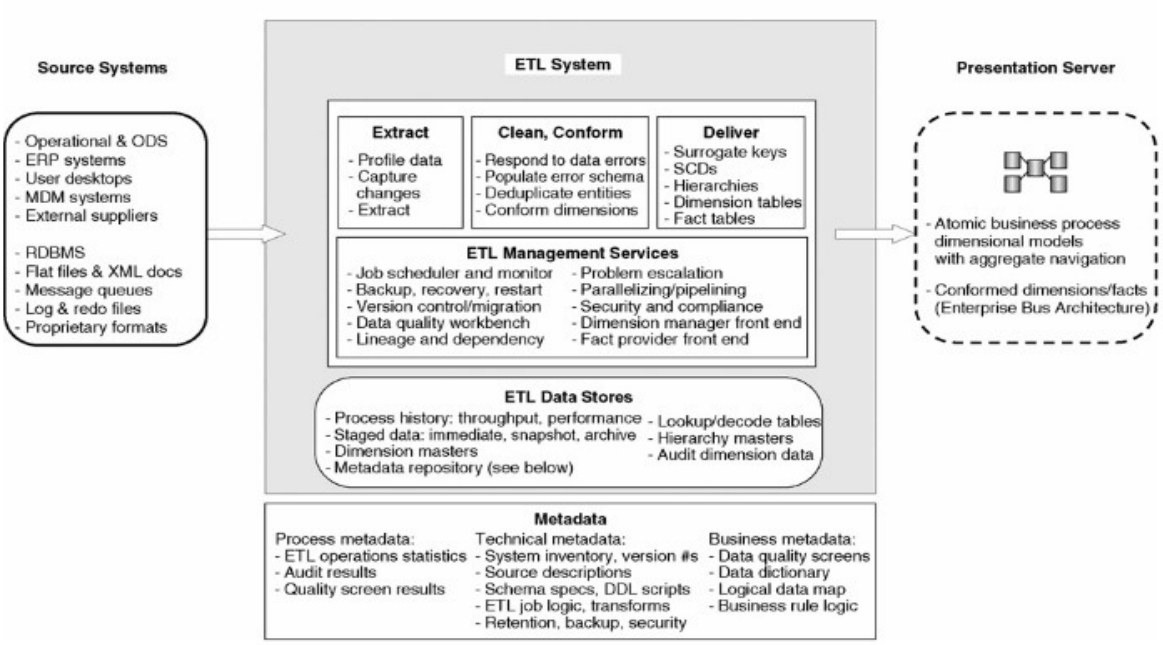

Рисунок 2.1 – Модель архітектури сховища даних

На випадок кожного випадку, IT-організації, консультанти, користувачі та постачальники тяжіють до простої розмірної структури(рис. 2.1), щоб відповідати основним людським потребам у простоті. Вимірне моделювання ділить світ на виміри та контекст. Вимірювання фіксуються бізнес-процесами організації та підтримуючими їх операційними джерелами. Розмірне моделювання широко прийнято як кращий підхід для презентації DW/BI. Практичні фахівці та досвідчені фахівці визнали, що дані, представлені

інструментам бізнес-аналітики, повинні бути просто обґрунтованими, щоб мати шанс на успіх. Вимірювання, як правило, є числовими значеннями; ми називаємо їх фактами. Факти оточені в основному текстовим контекстом, який відповідає дійсності на момент фіксування факту. Цей контекст інтуїтивно розділений на незалежні логічні згустки, які називаються вимірами. Розміри описують контекст вимірювання "хто, що, коли, де, чому і як". Кожен з бізнес-процесів організації може бути представлений мірною моделлю, яка складається з таблиці фактів, що містить числові вимірювання, оточені галом таблиць розмірів, що містять текстовий контекст(рис. 2.2). Цю характерну зіркоподібну структуру часто називають зоряним з'єднанням, що відноситься до найперших днів реляційних баз даних. Розмірні моделі, що зберігаються на реляційній платформі бази даних, зазвичай називаються зоряними схемами; розмірні моделі, що зберігаються в багатовимірних структурах аналітичної обробки в режимі онлайн (OLAP), називаються кубами.

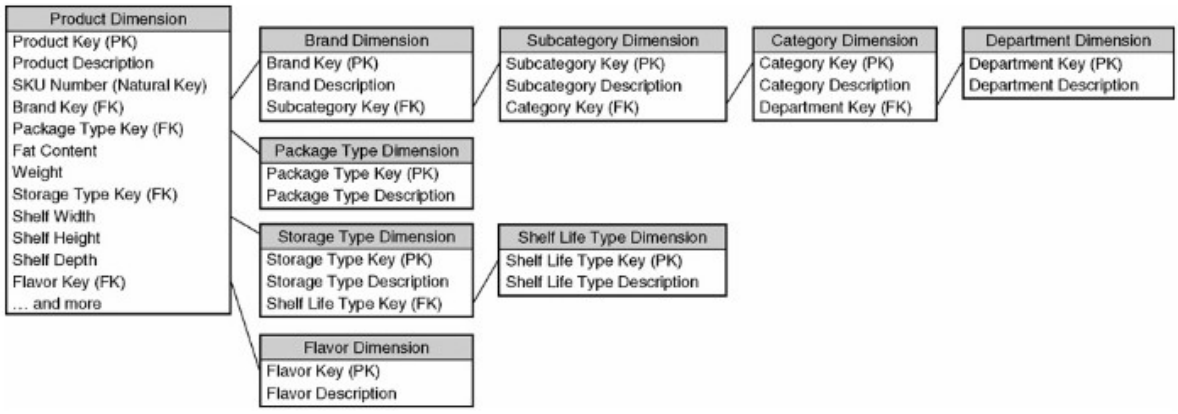

Рисунок 2.2 – Типова діаграма відношень таблиць вимірів

Рекомендують майже у всіх випадках заповнювати куби OLAP із розмірних схем зірок.

Моделювання третьої нормальної форми (3NF) значно відрізняється від розмірного моделювання. Моделювання 3NF - це техніка проектування, яка прагне усунути надмірність даних. Дані поділяються на безліч дискретних сутностей, кожна з яких стає таблицею в реляційній базі даних. Ця нормалізація

надзвичайно корисна для обробки транзакцій, оскільки робить завантаження та оновлення транзакцій простими та швидкими. Оскільки нормалізоване моделювання фіксує мікрозв’язки між елементами даних, навіть бізнес-процес простих замовлень призводить до десятків таблиць, які зв’язані між собою спантеличеною павутиною об’єднань. Нормалізована модель для підприємства має сотні логічних об'єктів. Системи корпоративного планування ресурсів (ERP), як правило, мають тисячі організацій. Кожна з цих сутностей зазвичай перетворюється на окрему фізичну таблицю, коли база даних реалізована. Моделі 3NF іноді називають моделями ER. ER - це скорочення від співвідношення сутність. Діаграми взаємозв'язку сутності (ER-діаграми або ERD) - це креслення вікон і рядків для передачі зв'язків між таблицями бази даних. І 3NF, і розмірна модель можуть бути представлені на діаграмах взаємозв’язків сутності, оскільки обидві складаються із об’єднаних реляційних таблиць; Ключова різниця між 3NF та розмірними моделями - це ступінь нормалізації. Оскільки обидва типи моделей можуть бути представлені як ERD, варто зазначити, що перехід від нормалізованих структур до мірних моделей є відносно простим. Першим кроком є позначення взаємозв’язків багато-до-багатьох у нормалізованій моделі, що містить числові та адитивні неключові метрики як таблиці фактів; таблиці фактів, як правило, нормалізуються до 3NF у розмірній моделі, оскільки відповідний контекст видаляється до таблиць розмірів. Другим кроком є денормалізація решти таблиць у таблиці з плоским розміром за допомогою однокомпонентних ключів, які підключаються безпосередньо до таблиці фактів; таблиці розмірностей найчастіше нагадують таблиці другої нормальної форми з багатьма дескрипторами низької потужності. До переваг вимірного моделювання відносять:

а) Зрозумілість є однією з основних причин того, що розмірне моделювання є загальновизнаною найкращою практикою структурування даних, представлених на рівні бізнес-аналітики. Вимірну модель легше зрозуміти діловому користувачеві, ніж типова нормалізована модель вихідної системи, оскільки інформація групується у цілісні бізнес-категорії або виміри,

які мають сенс для бізнес процесів. Категорії бізнесу допомагають користувачам орієнтуватися в моделі, оскільки цілі категорії можна ігнорувати, якщо вони не стосуються конкретного аналізу. Але "максимально проста" не означає, що модель є спрощеною. Розмірна модель повинна відображати складність бізнесу. Якщо ви спрощуєте, модель втрачає цінну інформацію, що є критично важливою для розуміння бізнесу та його ефективності.

б) Ефективність запитів є другим домінуючим фактором для розмірного моделювання. Денормалізація ієрархій вимірів та пошук декодування може мати значний вплив на продуктивність запиту. Більшість оптимізаторів реляційних баз даних налаштовані на приєднання зірок. Передбачувана структура розмірної моделі дозволяє базі даних робити суворі припущення щодо даних, які сприяють підвищенню продуктивності. Механізм бази даних використовує приєднання зірки, спочатку обмежуючи таблиці розмірностей та збираючи ключі, що задовольняють фільтрам запитів, а потім запитуючи таблицю фактів за допомогою декартового продукту відповідних ключів розмірності. Це далеко не спроба оптимізувати запити сотень взаємопов’язаних таблиць у великій нормалізованій моделі. І хоча загальний набір мірних моделей на підприємстві складний, обробка запитів залишається передбачуваною; ефективність управління здійснюється шляхом запиту кожної таблиці фактів окремо.

Окрім простоти використання та продуктивності запитів, існує ряд додаткових переваг, пов’язаних із розмірним моделюванням. Однією тонкою перевагою є те, що кожен вимір є еквівалентною точкою входу в таблицю фактів. Ця симетрія дозволяє розмірній моделі витримувати несподівані зміни в шаблонах запитів. Немає передбачуваної моделі при великій кількості запитів, що надходять з усіх можливих кутів під час спеціальної атаки. Симетрична структура розмірної моделі дозволяє їй ефективно обробляти все, що на неї накинуто. У технічному плані це означає, що оптимізація запитів для баз даних зіркового з’єднання є простою, передбачуваною та керованою.

2.3. Визначення компонентів

2.3.1 Аналіз процесу створення розмірної моделі

Створення розмірної моделі - це дуже ітераційний та динамічний процес. Після декількох етапів підготовки проектне завдання починається з початкової графічної моделі, отриманої з матриці шини. Він визначає обсяг конструкції та уточнює суть запропонованих таблиць фактів. На початкових сесіях проектування також слід визначити основні виміри, що застосовуються до кожної таблиці фактів, запропонований перелік атрибутів вимірювань та метрик фактів, а також будь-які проблеми, що потребують додаткового дослідження. Після завершення моделі високого рівня починають складати таблицю моделей за таблицею та детально аналізують визначення, джерела, взаємозв'язки, проблеми якості даних та необхідні перетворення [8]. Останній етап процесу моделювання передбачає перегляд та перевірку моделі із зацікавленими сторонами, особливо представниками бізнесу. Основними цілями цієї діяльності є створення моделі, яка відповідає бізнес-вимогам, перевірка наявності даних для заповнення моделі та надання команді ETL твердих вихідних цілей. Розмірні моделі розгортаються через низку сесій проектування з кожним проходом(рис. 2.3), що призводить до більш детального та надійного дизайну, який неодноразово перевірявся на ваше розуміння потреб бізнесу. Процес завершено, коли модель чітко відповідає вимогам бізнесу. Типовий дизайн зазвичай вимагає від трьох до чотирьох тижнів для однієї розмірної моделі бізнес-процесу, але необхідний час буде варіюватися в залежності від складності бізнес-процесу, можливості використовувати існуючі узгоджені розміри, наявності обізнаних учасників, існування добре задокументованого детального бізнесу вимог та труднощі досягнення організаційного консенсусу. Першим кроком є ретельний огляд детальної документації щодо вимог. Необхідно оцінити виклики та можливості, з якими стикаються бізнес-користувачі, намагаючись використовувати дані та

проводити аналіз у відповідь на бізнес-проблеми. Необхідно перевести бізнес-вимоги в гнучку розмірну модель, здатну підтримувати широкий спектр аналізу, а не лише конкретні звіти. Все це є критичним вкладом у визначення розмірної моделі. Зрештою, остаточна розмірна модель повинна мати можливість легко підтримувати ці вимоги.

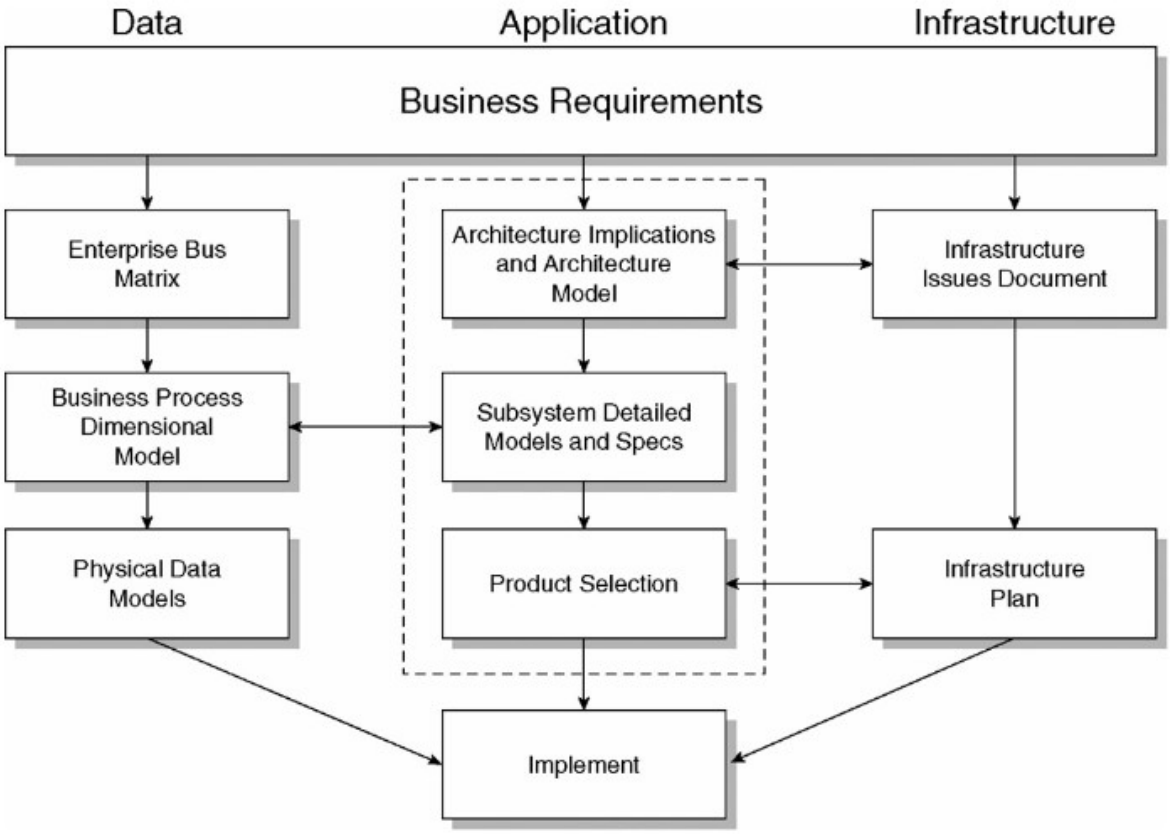

Рисунок 2.3 – блок-схема взаємодії компонентів

Надто зручно розробляти модель даних, орієнтовану на джерела, а не на бізнес. Однак ця модель неминуче не зможе задовольнити бізнес-потреби багатьма незначними, але суттєвими способами. Ці невеликі недоліки додають до слабкої розмірної моделі. Не вистачає розуміння того, як бізнес отримує цінність із наявних даних. Часто бізнес створює нові категорії даних на основі комбінації атрибутів і статусів. Розкриття та включення цих прихованих бізнес-правил у ваші розмірні моделі є секретним соусом для створення середовища, яке перевершує, а не лише відповідає очікуванням ділових користувачів [9]. Питання конвенцій про іменування неминуче виникне під час створення мірної

моделі даних. Важливо, щоб ярлики, які застосовують до даних, мали описовий та послідовний характер, відображаючи тверду ділову орієнтацію. Назви таблиць і стовпців є ключовими елементами інтерфейсу користувача системи DW / BI, коли користувачі орієнтуються в моделі даних, так і в програмах BI. Назва стовпця на зразок "Опис" може бути абсолютно зрозумілою в контексті моделі даних, але нічого не повідомляє в контексті звіту. Частиною процесу проектування розмірної моделі є узгодження загальних визначень та загальних позначень. Іменування є складним, оскільки різні бізнес-групи мають різне значення для одного і того ж імені, наприклад, дохід, і різні імена з однаковим значенням, як-от продаж. Витрата часу на правила присвоєння імен - одне з тих невтомних завдань, яке, здається, мало окупається, але, варто того в довгостроковій перспективі. Встановлення іменних конвенцій не повинно зайняти багато часу. Більшість великих організацій мають такі функції, як управління даними або адміністрування даних, які несуть відповідальність за конвенції щодо імен(рис. 2.4). Якщо в організації немає набору конвенцій іменування, який можливо адаптувати до моделі DW / BI, потрібно буде встановити їх під час розробки мірної моделі.

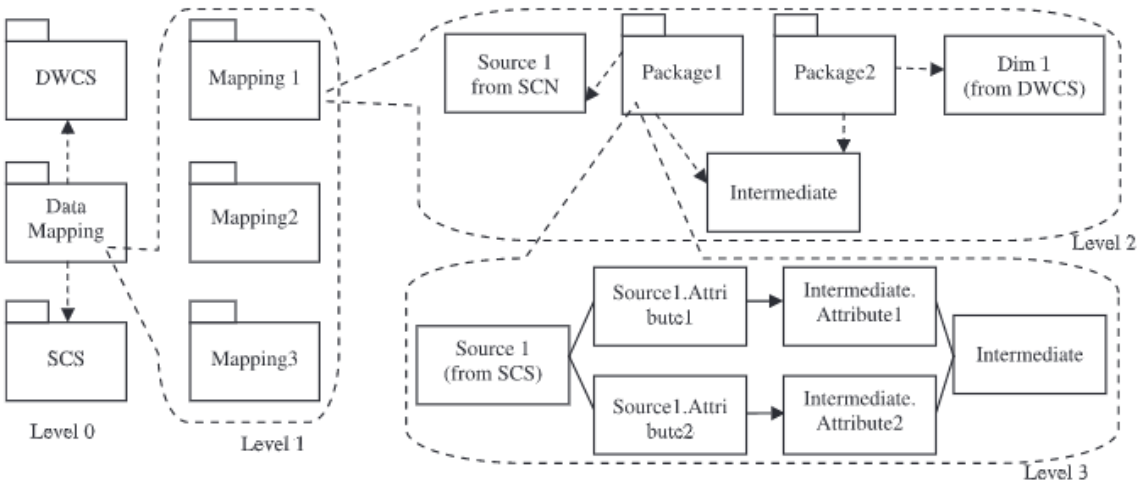

Рисунок 2.4 – Процеси та взаємодія між рівнями в сховищі даних

2.3.2. Таблиці фактів

У таблицях фактів зберігаються показники ефективності, породжені діловою діяльністю або подіями організації. Термін факт стосується кожного показника ефективності. Зазвичай ми не знаємо значення факту заздалегідь, оскільки він змінний; оцінка факту відбувається під час події вимірювання, наприклад, коли отримується замовлення, відправляється відправлення або реєструється проблема служби. Ви можете собі уявити, що стоїте на вантажному причалі, щоб спостерігати за процесом відвантаження; кожен рух товару на вихідну вантажівку породжує показники ефективності або факти, такі як кількість відвантаження. Факти, як правило, постійно оцінюються, тобто вони можуть набувати практично будь-яке значення в широкому діапазоні. Майже кожен факт є числовим. Найкорисніші факти - як числові, так і адитивні. Аддитивність важлива, оскільки програми BI рідко отримують один рядок таблиці фактів; запити зазвичай вибирають сотні або тисячі рядків фактів одночасно, і єдине корисне, що можна зробити із такою кількістю рядків, - це складання. Більшість показників є повністю адитивними. Однак числові вимірювання інтенсивності на момент часу, такі як залишки на рахунках або рівні запасів, є напівдодатковими, оскільки їх неможливо підсумувати за періодами часу. Деякі числові міри абсолютно не адитивні, наприклад, ціна за одиницю товару або температура, оскільки ці факти не можуть бути додані в будь-який аспект [18]. Оскільки більша частина текстового контексту береться з дискретного списку значень домену, такий контекст повинен зберігатися в таблицях вимірів, а не як текстові факти. Хоча велика увага зосереджена на відповідних вимірах, факти також узгоджуються, якщо їх визначення ідентичні. Узгоджені факти можуть мати однакові стандартизовані назви в окремих таблицях. Якщо факти не відповідають, тоді різні тлумачення мають давати різні назви. Таблиці фактів величезні, містять мільйони або мільярди рядків, але вони ефективні. Оскільки таблиці фактів часто зберігають більше 90 відсотків даних у мірній моделі, ми обережно мінімізуємо зайві дані в таблиці

фактів. Крім того, прагнуть зберігати вимірювання, отримані в результаті бізнес-процесу, в єдиній таблиці фактів, яка ділиться в організації.

Таблиці фактів характеризуються багаточастинним ключем, який складається із зовнішніх ключів із таблиць розмірностей, що перетинаються, що беруть участь у бізнес-процесі. Багаточастинний ключ означає, що таблиці фактів завжди виражають взаємозв'язок "багато-до-багатьох". Кожен зовнішній ключ у таблиці фактів повинен відповідати унікальному первинному ключу у відповідній таблиці вимірів. Це означає, що зовнішні ключі таблиці фактів ніколи не повинні мати нульові значення, оскільки це порушує цілісність посилань. Іноді значення розміру відсутнє у цілком законній події вимірювання, наприклад, покупці, здійсненій кимось без частої картки лояльності покупця. У цьому випадку рядок таблиці фактів повинен посилатися на спеціальний ключ у вимірі замовника, що представляє "Нечастий покупець", щоб встановити належний зв'язок зовнішнього ключа та первинного ключа(рис. 2.5). Первинний ключ таблиці фактів, як правило, є підмножиною зовнішніх ключів розмірності. Іноді первинний ключ, який однозначно ідентифікує рядок у таблиці фактів, складається з вироджених вимірів.

Зерно таблиці фактів - це ділове визначення події вимірювання, яка створює рядок фактів. Декларування зерна означає сказати, що саме являє собою рядок таблиці фактів: рядок факту створюється, коли це відбувається.

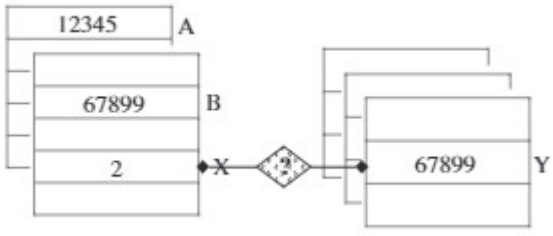

Рисунок 2.5 – Зв’язок ключів таблиці фактів і вимірів

Хоча зерно визначає первинний ключ таблиці фактів, деталізація сама завжди виражається в бізнес-термінах, заздалегідь розглядаючи зовнішні ключі, які можуть бути присутніми. Усі рядки таблиці фактів повинні мати єдине

рівномірне зерно. Зерно визначається фізичними реаліями джерела даних; його визначення стає зрозумілим, коли візуалізується процес вимірювання. Після встановлення зерна можна продовжувати проектування, визначаючи розміри, що прикрашають виміри на такому рівні деталізації [22]. Ця логічна послідовність спочатку декларування зерна, а потім складання розмірного контексту надзвичайно корисна і є ключовою характеристикою процесу розмірного проектування. Фактичні таблиці повинні містити найнижчі, найбільш деталізовані атомні зерна, зафіксовані в процесі бізнесу. Гранульовані дані про атомні факти містять надзвичайну потужність, гнучкість та стійкість. Це дозволяє запитам задавати максимально точні запитання. Навіть якщо користувачів не хвилюють подробиці однієї транзакції або рядка транзакції, їх "питання моменту" вимагає узагальнення цих деталей непередбачуваними способами. Атомарні дані витримують напади за непередбачуваними спеціальними запитами, оскільки деталі можна згрупувати "будь-яким способом". Гранульовані атомарні таблиці фактів також більш не схильні до змін; їх можна витончено розширити, додавши нещодавно отримані факти, нещодавно отримані атрибути виміру та додавши цілком нові виміри (за умови, що зерно не змінено новим виміром).

2.3.3. Таблиці вимірів

На відміну від суворих і гладких якостей таблиць фактів, що складаються лише з клавіш та числових вимірювань, розмірні таблиці є чим завгодно різкими та гладкими; вони заповнені великими та громіздкими описовими полями. Атрибути таблиці вимірів слугують двом критичним цілям: обмеження / фільтрація запитів та маркування результатів запиту. Багато в чому потужність сховища даних пропорційна якості та глибині атрибутів вимірювання; надійні розміри перетворюються на надійні можливості запитів та аналізу. Атрибути виміру - це, як правило, текстові поля, або це дискретні числа, які ведуть себе як текст. Краса розмірної моделі полягає в тому, що всі

атрибути є рівноцінними кандидатами на те, щоб стати фільтрами або мітками. Атрибути виміру повинні бути:

1. багатослівним (мітки, що складаються з повних слів);
2. описовим;
3. повним (відсутні відсутні значення);
4. дискретно оціненим (беруть лише одне значення для кожного рядка в таблиці розмірностей);
5. гарантована якість (відсутність орфографічних помилок, неможливі значення застарілі чи осиротілі значення, або косметично різні версії одного і того ж атрибута).

Нерідкі випадки, коли таблиця розмірів має десятки атрибутів, хоча це не завжди так(рис. 2.6). Коди та скорочення можуть зберігатися як атрибути розмірів; однак доцільно включити відповідне описове поле. Багато інструментів бізнес-аналітики можуть виконувати декодування в межах свого семантичного рівня метаданих; однак ми радимо вам зберігати описові декоди у базі даних, а не покладатися на інструмент BI, оскільки організація часто використовує більше одного інструмента. Ширша узгодженість забезпечується, коли опис ведеться один раз системою ETL і фізично зберігається в таблиці.

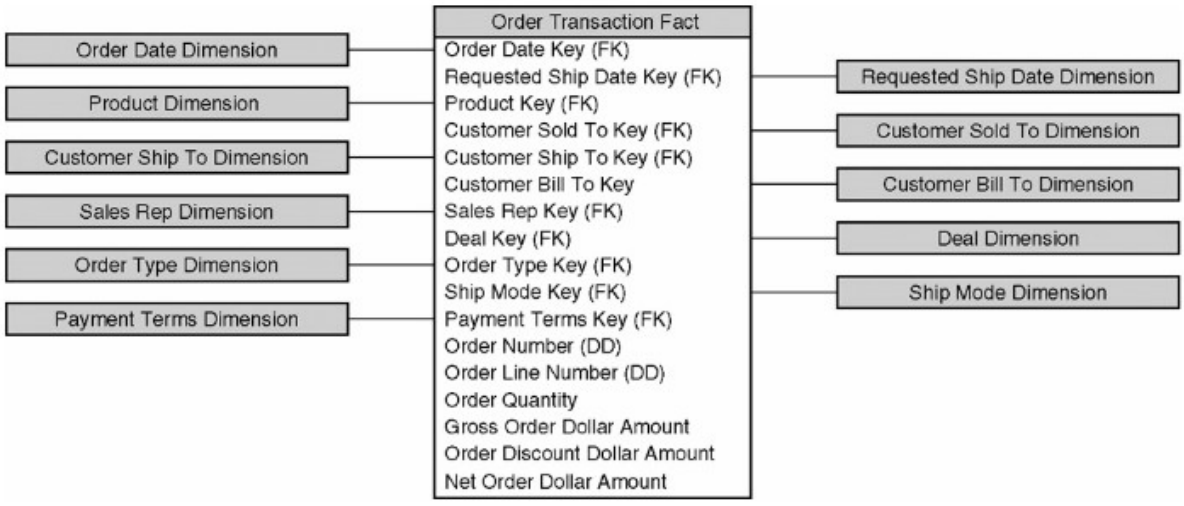

Рисунок 2.6 – Модель таблиці замовлень для підприємства

Таблиці розмірів складаються з високо корельованих наборів атрибутів,

згрупованих для представлення ключових об’єктів бізнесу, таких як його продукція, клієнти, працівники чи приміщення(рис. 2.7). Існує певна ступінь судження дизайнера та інтуїції щодо рішення створити окремі або комбіновані виміри [18]. Наприклад, у роздрібній схемі можливо спробувати поєднати розмір товару з розміром магазину та зробити єдиний розмір продукту-магазину. Припустимо, 1000 товарів і 100 магазинів. Якби не було суттєвої кореляції між товаром і магазином, і кожен товар продавався в кожному магазині, тоді спільний вимір продуктового магазину був би декартовим продуктом двох вихідних вимірів із 100 000 товарних магазинів.

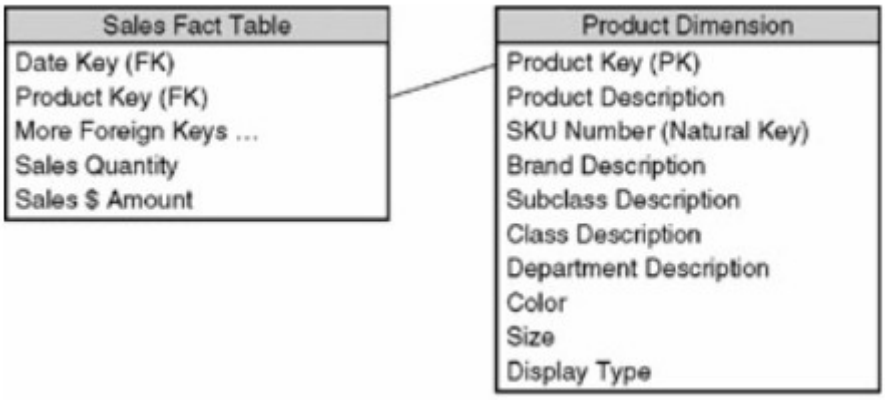

Рисунок 2.7 – Зв’язок таблиці фактів продаж і виміру продуктів

Хоча цей новий комбінований вимір міститиме ту саму інформацію, що і окремі розміри магазину та продукту, більший комбінований вимір буде громіздким, і потенційно може спричинити проблеми з продуктивністю, що не забезпечить переваг користувальницького інтерфейсу і, ймовірно, не буде логічним для бізнес-користувачів. Більшість розмірних моделей отримують десь від 8 до 15 таблиць розмірів. Таблиці фактів(рис. 2.8), які зберігають деталі транзакцій, виявляються найбільш розмірними, а ті, що фіксують моментальні знімки періоду, незмінно мають меншу кількість вимірів. Деякі галузі, такі як охорона здоров’я, горезвісно ускладнюються численними взаємозв’язками «багато-до-багатьох», які неможливо вирішити за допомогою таблиці вимірів. У цих випадках таблиця фактів може мати понад двадцять розмірних зовнішніх

ключів; однак це має бути винятком у верхній частині спектру, а не правилом.

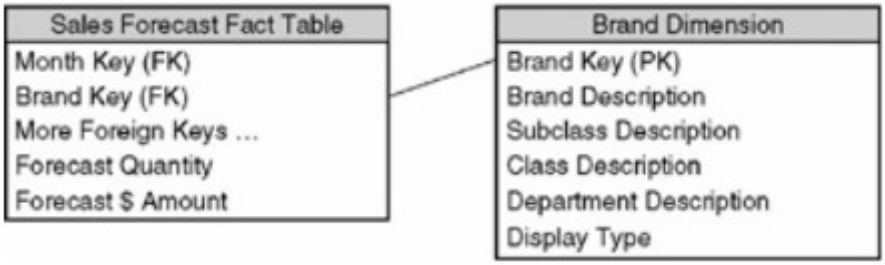

Рисунок 2.8 – Зв’язок таблиці прогнозів продаж і виміру брендів

Логічне проектування розмірної моделі визначається чотирма ключовими рішеннями. Першим кроком у процесі проектування є визначення бізнес-процесу або події вимірювання, що підлягають моделюванню. Кожен рядок матриці шини сховища даних підприємства відповідає бізнес-процесу, визначеному під час збору бізнес-вимог. Цей крок вибору, ймовірно, відбувся під час розстановки пріоритетів із вищим керівництвом бізнесу. Після того, як бізнес-процес був визначений, команда проектувальників повинна оголосити зерно таблиці фактів. Дуже важливо чітко визначити, що саме являє собою рядок таблиці фактів у запропонованій мірній моделі бізнес-процесу. Без узгодження детальної таблиці фактів процес проектування не може успішно рухатися вперед. Створення виміру дати займає особливе місце в кожному сховищі даних(рис. 2.3.3.3), оскільки практично кожна таблиця фактів є тимчасовим рядом спостережень; таблиці фактів завжди мають один або кілька вимірів дати. Свята, робочі дні, фінансові періоди, позначки останнього дня місяця та інші групування або фільтри повинні бути вбудовані у вимір [21]. Основною причиною цієї детальної таблиці дат є видалення останнього залишку знань календаря з додатків BI. Навігація календарем повинна здійснюватися через таблицю розмірів дати(рис. 2.9), а не через жорстко закодовану логіку програми. Незалежно від галузі та пов’язаних із нею бізнес-процесів, усі події вимірювання можна охарактеризувати як транзакції, періодичні знімки або накопичувальні знімки.

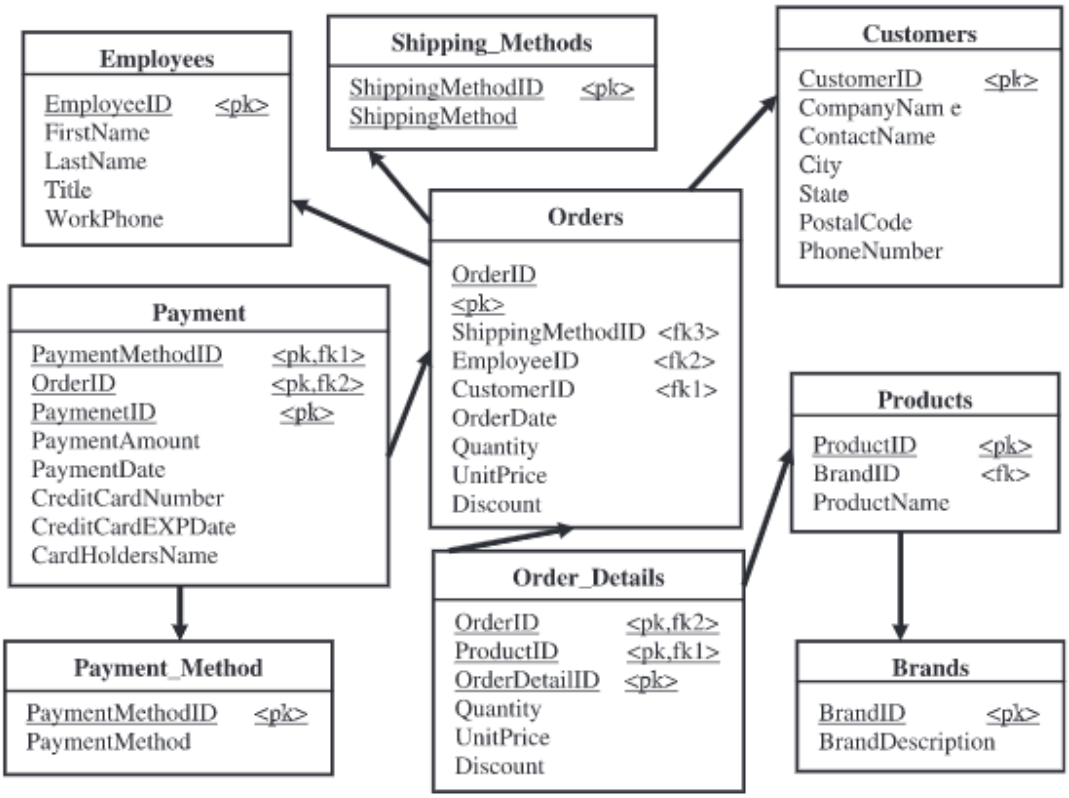

Рисунок 2.9 – Відношення між таблицями фактів і вимірів

Опишемо характеристики цих трьох основних зернових таблиць фактів у наступних розділах. Середовище DW / BI часто поєднує додаткові таблиці фактів, які надають дані з різною базовою деталізацією для більш повного уявлення про бізнес. Хоча таблиці фактів мають спільні розміри, відповідне введення змінюється залежно від основного зерна. Найбільш базовими та найпоширенішими таблицями фактів є транзакційні. Зерно вказано як один рядок на транзакцію або один рядок на рядок транзакції. Зерно транзакцій - це точка в просторі та часі; на той момент вимірювання зерна транзакції повинні бути істинними. Дані транзакцій на найнижчому рівні, як правило, мають найбільшу кількість вимірів, пов’язаних з ними. Зерно - це ділова відповідь на питання "Що таке фактичний рядок?" Як правило, зерно таблиці фактів вибирається якомога атомарнішим або дрібнозернистим. Фактичні таблиці, розроблені з найбільш детальними даними, забезпечують найнадійніший дизайн. Атомарні дані набагато краще відповідають як на несподівані нові

запити, так і на непередбачувані нові елементи даних, ніж вищі рівні деталізації.

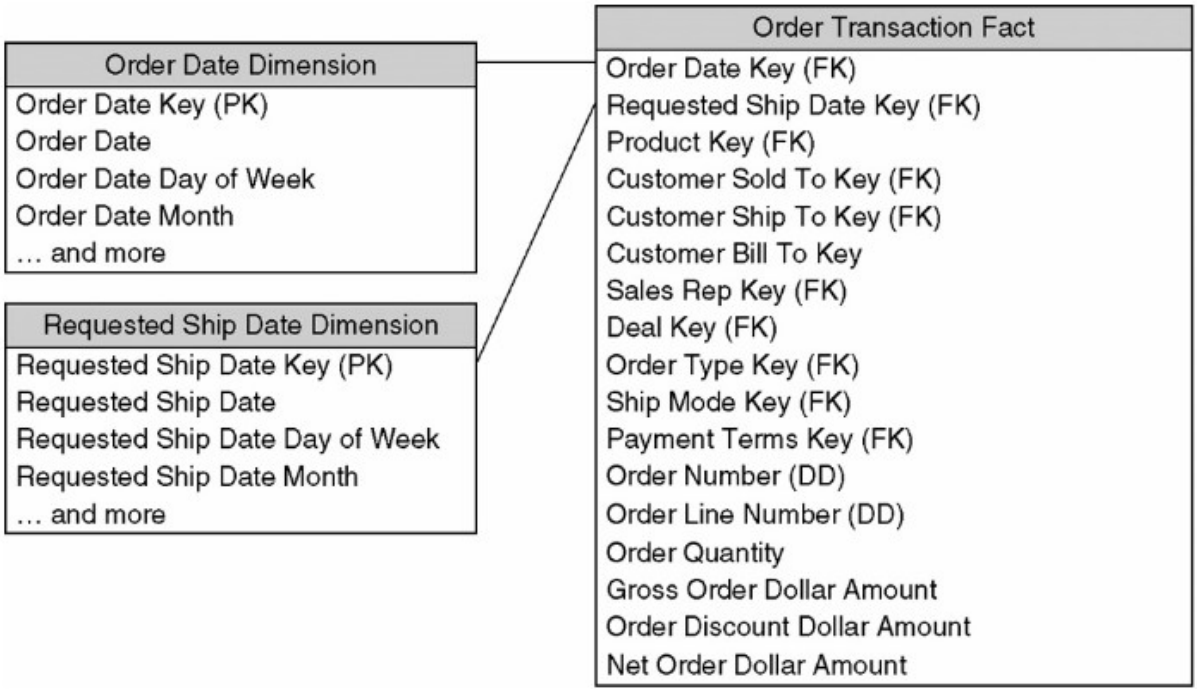

Рисунок 2.10 – Зв’язок виміру дати з таблицею фактів замовлень

Після того, як зерно таблиці фактів міцно встановлено, вибір розмірів стає досить простим. Саме в цей момент можна почати проектувати зовнішні ключі. Саме зерно часто визначає основний або мінімальний набір розмірів. З цього моменту дизайн прикрашений додатковими розмірами, які набувають унікальної цінності в оголошеному зерні таблиці фактів. Завершальним етапом чотириетапного процесу проектування є ретельний підбір фактів або показників, які застосовуються до бізнес-процесу(рис.2.10).

Факти можуть бути фізично зафіксовані подією вимірювання або отримані в результаті цих вимірювань. Кожен факт повинен бути вірним зернистості таблиці фактів.

Щоразу, коли відбувається подія транзакції, також фіксується великий контекст транзакції. Рядки вставляються в таблицю фактів транзакцій лише тоді, коли відбувається активність(рис. 2.11). Після того, як рядок факту транзакції опублікований, він, як правило, не переглядається. Як ми вже

описали раніше в цьому розділі, детальна схема транзакцій є дуже потужною.

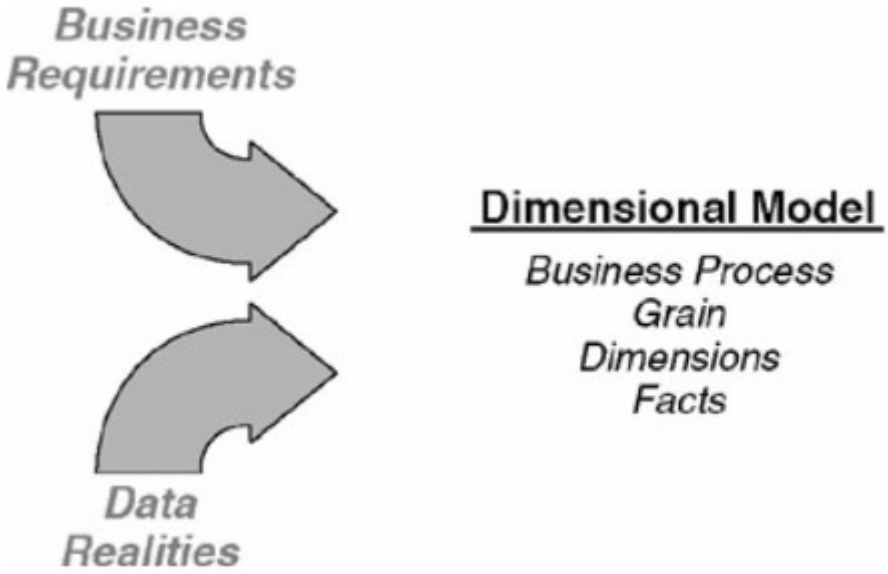

Рисунок 2.11 – Вплив інших компонентів на вимірну модель

Хвилинні вимірювання дозволяють стежити за найбільш детальною діяльністю бізнесу. Однак іноді потрібно доповнити цю деталь знімками, щоб легше візуалізувати тенденції поведінки або ефективності. Другим найбільш поширеним типом таблиць фактів є періодичний знімок. За допомогою періодичних знімків робиться набір знімків, фіксуючи ефективність бізнес-процесу, що охоплює чітко визначений періодичний проміжок часу, і завантажуєте ці знімки в таблицю фактів. З попередньо визначеним інтервалом, наприклад щодня, щотижня або щомісяця, робиться більше фотографій з однаковим рівнем деталізації та послідовно складаються у таблицю фактів. Це надає дизайнеру багато свободи: будь-який факт, який описує діяльність протягом певного періоду, є чесною грою. Деякі бізнес-процеси природно представляються у вигляді періодичних знімків. Наприклад, вартість банківського рахунку природно приймається як знімок щодня або щомісяця. Ви можете передбачити знімки залишків запасів на щоденному або тижневому періоді. Наприклад фінансові звіти - це періодичні знімки. Зрозуміло, що періодичний знімок доповнює докладні факти транзакції, але не є заміною. Все ще потрібні деталі, щоб задати дуже точні запитання щодо продуктивності [9].

Як і таблиці фактів транзакцій, періодичні рядки моментальних знімків, як правило, не переглядаються після завантаження їх у таблицю фактів. Періодичні знімки часто поділяють багато однакових узгоджених вимірів з таблицями фактів транзакцій, проте, як правило, розмірів набагато менше. Третій тип таблиць фактів не часто зустрічається, проте він виконує дуже потужну функцію в певних програмах. На відміну від періодичних знімків, в яких регулярно проводяться вимірювання, накопичувальні знімки використовуються для відображення активності протягом невизначеного періоду часу для процесів, які мають чітко визначені початок і кінець. Коли замовлення рухається через процес виконання, рядки таблиці фактів повинні відображати останній статус. Зрештою, конвеєрна діяльність на позиції замовлення завершується під час створення рахунка-фактури.

2.4. Визначення вихідних даних

Деякі конкретні вихідні системи заслуговують на додаткове визначення, включаючи ERP-системи клієнт/сервер, системи зберігання оперативних даних (ODS), системи керування основними даними (MDM) та XML-файли. Кілька великих компаній виграли битву за забезпечення генерації операційних систем клієнт/сервер [23]. Ці системи планування корпоративних ресурсів (ERP), як правило, складаються з десятків модулів, які охоплюють основні функціональні сфери бізнесу, такі як введення замовлень, людські ресурси, закупівлі та виробництво. Спочатку ці системи були створені для вирішення проблеми додаткових транзакцій. Ми платимо два основних штрафи за цю чудову інтеграцію з точки зору сховища даних. По-перше, у цих вихідних системах ERP часто бувають буквально тисячі таблиць, і в деяких випадках вони можуть бути названі іноземною для вас мовою. Визначення, в яких таблицях містяться цікаві дані для складу та їх взаємозв’язок, може стати кошмаром для ETL процесу. По-друге, ці системи зазвичай не можуть підтримувати значне навантаження аналітичного запиту. Системи, призначені для обробки

транзакцій, мають проблеми з широкими аналітичними запитами - це є головним виправданням для побудови систем DW / BI. Ці системи повинні керувати набагато складнішим розподіленим процесом; їм рідко залишається цикл для аналітичної підтримки. Постачальники ERP визнають цей пробіл і пропонують повну систему DW/BI як частину своєї товарної лінійки. Постачальники інструментів ETL та інші треті сторони також пропонують заздалегідь побудовані модулі витяжки, призначені для стандартних систем ERP. Хоча модулі зберігання даних ERP можуть бути корисними, багато компаній не мають усіх своїх даних у системі ERP. Введення зовнішніх даних у систему DW/BI, що постачається, може бути проблемою. З іншого боку, це може бути настільки ж великою проблемою, як отримати дані. Можливий сценарій, коли компанії використовують постачальника послуг постачальника DW/BI як першу стадію свого процесу ETL, використовуючи свої вбудовані знання про вихідну систему для вилучення даних та переведення їх у більш зрозумілу форму. Потім вони витягують із наданої ERP системи DW / BI та інтегрують висновок з іншими даними, щоб створити остаточну систему, що відповідає DW/BI(рис. 2.12). Термін сховище оперативних даних(ODS) використовувався для опису багатьох різних функціональних компонентів протягом багатьох років. У перші дні зберігання даних часто доводилося створювати ОРС, орієнтований на підтримку рішень, окремо від основного сховища даних, оскільки він базувався на "оперативних" даних, тобто детальних даних на рівні транзакцій. Сховище даних майже завжди було дещо узагальненим і часто затримувалося днями або навіть тижнями [15]. Оскільки склад був повним історичним записом, оперативні дані транзакцій не зберігали у сховищі даних як повну історію. Однак апаратна та програмна технологія, яка підтримує складське господарство, просунулася, і тепер можливо охоплювати дані транзакцій найнижчого рівня як багаторічні історичні часові ряди. Єдиним окремим ОРВ, з яким ми маємо очікувати, є підтримка операційних потреб за допомогою інтегрованих даних про транзакції. Однією з основних оперативних потреб, яку ми спостерігали для ОРВ, є вивантаження потреб оперативної

звітності із системи транзакцій. Існують також сховища даних операційної системи транзакцій, що описуються як ODS. Насправді одним із оригінальних посилань на ОРВ була система транзакцій автоматизованих банкоматів (ATM) у банківській діяльності.

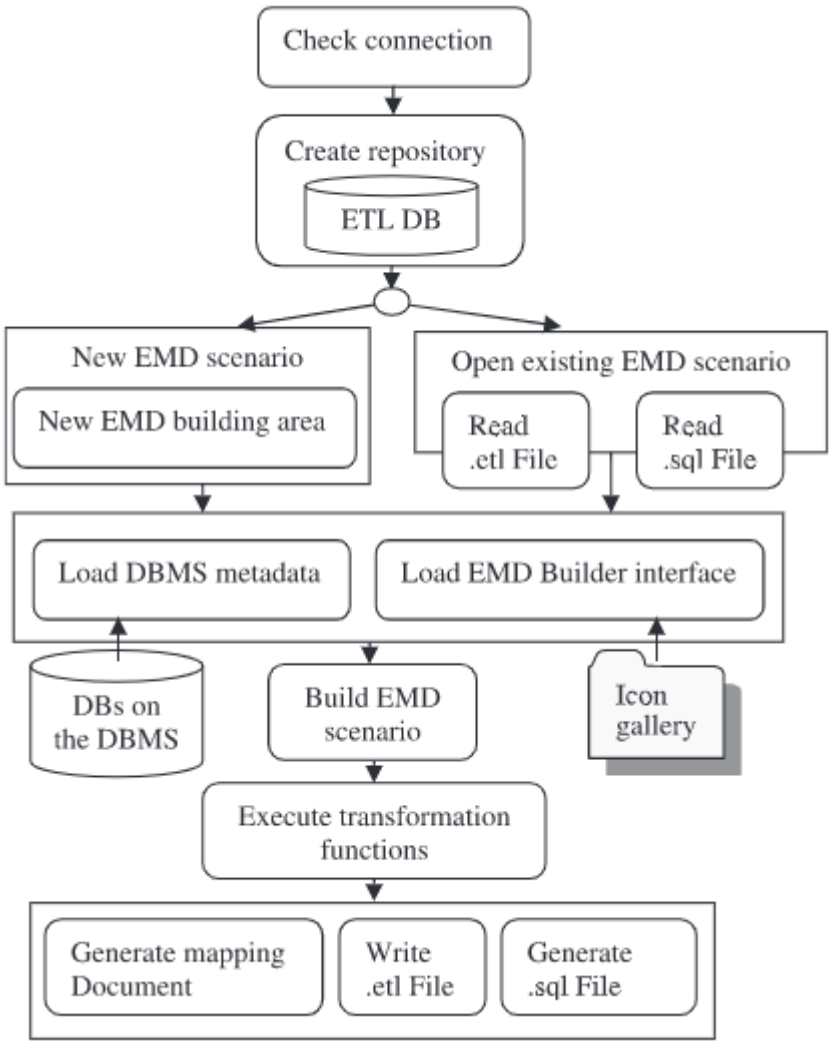

Рисунок 2.12 – Блок схема алгоритму роботи системи

ODS ATM об’єднав ключі бізнесу від різних систем перевірки та заощадження для підтримки транзакцій ATM у реальному часі. Така ODS транзакції є частиною системи транзакцій, відповідаючи на запити транзакцій та відповідаючи рівням обслуговування транзакцій. Він не є частиною системи DW/BI, але служить чудовою потенційною системою джерела. Зрештою, більшість використання ODS, пов’язаних із звітуванням, можна перенести на систему DW/BI, коли дані завантажуються частіше. Деякі організації із

системами транзакцій на основі клієнта/сервера або застарілими системами, обмеженими ресурсами, створюють окрему копію оперативної бази даних, яка служить середовищем звітування для операційних систем. Це відбувається тому, що операційні системи не мають достатньо потужності для підтримки як обробки транзакцій, так і стандартної звітності. ODS, що звітує, підтримує звітування щодо операційних даних за цикл завантаження менш ніж за 24 години без безпосереднього запиту системи транзакцій. Зазвичай це досягається за допомогою якоїсь форми копії системи транзакцій, наприклад, знімка або копії диска, яка робиться кілька разів на день, або реплікації, яка може підтримувати ODS в потоці з системою транзакцій. ODS звітування корисний, оскільки підтримує оперативні звіти, призначені для роботи з моделлю даних системи транзакцій. З іншого боку, це не так добре, оскільки в ньому немає даних про очищення даних, відстеження змін, історичні дані, розміри, що відповідають вимогам підприємства, налаштування продуктивності та корисну розмірну модель, яку має система DW/BI. Це означає, що організація матиме дві версії істини. ODS звітування може бути законним способом задоволення вимог до оперативної звітності, але сподіваємося перенести його в систему DW / BI в якийсь момент.

Системи управління основними даними (MDM) - це централізовані засоби, призначені для зберігання основних копій ключових об'єктів, таких як клієнти або продукти. Ці системи були побудовані у відповідь на поширення таблиць, в яких часто один і той самий клієнт або продукт представлені кілька разів із декількома ключами. Системи MDM призначені для підтримки систем транзакцій і, як правило, мають певні засоби для узгодження різних джерел для одного і того ж атрибута, наприклад, Ім'я Клієнта. Вони забезпечують перехідний механізм для переходу від старих транзакцій до одного джерела інформації про клієнта чи товар. На момент написання цієї статті ми починаємо бачити прийняття систем управління основними даними як частину середовища системи транзакцій. Це чудова річ для системи DW / BI, оскільки велика частина роботи на етапі очищення та відповідності системи ETL

зосереджується на створенні саме цієї єдиної відповідної версії замовника або продукту. Насправді, експерти з якості вже десятки років говорять, що проблеми якості повинні вирішуватись у джерела. Системи управління основними даними допомагають перенести проблему інтеграції даних назад до вихідних систем, які її створили. Але, як зазначено раніше, поки що передчасно поспішати з підходом до SOA, поки системи MDM на основі SOA не стануть більш зрілими [19].

Одним із найпоширеніших форматів за замовчуванням для вхідних вихідних даних був плоский файл. Протягом останніх кількох років дані, надані в схемі XML, все частіше виглядають. Це добре для процесу ETL, оскільки файл XML набагато більше самодокументується, ніж старі плоскі файли. Файли XML найчастіше використовуються для обміну даними через організаційні межі та забезпечення незалежності від конкретних комп'ютерних реалізацій. Насправді в багатьох галузях промисловості створені стандартні схеми XML. Наприклад, галузь страхування має стандартні схеми XML, визначені для компонентів претензії. Уряди також працюють над стандартами XML для багатьох наборів даних, які вони вимагають від компаній, штатів і навіть приватних осіб.

Завжди існує певне місцеве сховище, необхідне для процесу ETL. Зазвичай постійно зберігають підтримуючі таблиці, такі як основні копії розмірів, створені таблиці пошуку та керовані користувачами дані. Ці таблиці зазвичай містяться в базі даних для зручного доступу, обслуговування та резервного копіювання. Також зберігають копії наборів даних про факти та виміри, що поступово витягуються, протягом певного періоду часу. Ці набори даних часто живуть у плоских файлах або файлах, що базуються на власній структурі системи ETL, що підтримує швидке читання та запис. У деяких випадках набір даних зберігатиметься лише до тих пір, поки не буде відомо напевне, що завантаження працювало належним чином і результати правильні. Це може тривати сім днів і більше. В інших випадках, якщо система-джерело перезаписує дані, і ми ніколи не зможемо точно відтворити набір даних, ми

можемо зберігати його роками, використовуючи архівну систему, щоб перенести його на більш дешеве сховище, якщо це необхідно. Індексування додаткових даних вилучення в реляційній базі даних є загальним, і хоча воно має переваги, воно може бути не таким ефективним, як процес потокового передавання. Реляційна система накладає багато накладних витрат на завантаження, індексацію, сортування та інше управління даними. Мови програмування, вбудовані в СУБД, також відносно повільні. З іншого боку, однією з переваг реляційної платформи є її широка доступність. Багато інструментів ETL призначені для роботи з реляційними базами даних. Крім того, можливо вибрати широкий спектр інструментів запитів, щоб перевірити вміст сховищ даних ETL у будь-який момент часу. Часто створюють таблиці пошуку, яких немає у вихідних системах, щоб підтримувати процес ETL. Хорошим прикладом є проста таблиця пошуку для перекладу кодів у вихідній системі на більш змістовні повні описи. Ці описи полегшують читання та розуміння звітів та аналізів BI. Це означає, що процес ETL повинен пам’ятати про відсутні записи, додавати значення за замовчуванням та повідомляти відповідального користувача про необхідність нового запису [14].

2.5. Вимоги до ETL-процесу

Система Extract-Transform-Load (ETL) є основою сховища даних. Правильно розроблена система ETL витягує дані з вихідних систем, забезпечує дотримання стандартів якості та узгодженості даних, узгоджує дані, щоб окремі джерела могли використовуватися разом, і нарешті доставляє дані у готовому до відображення форматі, щоб розробники програм могли створювати додатки та кінцевих користувачів може приймати рішення. Правильно спроектована система ETL(рис. 2.13) робить ефективним сховище даних або призводить до повної недієспроможності, якщо обрані під час проектування рішення були помилковими. Хоча побудова системи ETL є заходом, який не дуже помітний для кінцевих користувачів, він легко споживає 70 відсотків ресурсів,

необхідних для впровадження та обслуговування типового сховища даних. Система ETL додає значну цінність даним. Це набагато більше, ніж інструмент для виведення даних із вихідних систем у сховище даних. Зокрема, система ETL:

а) усуває помилки та виправляє відсутні дані;

б) налаштовує дані з кількох джерел для спільного використання;

в) структурує дані, які можуть бути використані інструментами кінцевих користувачів.

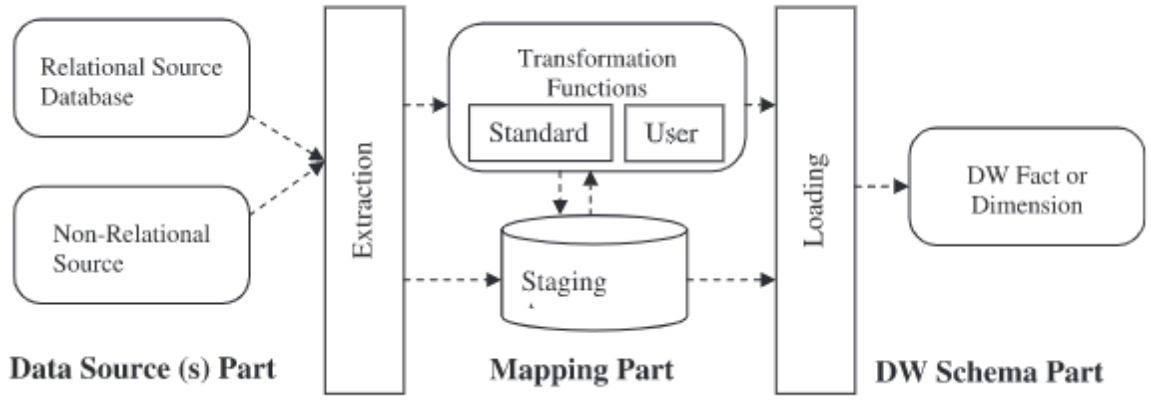

Рисунок 2.13 – Схема роботи сучасного ETL-процесу

Майже всі розуміють основну місію системи ETL: витягувати дані з джерела та завантажувати їх у сховище даних. Окрім цього спеціалісти все частіше усвідомлюють необхідність очищення та трансформації даних. Побудова системи ETL є надзвичайно складною, оскільки вона настільки сильно обмежена неминучими реаліями. система ETL повинна відповідати бізнес-вимогам, форматам та недолікам вихідних даних, існуючим застарілим системам, набору навичок наявного персоналу та постійно мінливим (і законним) потребам кінцевих користувачів. Якщо цих факторів недостатньо, бюджет обмежений, вікна часу обробки занадто вузькі, і важливі частини бізнесу зупиняються, якщо система ETL не доставляє дані до сховища даних. Першим кроком на етапі планування та проектування є облік усіх вимог та

реалій. До них належать:

1. потреби бізнесу;
2. профілювання даних та інші реалії джерел даних;
3. вимоги дотримання;
4. вимоги безпеки;
5. інтеграція даних;
6. затримка даних;
7. архівування та походження даних.

Наступним етапом є етап архітектури. Тут повинні прийматись важливі рішення щодо того, як планується будувати систему ETL. Ці рішення включають:

1. інструмент розроблений власноруч або готові рішення ETL;
2. пакетний та потоковий потік даних;
3. горизонтальна або вертикальна залежність задач;
4. автоматизація планування процесів;
5. обробка винятків;
6. безпека;
7. підтримка якості;
8. відновлення та перезапуск;
9. метадані.

Третім кроком планування та проектування є впровадження системи, що включає:

1. апаратне забезпечення;
2. програмне забезпечення;
3. практики кодування;
4. документаційні практики;
5. конкретні перевірки якості.

Останній крок звучить як адміністрування, але розробка процедур тестування та випуску настільки ж важлива, як і більш відчутні конструкції попередніх двох кроків. Тест і випуск включає в себе проектування:

1. системи розвитку;
2. тестові системи;
3. виробничі системи;
4. процедури передачі;
5. оновлення підходу розповсюдження;
6. процедури моментальних знімків та відкотів системи;
7. налаштування продуктивності.

Потік даних , мабуть, є більш впізнаваним для, оскільки це просте узагальнення старого сценарію E-T-L. Крок вилучення включає:

1. читання моделей джерел даних;
2. підключення та доступ до даних;
3. планування вихідної системи, перехоплення сповіщень та демонів;
4. збір змінних даних;
5. інсценування вилучених даних на диск.

Крок очистки передбачає:

1. застосування властивостей стовпців;
2. примусова структура;
3. застосування правил даних та значення;
4. застосування складних ділових правил;
5. побудова основи метаданих для опису якості даних;
6. перенесення очищених даних на диск.

За цим кроком тісно слідує етап встановлення відповідностей, який включає:

1. відповідність ярликам бізнесу (за розмірами);
2. відповідність бізнес-метрик та показників ефективності (фактично таблиці);
3. дедуплікація;
4. розміщення відповідних даних на диску.

Наступним є крок, де дані передаються до програми кінцевого користувача. Доставка даних із системи ETL включає:

1. завантаження плоских та сніжинкових розмірів;

2. генерування часових вимірів;
3. навантаження вироджених розмірів;
4. завантаження субвимірів;
5. опрацювання розмірів, що пізно надходять, та фактів, що пізніше з’являються:

   5.1. завантаження багатозначних розмірів;

   5.2. завантаження нерівних розмірів ієрархії;

   5.3. завантаження текстових фактів у розмірах;

   5.4. запуск конвеєра сурогатних ключів для таблиць фактів;

   5.5. завантаження трьох основних таблиць фактів;

   5.6. завантаження та оновлення агрегатів;

   5.7.індексування доставлених даних на диск.

Більша частина списку є моделюванням, а не ETL. Проектування системи ETL відбувається одночасно з проектуванням цільових таблиць. Базовий потік даних із чотирьох кроків контролюється етапом операцій, який триває від початку етапу вилучення до кінця етапу доставки. Операції включають:

1. планування;
2. виконання роботи;
3. обробка винятків;
4. відновлення та перезапуск;
5. перевірка якості;
6. звільнення;
7. підтримка.

Коротко представимо основні етапи потоку даних:

а) вилучення;

б) очищення;

в) відповідність та доставка.

Кожна система ETL повинна індексувати дані у різних постійних та напівпостійних формах. йде мова про термін ”Staging”, мається на увазі запис даних на диск, і з цієї причини систему ETL іноді називають областю

підготовки. Зазвичай рекомендують принаймні якусь інсценізацію після кожного з основних етапів ETL (витяг, очищення, відповідність та доставка). Можна виділити такі важливі структури даних, що необхідні у типових системах ETL: плоскі файли, набори даних XML, незалежні робочі таблиці СУБД, нормовані схеми сутності/відносин (E/R) та мірні моделі даних. Також сюди включають деякі спеціальні таблиці, юридично значущі таблиці відстеження аудиту, що використовуються для підтвердження походження важливих наборів даних, а також таблиці відображення, що використовуються для ведення ланцюжка сурогатних ключів. Цей етап починається з пояснення того, що потрібно для розробки логічного відображення даних після завершення аналізу даних. Карта логічних даних надає розробникам ETL функціональні характеристики, необхідні для побудови фізичного процесу ETL. Основною відповідальністю сховища даних є надання даних із різних застарілих додатків у межах усіх корпоративних даних в одному згуртованому сховищі. Після вилучення дані піддають очищенню та відповідності. Очищення означає виявлення та виправлення помилок та недоліків у даних. Відповідність означає вирішення конфліктів маркування між потенційно несумісними джерелами даних, щоб їх можна було використовувати разом у корпоративному сховищі даних. Зазвичай зосереджують увагу на цілях очищення даних, техніках, метаданих та вимірах(рис. 2.14). Зокрема, розділ техніки оглядає ключові підходи до профілювання даних та очищення даних, а розділ вимірювань наводить приклади того, як здійснювати перевірку якості даних, яка викликає оповіщення, а також як наводити керівника якості даних щодо загальний стан даних. Вся суть сховища даних полягає в тому, щоб доставляти дані у простому, дієвому форматі на користь кінцевих користувачів та їх аналітичних програм. Таблиці розмірів - це контекст вимірювань бізнесу. Вони також є точками входу до даних, оскільки вони є цілями для майже всіх обмежень сховища даних, і вони забезпечують значущі мітки на кожному рядку виводу. Процес ETL, який завантажує розміри, є складним завданням, оскільки він повинен поглинати складність вихідних систем і перетворювати дані у

прості, що обираються як атрибути розмірів.

Чітко виділяють наступні кроки:

1. призначити сурогатні ключі;
2. навантажувати повільно мінливі розміри типів 1, 2 і 3;
3. заповнювати таблиці мостів для багатозначних та складних ієрархічних вимірів ;
4. згладжувати ієрархії та вибірково розміри сніжинки .

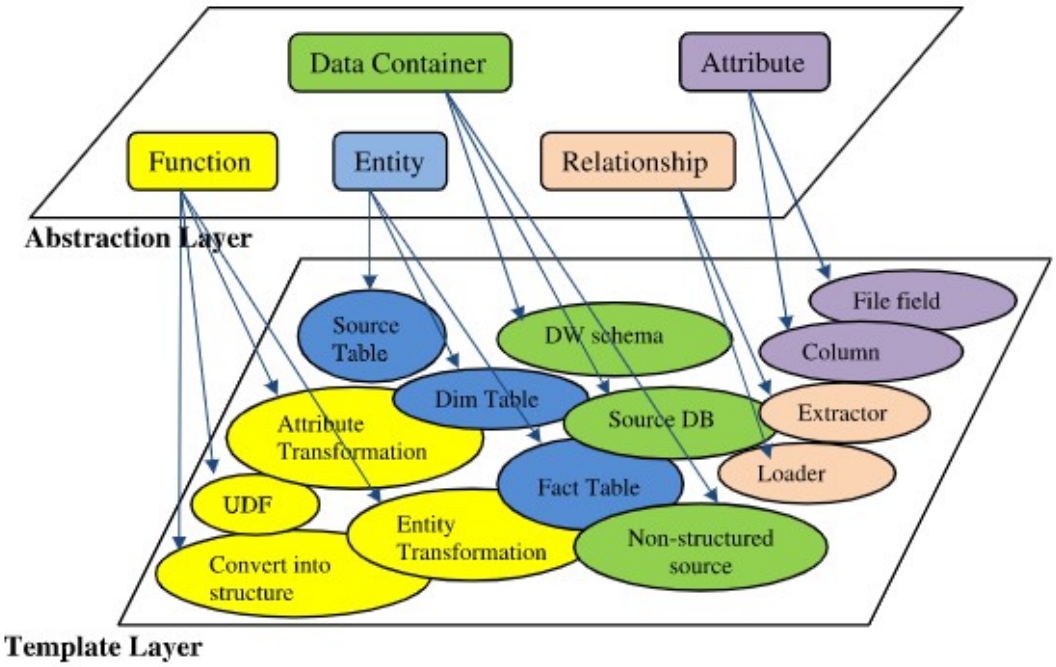

Рисунок 2.14 - Логічні рівні ETL-процесів і схема їх взаємодії

Середовище ETL часто бере на себе відповідальність за зберігання та управління метаданими для всього сховища даних. Немає кращого місця, ніж система ETL, для зберігання та управління метаданими, оскільки середовище повинно знати більшість аспектів даних, щоб нормально функціонувати. Визначають три типи метаданих - ділові, технічні та технологічні - та представляють елементи кожного типу, як вони застосовуються до системи ETL. Першочерговим завданням розробників ETL, є отримання правильних даних з пункту А в пункт В з відповідними перетвореннями у відповідний час. Сучасні засоби ETL можуть значною мірою пришвидшити розробку та

автоматизувати сам процес. У цьому розділі розглядається архітектура задньої кімнати [17]. Розглянемо чотири основні компоненти процесу ETL:

1. витяг;
2. очищення;
3. відповідність;
4. доставка та управління.

Будь-яка система ETL повинна забезпечувати основні можливості середовища розробки, такі як контроль версій та побудова системи виробництва та розробки. Система ETL повинна бути максимально придатною для використання, враховуючи основну складність завдання.

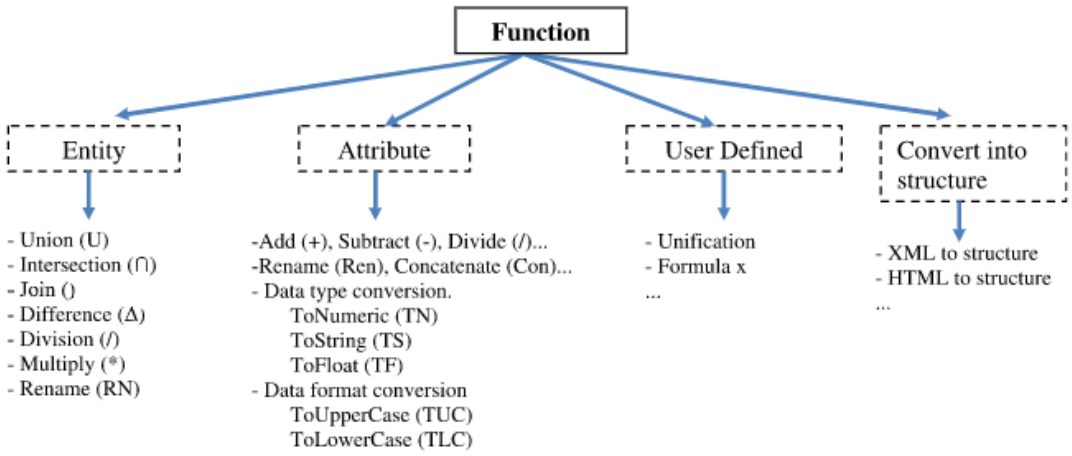

Рисунок 2.15 – Класифікація функцій ETL-системи

Більшість інструментів ETL використовують графічний інтерфейс користувача для визначення завдання ETL. Хороший інтерфейс може скоротити час навчання, пришвидшити розвиток та самодокументовуватися. Звичайно, поганий графічний інтерфейс може зменшити продуктивність, а не покращувати її(рис. 2.15).

Системна документація - це ще одна частина зручності використання. Система ETL повинна дозволяти легко збирати інформацію про процеси, які вони створюють. Ці метадані повинні бути легко доступними для команди та користувачів за допомогою звітів та запитів. Однією з найважливіших

характеристик служб, що підтримують процес ETL, є те, що вони повинні керуватися метаданими. Під цим ми маємо на увазі, що вони повинні черпати зі сховища інформацію про таблиці, стовпці, зіставлення, перетворення, екрани якості даних, завдання та інші компоненти, а не вставляти цю інформацію в інструмент ETL або код SQL, де це майже неможливо знайти і змінити. Дані переміщуються із вихідних систем через процес ETL на сервери презентацій. Послуги ETL, - це інструменти та методи, що використовуються для підтримки процесу ETL. Потік процесу ETL включає чотири основні операції: вилучення даних із джерел, запуск їх через набір процесів очищення та відповідності трансформації, доставка їх на сервер презентацій, а також управління процесом ETL та навколишнім середовищем. Вихідні системи є відправною точкою майже для кожного завдання ETL. У більшості випадків дані повинні надходити з декількох систем, побудованих з кількома сховищами даних, розміщеними на декількох платформах. Зазвичай початкова потреба бізнесу полягає у доступі до основних операційних систем бізнесу, включаючи введення замовлень, виробництво, доставку, обслуговування споживачів та системи бухгалтерського обліку. Інші джерела великої вартості можуть бути зовнішніми для бізнесу, такі як демографічна інформація про клієнтів, цільові списки клієнтів, комерційні сегменти бізнесу та дані про конкурентні продажі. У деяких випадках, особливо у великих організаціях, може не бути прямого доступу до вихідних систем, тому група вихідних систем повинна регулярно надавати відповідні дані [22]. Це часто означає створення файлу витягу та передачу його на склад.

2.6 Взаємодія компонентів Fog обчислень зі сховищем даних

Інфраструктура туману складається з набору пристроїв без обчислювальних можливостей (наприклад, датчиків) та набору ресурсів, що пропонують обчислювальну потужність, ємність сховища тощо (вузли туману, хмарні сервери), які взаємопов’язані, утворюючи складну систему(рис. 2.16).

Існує велика різноманітність характеристик усіх цих пристроїв. Вони можуть стосуватися, серед іншого, потужності центрального процесора, обсягу оперативної пам'яті, ємність диска, пропускної здатності, затримки тощо. З функціональної точки зору, інфраструктура може бути організована пошарово і, як правило, моделюється як пов'язаний графік, де вершини позначають набір пристроїв/ресурсів, а краю означають посилання. Симулятори повинні принаймні підтримувати найпоширеніші типи ресурсів та їх характеристики: зберігання, обчислення та зв’язок. Однією з основних функцій обчислень туману є надання ресурсів зберігання, які знаходяться ближче до кінцевих пристроїв, ніж хмарне сховище.

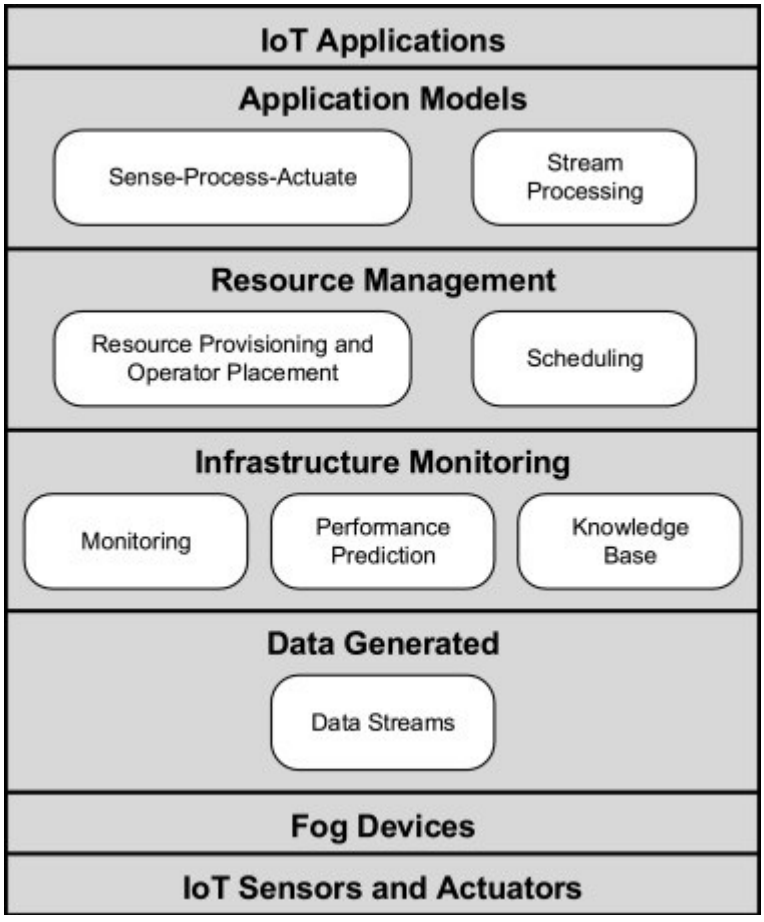

Рисунок 2.16 – Схема типового IoT-додатка

Вимоги до сховища походять від великої кількості кінцевих пристроїв, які можуть поширюватися на великій географічній території, використовуваної різними користувачами під час запуску своїх програм. Багато типів сховища

можна використовувати у вузлах туману для забезпечення надійності та цілісності даних, необхідних системі туману. Користувач потребує додатку вимоги чи обмеження та вартість визначають тип та місткість сховища, тоді як ключовим питанням є розміщення ресурсів зберігання. Вузли туману можуть мати обмежену ємність, і хмарні центри обробки даних можуть використовуватися для обчислень та функцій зберігання [16]. Загалом, обидва вони могли працювати взаємодоповнюючі. У цьому сценарії виникає низка питань: де розмістити дані (оскільки локалізація даних є важливим фактором для швидких та ефективних обчислень), як боротися з величезною кількістю запитів на передачу даних, як керувати ресурсами сховища або як справлятися з неоднорідністю, що походить від різних пристроїв та додатків. Відповідне моделювання та моделювання необхідні для тестування будь-яких задуманих рішень з урахуванням того, наскільки їх поведінка задовольняє фактичні потреби.

3 ПРОЕКТУВАННЯ І РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Таблиці для “сирих” даних

Після етапу визначення вимог наступним питанням є проектування таблиць, які будуть первинним місце призначення для даних, що завантажуються з зовнішніх файлів в систему. Для запобігання помилок під час процесу та швидкої ідентифікації проблем на першому етапі необхідно завантажувати дані в тому вигляді, в якому вони представлені в джерелі не виконуючи жодних явних або неявних перетворень. Для цього використовуються таблиці, що призначені для збереження сирих даних.

Зазвичай вони складаються з тих самих стовпців, що і в файлі, і кожен стовпець є текстовим полем достатнього розміру для збереження даних будь-якого типу, представленого в цьому стовпці. Дані з файлів вилучаються шляхом парсингу, тому існує необхідність у першу чергу визначити максимально можливу кількість інформації, що стосується структури файлу. Цей етап називається “Data Profiling”. В результаті стає відома кількість стовпців з даними, їх максимальний розмір, символи, що відділяють стовпці і рядки. В результаті проектування таблиць, було створено наступні сутності для завантаження “сирих ” даних в систему:

a) Services_raw;
b) Charges_raw;
c) AdjtTran_raw.

Ці сутності є попередниками відповідних стейджингових таблиць і таблиць фактів. на даному єтапі іх структура складається зі стовпців рівного та надлишкового розміру, для того, щоб однозначно розмістити дані в системі, без

помилок на етапі завантаження. Наведемо нижче фрагменти скриптів для створення таблиць для сирих даних.

Лістинг 3.1 – Створення таблиць “сирих” даних

```
CREATE TABLE [dbo].[Services_raw](
	[Service Number] [varchar](100) NULL,
	[AdmissionDxCode] [varchar](100) NULL,
	[AdmissionDxCodeDesc] [varchar](100) NULL,
	[AdmitDate] [varchar](100) NULL,
	[AdmitSourceCode] [varchar](100) NULL,
	[AdmitSourceCodeDesc] [varchar](100) NULL,
	[AdmitTime] [varchar](100) NULL,
	[AdmitTypeCode] [varchar](100) NULL,
	[AdmitTypeCodeDesc] [varchar](100) NULL,
	[CurrentBalance] [varchar](100) NULL,
	[DischargeDate] [varchar](100) NULL,
	[DischargeDisposition] [varchar](100) NULL,
	[DischargeDispositionDesc] [varchar](100) NULL,
	[DischargeMonthandYear] [varchar](100) NULL,
	[DischargeTime] [varchar](100) NULL,
	[EntityCode] [varchar](100) NULL,
	[EntityDesc] [varchar](100) NULL,
	[ERLevelCode] [varchar](100) NULL,
	[FinalBillDate] [varchar](100) NULL,
	[DischargeFiscalQuarter] [varchar](100) NULL,
	[DischargeFiscalYear] [varchar](100) NULL,
	[HospitalServiceCode] [varchar](100) NULL,
	[InitialBillDate] [varchar](100) NULL,
	[LengthofService(LOS)] [varchar](100) NULL,
	[ClientAge] [varchar](100) NULL,
	[ClientClassification] [varchar](100) NULL,
	[ClientFirstName] [varchar](100) NULL,
	[ClientLastName] [varchar](100) NULL,
	[ClientStatus] [varchar](100) NULL,
	[ClientType] [varchar](100) NULL,
	[ClientTypeDesc] [varchar](100) NULL,
	[ClientZip] [varchar](100) NULL,
	[TYPE] [varchar](100) NULL,
	[ServiceLine] [varchar](100) NULL,
	[TotalAdjustments] [varchar](100) NULL,
	[TotalCharges] [varchar](100) NULL,
	[TotalPayments] [varchar](100) NULL,
	[ProductLine] [varchar](100) NULL,
	[ClientBirthDate] [varchar](100) NULL,
	[ClientLocation] [varchar](100) NULL,
	[ClientRace] [varchar](100) NULL,
	[ClientRegistrationLocation] [varchar](100) NULL,
	[Service] [varchar](100) NULL,
	[ServiceGroup] [varchar](100) NULL,
	[ImportFileName] [varchar](100),
	[LoadDate][datetime] default GetDate()
)
GO

CREATE TABLE [dbo].[AdjtTran_raw](
	[ServiceNumber] [varchar](100) NULL,
	[EntityCode] [varchar](100) NULL,
	[EntityDesc] [varchar](100) NULL,
	[FacilityCode] [varchar](100) NULL,
	[FacilityDesc] [varchar](100) NULL,
```

```
	[PaymentDate] [varchar](100) NULL,
	[TxnAmount] [varchar](100) NULL,
	[TxnCategory] [varchar](100) NULL,
	[TxnCode] [varchar](100) NULL,
	[TxnCodeDesc] [varchar](100) NULL,
	[TxnPostDate] [varchar](100) NULL
)
GO

CREATE TABLE [dbo].[Charges_raw](
	[ServiceNumber] [varchar](100) NULL,
	[TotalChargeAmount] [varchar](100) NULL,
	[ChargeCode] [varchar](100) NULL,
	[ChargeCodeDesc] [varchar](100) NULL,
	[ChargeDepartment] [varchar](100) NULL,
	[ChargeDeptDesc] [varchar](100) NULL,
	[ChargePostDate] [varchar](100) NULL,
	[ChargeQuantity] [varchar](100) NULL,
	[ChargeServiceDate] [varchar](100) NULL,
	[Quantity] [varchar](100) NULL,
	[FacilityCode] [varchar](100) NULL,
	[FacilityDesc] [varchar](100) NULL,
	[EntityCode] [varchar](100) NULL,
	[EntityDesc] [varchar](100) NULL,
	[RevenueCode] [varchar](100) NULL,
	[RevenueCodeDesc] [varchar](100) NULL,
	[UnitPrice] [varchar](100) NULL
)

GO
```

Наступним кроком є створення процедур для парсингу даних з файлів або інших джерел. Процедури для цього повинні отримувати дату в імені файла як параметр та парсити джерело за конкретним місцезнаходженням в файловій системі. Для моніторингу коректності роботи процедур по-перше створимо таблиці логів для запису імені файлів, імені цільової таблиці, дати запису в таблиці, дати з імені файлу та кількості строк, що були записані.

Лістинг 3.2 - Створення таблиць для запису логів процедур

```
USE [test_of]

CREATE TABLE [dbo].[LogFactInfo](
	[LogID] [int] IDENTITY(1,1) NOT NULL,
	[TableNameTarget] [varchar](50) NULL,
	[RowsAccumulated] [int] NULL,
	[NewRowsLoaded] [int] NULL,
	[LoadingDate] [datetime] default getDate()
)
GO
CREATE TABLE [dbo].[LogStgInfo](
	[LogID] [int] IDENTITY(1,1) NOT NULL,
	[TableNameSource] [varchar](100) NULL,
	[StgRowCount] [int] NULL,
```

```
	[LoadingDate] [datetime] default getDate(),
	[SourceFileName] [varchar](100) NULL
)
GO

CREATE TABLE [dbo].[LogImportInfo](
	[LogID] [int] IDENTITY(1,1) NOT NULL,
	[ImportFileName] [varchar](100) NULL,
	[FileRowCount] [int] NULL,
	[FileDate] [date] NULL
)
GO
```

Після того, як таблиці логів для кожного етапу було створено, можна приступати до створення процедур для першого етапу. Процедури реалізують завантаження даних до “сирих” таблиць і запис в таблицю логів. Далі наведено приклад процедури парсингу файлу та завантаження в таблицю логів для таблиці Services_raw. Аналогічні процедури створено для завантаження таблиць Charges_raw та AdjtTran_raw.

Лістинг 3.3 – Процедура для запису в Services_raw

```
CREATE procedure [dbo].[sp_services_extract]
	@dt varchar(8)
as
	truncate table Services_raw
	declare @services_raw varchar(max)
	set @services_raw = '
	bulk insert dbo.Services_raw
	from "C:\Users\Oleks\Flat File  Parsing\Services_' + @dt + '.txt" '
+ '
	with(FORMATFILE=''C:\Users\Oleks\Flat File  Parsing\Format
Files\Services.fmt'',
	 firstrow = 2)'
	exec(@services_raw)

	insert into LogImportInfo(ImportFileName, FileDate, FileRowCount)
	select  ImportFileName = 'Services_' + @dt + '.txt'
		 ,FileDate =  DATEADD(day, -1, cast(@dt as date))
		 ,FileRowCount = @@ROWCOUNT

	update dbo.Services_raw set [ImportFileName] = 'Services_' + @dt +
'.txt'
GO
```

3.2 Таблиці вимірів

Після того як було створено таблиці для розміщення сирих даних, відповідно до вимірної моделі необхідно створити та заповнити таблиці вимірів

так, щоб їх можливо було пов’язати з таблицями фактів. Так, окрім таблиці для специфічних даних обов’язковим є створення таких таблиць, як, наприклад вимір для збереження дат - dbo.DicDate.

Лістинг 3.4 – Скрипт створення виміру дат

```
CREATE TABLE [dbo].[DicDate](
	[DateID] [int] NULL,
	[Date] [date] NULL,
	[DayOfMonth] [tinyint] NULL,
	[MonthID] [int] NULL,
	[Month] [tinyint] NULL,
	[MonthName] [varchar](10) NULL,
	[MonthName_Short] [char](3) NULL,
	[WeekID] [int] NULL,
	[WeekOfMonth] [tinyint] NULL,
	[WeekOfYear] [tinyint] NULL,
	[DayOfYear] [smallint] NULL,
	[WeekdayID] [int] NULL,
	[WeekDay] [tinyint] NULL,
	[WeekdayName] [varchar](10) NULL,
	[WeekDayName_Short] [char](3) NULL,
	[Quarter] [tinyint] NULL,
	[QuarterName] [varchar](6) NULL,
	[Year] [int] NULL,
	[FiscalYear] [int] NULL,
	[IsWeekend] [bit] NULL
)

GO
```

Зазвичай логіка завантаження таблиць вимір може відрізнятись в залежності від походження таблиць. Так, для дат було обрано сценарій наповнення, коли кожний цикл ETL-процесу дати, які ще не були оброблені в таблиці записуються до неї(рис. 3.1).

Results | Messages

	DateID	Date	DayOfMonth	MonthID	Month	MonthName	MonthName_Short	WeekID	WeekOfMonth	WeekOfYear	DayOfYear	WeekdayID	WeekDay	WeekdayName	WeekDayName_Short	Quarter	QuarterName	Year	FiscalYear	IsWeekend
1	20101009	2010-10-09	9	201010	10	October	OCT	201041	2	41	282	2014107	7	Saturday	SAT	41	Fourth	2010	2011	1
2	20150214	2015-02-14	14	201502	2	February	FEB	201507	2	7	45	2015707	7	Saturday	SAT	7	First	2015	2015	1
3	20130329	2013-03-29	29	201303	3	March	MAR	201313	5	13	88	2014306	6	Friday	FRI	13	First	2013	2013	0
4	20120826	2012-08-26	26	201208	8	August	AUG	201235	5	35	239	2015501	1	Sunday	SUN	35	Third	2012	2012	1
5	20070127	2007-01-27	27	200701	1	January	JAN	200704	4	4	27	2007407	7	Saturday	SAT	4	First	2007	2007	1
6	19990812	1999-08-12	12	199908	8	August	AUG	199933	2	33	224	2002305	5	Thursday	THU	33	Third	1999	1999	0
7	20190930	2019-09-30	30	201909	9	September	SEP	201940	5	40	273	2023002	2	Monday	MON	40	Third	2019	2019	0
8	20180719	2018-07-19	19	201807	7	July	JUL	201829	3	29	200	2020905	5	Thursday	THU	29	Third	2018	2018	0
9	20171112	2017-11-12	12	201711	11	November	NOV	201746	3	46	316	2021601	1	Sunday	SUN	46	Fourth	2017	2018	1
10	20120518	2012-05-18	18	201205	5	May	MAY	201220	3	20	139	2014006	6	Friday	FRI	20	Second	2012	2012	0
11	20100701	2010-07-01	1	201007	7	July	JUL	201027	1	27	182	2012705	5	Thursday	THU	27	Third	2010	2010	0
12	20151226	2015-12-26	26	201512	12	December	DEC	201552	4	52	360	2020207	7	Saturday	SAT	52	Fourth	2015	2016	1
13	20060915	2006-09-15	15	200609	9	September	SEP	200637	3	37	258	2009706	6	Friday	FRI	37	Third	2006	2006	0
14	20070522	2007-05-22	22	200705	5	May	MAY	200721	4	21	142	2009103	3	Tuesday	TUE	21	Second	2007	2007	0
15	20080505	2008-05-05	5	200805	5	May	MAY	200819	2	19	126	2009902	2	Monday	MON	19	Second	2008	2008	0
16	20090408	2009-04-08	8	200904	4	April	APR	200915	2	15	98	2010504	4	Wednesday	WED	15	Second	2009	2009	0
17	20071208	2007-12-08	8	200712	12	December	DEC	200749	2	49	342	2011907	7	Saturday	SAT	49	Fourth	2007	2008	1
18	20090717	2009-07-17	17	200907	7	July	JUL	200929	3	29	198	2011906	6	Friday	FRI	29	Third	2009	2009	0
19	20100323	2010-03-23	23	201003	3	March	MAR	201013	4	13	82	2011303	3	Tuesday	TUE	13	First	2010	2010	0

Рисунок 3.1 – Вигляд даних після заповнення виміру дат

Зазвичай виміри наповнюються не так часто, як факти, і так як вони зберігають одні і ті ж самі описи ключів і ключі, не існує необхідності кожного разу поновлювати дані в таких таблицях. Коли у вхідних даних існують записи, які не можуть бути пов’язані з виміром, необхідно реалізувати логіку запису таких строк в таблицю вимірів. Тому перед створенням стейджингових таблиць і фактів, необхідно реалізувати процедури для доповнення існуючих даних в вимірах.

Лістинг 3.5 – Скрипт процедури оновлення даних в таблиці вимірів дат

```
with dates_cte(dt)
      as (select distinct t.dt from(
                  select cast(AdmitDate as date) as dt from Services_raw
                  union all
                  select cast(DischargeDate as date) as dt from Services_raw
                  union all
                  select cast(FinalBillDate as date) as dt from Services_raw
                  union all
                  select cast(TxnPostDate as date) as dt from AdjtTran_raw
                  union all
                  select cast(ChargePostDate as date) as dt from Charges_raw
                  union all
                  select cast(ChargeServiceDate as date) as dt from
Charges_raw)t
            where t.dt is not null
      )

      INSERT INTO [dbo].[DicDate]
                     ([DateID]
                     ,[Date]
                     ,[DayOfMonth]
                     ,[MonthID]
                     ,[Month]
                     ,[MonthName]
                     ,[MonthName_Short]
                     ,[WeekID]
                     ,[WeekOfMonth]
                     ,[WeekOfYear]
                     ,[DayOfYear]
                     ,[WeekdayID]
                     ,[WeekDay]
                     ,[WeekdayName]
                     ,[WeekDayName_Short]
                     ,[Quarter]
                     ,[QuarterName]
                     ,[Year]
                     ,[FiscalYear]
                     ,[IsWeekend])
      select
            [DateID] = YEAR(dt) * 10000 + MONTH(dt) * 100 + DAY(dt)
         ,[Date] = dt
         ,[DayOfMonth] = DAY(dt)
         ,[MonthID] = YEAR(dt) * 100 + MONTH(dt)
```

```
        ,[Month] = MONTH(dt)
        ,[MonthName] = DATENAME(mm, dt)
        ,[MonthName_Short] = UPPER(LEFT(DATENAME(mm, dt), 3))
        ,[WeekID] = YEAR(dt) * 100 + DATEPART(wk, dt)
        ,[WeekOfMonth] = DATEPART(WEEK, dt) - DATEPART(WEEK, DATEADD(MM,
DATEDIFF(MM, 0, dt), 0)) + 1
        ,[WeekOfYear] = DATEPART(wk, dt)
        ,[DayOfYear] = DATENAME(dy, dt)
        ,[WeekdayID] = YEAR(dt) * 1000 + DATEPART(wk, dt) *10 + DATEPART(dw,
dt)
        ,[WeekDay] = DATEPART(dw, dt)
        ,[WeekdayName] = DATENAME(dw,dt)
        ,[WeekDayName_Short] = UPPER(LEFT(DATENAME(dw, dt), 3))
        ,[Quarter] = DATEPART(q, dt)
        ,[QuarterName] = CASE WHEN DATENAME(qq, dt) = 1 THEN 'First'
                                        WHEN DATENAME(qq, dt) = 2 THEN
'second'
                                        WHEN DATENAME(qq, dt) = 3 THEN
'third'
                                        WHEN DATENAME(qq, dt) = 4 THEN
'fourth'
                                    END
        ,[Year] = YEAR(dt)
        ,[FiscalYear] = CASE WHEN MONTH(dt) >= 10 THEN YEAR(dt) + 1 ELSE
YEAR(dt)
                                END
        ,[IsWeekend] = CASE WHEN DATENAME(dw, dt) = 'Sunday'
                                        OR DATENAME(dw, dt) = 'Saturday' THEN
1 ELSE 0
                               END
        from dates_cte t
        left join [DicDate] dtcode
        on t.dt = dtcode.[Date]
        where dtcode.[DateID] is null ;
```

3.3 Стейджингові таблиці

Етап завантаження даних в стейджингові таблиці - це момент, коли дані, що зберігаються в загальному вигляді в “сирих” таблицях перетворюються на конкретні дані для подальшого завантаження в таблиці фактів. Від простого зберігання на цьому єтапі ми переходимо до встановлення взаємозв’язків з таблицями вимірів, обробки пустих значень та реалізації явних перетворень для окремих типів. Головна задача - до кожного ключа підібрати відповідний в таблиці виміру, створити ідентифікатори для дат, та виконати додаткові обчислення.

Лістинг 3.6 – Завантаження стейджингової таблиці Services_stg

```
CREATE procedure [dbo].[sp_Services_stg_loading]
```

```
as
      truncate table [Services_stg]

      INSERT INTO [dbo].[Services_stg]
      SELECT srv.[ServiceNumber]
             ,DATEDIFF(day, cast(srv.DischargeDate as date), cast(l.FileDate as
date))
             ,0--ISNULL(ag.ageID, 0)
             ,cast(l.[FileDate] as date)
             ,ISNULL(YEAR(cast(l.[FileDate] as date)) * 10000 + MONTH(cast(l.
[FileDate] as date)) * 100 + DAY(cast(l.[FileDate] as date)), 19000101)
             , srv.[AdmissionDxCode]
             ,ISNULL(dx.DxCodeID, 0)
             ,srv.[AdmissionDxCodeDesc]
             ,cast([AdmitDate]  as date)
             ,ISNULL(YEAR(cast(srv.[AdmitDate] as date)) * 10000 +
MONTH(cast(srv.[AdmitDate] as date)) * 100 + DAY(cast(srv.[AdmitDate] as date)),
19000101)
             ,srv.[AdmitSourceCode]
             ,srv.[AdmitSourceCodeDesc]
             ,srv.[AdmitTime]
             ,srv.[AdmitTypeCode]
             ,srv.[AdmitTypeCodeDesc]
             ,cast(srv.[CurrentBalance] as money)
             ,tr.[ThresholdID]
             ,srv.([DischargeDate] as date)
             ,ISNULL(YEAR(cast(srv.[DischargeDate] as date)) * 10000 +
MONTH(cast(srv.[DischargeDate] as date)) * 100 + DAY(cast(srv.[DischargeDate] as
date)), 19000101)
             ,srv.[DischargeDisposition]
             ,srv.[DischargeDispositionDesc]
             ,cast(srv.[DischargeTime] as time)
             ,srv.[EntityCode]
             ,srv.[EntityDesc]
             ,srv.[ERLevelCode]
             ,srv.[FacilityCode]
             ,srv.[FacilityDesc]
             ,cast([FinalBillDate] as date)
             ,ISNULL(YEAR(cast(srv.[FinalBillDate] as date)) * 10000 +
MONTH(cast(srv.[FinalBillDate] as date)) * 100 + DAY(cast(srv.[FinalBillDate] as
date)), 19000101)
             ,cast(srv.[InitialBillDate] as date)
             ,srv.[LengthofService(LOS)]
             ,srv.[Observation]
             ,srv.[ClientAge]
             ,srv.[ClientClassification]
             ,srv.[ClientFirstName]
             ,srv.[ClientLastName]
             ,srv.[ClientStatus]
             ,srv.[ClientType]
             ,ISNULL(pt.ClientTypeID, 0)
             ,srv.[ClientTypeDesc]
             ,srv.[ClientZip]
             ,srv.[TYPE]
             ,srv.[ServiceLine]
             ,cast(srv.[TotalAdjustments] as money)
             ,cast(srv.[TotalCharges] as money)
             ,cast(srv.[TotalPayments] as money)
             ,srv.[ProductLine]
             ,srv.[Location]
             ,srv.[RegistrationLocation]
             ,srv.[Service]
             ,srv.[ServiceGroup]
```

```
              FROM [dbo].[Services_raw] srv
        left join dbo.DicDxCode dx on srv.[AdmissionDxCode] = dx.DxCode
        left join  [dbo].[DicClientType] pt on srv.[ClientType] = pt.ClientType
        left join [dbo].[DicBalanceThreshold] tr on srv.CurrentBalance between
tr.MinValue and tr.MaxValue
        --left join [dbo].[AgingbyDsch] ag on DATEDIFF(day,
cast(srv.DischargeDate as date), cast(srv.PostDate as date)) between ag.ageMin
and ag.ageMax
        left join (select distinct ImportFileName, FileDate from [dbo].
[LogImportInfo]) l on srv.ImportFileName = l.ImportFileName
        insert into [dbo].[LogStgInfo]([TableNameSource], StgRowCount,
LoadingDate, SourceFileName)
        select top 1 [TableNameSource] = '[dbo].[Services_stg]',
                       [StgRowCount] = @@ROWCOUNT,
                     [LoadingDate] = getDate(),
                     [SourceFileName] = ImportFileName
        from [dbo].[LogImportInfo] where  ImportFileName like 'Services_%'


        update dbo.Services_stg
        set [AgingID] = ISNULL(ag.ageID, 0)
        from dbo.Services_stg srv
        left join [dbo].[AgingbyDsch] ag on DATEDIFF(day, cast(srv.DischargeDate
as date), cast(srv.PostDate as date)) between ag.ageMin and ag.ageMax
GO
```

За аналогією було створено таблиці Charges_stg і AdjtTran_stg.

3.4 Таблиці фактів

Таблиці фактів за своєю структурою ідентичні стейджинговим таблицям. Їх призначення полягає в реалізації логіки завантаження таблиць, яка відрізняється в залежності від сутності таблиць і призначення даних. Так таблиці транзакції і рахунків потребують логіки завантаження append, дані з таблиці Services - append/replace. Сутність методу append полягає в простому додаванні строк з даними в таблицю. Append/replace - дещо складніший принцип для реалізації тому що потрібно розуміти які саме дані залишати в таблиці до завантаження наступного набору даних, а які видаляти. В нашому випадку будемо залишати ті записи, яких не буде в наступному датасеті, що будет означати, що конкретний випадок вичерпаний або баланс став нульовим. Залишати ж ті записи, що будуть прислані наступного ж дня не має потреби.

Лістинг 3.7 – Релізація логіки завантаження таблиці фактів Services_fact

```
CREATE procedure [dbo].[sp_Services_fact_loading]
```

```
as
      declare @rcnt int

      delete FROM Services_fact
      WHERE ServiceNumber IN(select ServiceNumber From Services_stg) and
year(PostDate) *  100 + month(PostDate) IN(select year(PostDate) *  100 +
month(PostDate) From Services_stg)

      update Services_fact
      set PostDate = (Select top 1 PostDate from Services_stg)
      where year(PostDate) *  100 + month(PostDate) IN(select
year(PostDate) *  100 + month(PostDate) From Services_stg)

      set @rcnt = (select count(*) from Services_fact)

---Скрипт не включає частину Insert into і Select from…..

      Insert Into LogFactInfo
(TableNameTarget,RowsAccumulated,NewRowsLoaded,LoadingDate)
      Select TableNameTarget = 'Services_fact',
               RowsAccumulated = @rcnt,
               NewRowsLoaded = @@ROWCOUNT,
               LoadingDate = GetDate()
GO
```

Завантаження в таблиці фактів транзакцій і рахунків реалізується лише повною заливкою даних зі стейджингової таблиці і попереднью очисткою таблиці фактів. Після створення таблиці фактів було прийнято рішення розбити ітогову таблицю фактів на партиції відповідно до поля PostDate за місяцями, для підвищення ефективності виконання процесів пов’язаних за цією таблицею фактів(рис. 3.2).

Рисунок 3.2 – Властивоці партиціонованої таблиці фактів

3.5 Запуск ETL - процесу і перегляд таблиць логів

Після виконання всіх попередніх етапів і створення всіх необхідних сутностей - таблиць і процедур можливо переходити до перевірки роботи системи. Для цього необхідно викликати процедури кожного з етапів - вилучення даних, завантаження в стейджингові таблиці і перетворення і завантаження в таблиці фактів. На виході маємо 3 таблиці “сирих” даних, 3 таблиці логів, для кожного з етапів, 3 стейджингові таблиці, 3 таблиці фактів відповідної структури і 6 таблиць вимірів, включаючи дати. Після запуску процедур і успішної роботи системи на трьох наборах даних за 3 різні дати, отримано результат у таблиці логів для таблиці фактів(рис. 3.3).

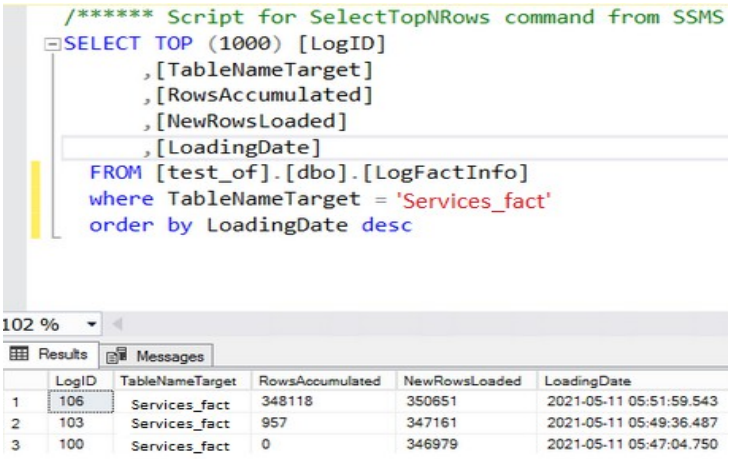

Рисунок 3.3 – Зміст таблиці логів після виконання процедур

Лістинг 3.8 – виклик процедур для запуску ETL - процесів

```
----Reading data from files and loading into raw tables
exec [dbo].[sp_services_extract] '20200323'
exec [dbo].[sp_adjttran_extract] '20200323'
exec [dbo].[sp_charges_extract] '20200323'

exec [dbo].[dic_update] --Loading values into dimensional tables
which exists in source file and not in current dictionaries

--------Loading data from raw to staging tables. Mapping keys and
doing explicit conversion for date and
money--------------------------------
exec [dbo].[sp_services_stg_loading]
exec [dbo].[sp_adjttran_stg_loading]
exec [dbo].[sp_charges_stg_loading]
------Loading data into fact tables-----
exec [dbo].[sp_services_fact_loading]
exec [dbo].[sp_adjttran_fact_loading]
exec [dbo].[sp_charges_fact_loading]
```

3.5. Аналіз можливостей використання Fog-вузлів в системі

Сьогодні розумні системи використовують безліч розумних датчиків, крайових пристроїв, крайових шлюзів та вузлів, щоб збирати дані для прийняття рішень у реальному часі та аналізу. Як і в інших парадигмах розподіленої обчислювальної техніки, в екосистемі крайових та туманних обчислень необхідно подолати кілька викликів, щоб дозволити кінцевим споживачам розумних міст, провайдерам послуг та провайдерам інфраструктури скористатися послугами, що надаються крайовими та туманними серверами. Чотири найважливіші проблеми та відкриті питання досліджень - це безпека, конфіденційність, сумісність та характеристика додатків інтелектуального міста.

При проектуванні мережі топологія була розділена на дві основні підмережі: одна, де були підключені пристрої IoT для загального управління IoT. Всі компоненти мережі були з'єднані між собою центральним основним маршрутизатором, розміщеним у мережі управління IoT. Цей макет, що не є надлишковим, може бути непридатним для використання в реальному житті, проте він допоміг спростити вправу Cisco Packet Tracer(рис. 3.5.1). Друга найпростіша мережа є корпоративною мережею LAN. Мережа складається з основного маршрутизатора, підключеного до центрального маршрутизатора, та комутатора місцевого офісу. ПК та офісний DHCP-сервер також були підключені до локального комутатора. За дизайном мережі жоден офісний ПК не може перейти на домашню сторінку IoT або на будь-який з пристроїв IoT. Задача насправді була ізолювати та обмежити доступ до пристрою IoT для керування лише авторизованим користувачем, фізично підключеним до мережі управління IoT. Хоча туманні обчислення або крайова аналітика можуть стосуватися виключно аналітики на пристроях, які знаходяться недалеко від краю мережі, архітектура туману виконуватиме аналітику на будь-чому, від центру мережі до краю. Одним із варіантів використання обчислень туману є розумна система складу підприємств, або інших приміщень, яка може

змінювати свої сигнали на основі спостереження за інформацією, що отримується з датчиків, щоб оптимізувати роботу підприємства. Дані також можуть надсилатися в хмару для більш тривалої аналітики. Інші випадки можуть включати фізичну безпеку системи та кібербезпеку. Хоча додавання обчислень туману до мережі IoT, здається, додає складності, ця складність іноді необхідна.

Взаємодія у системах, що підтримують туман, є ще однією важливою проблемою, яку ці системи потребують надто великого рівня. Деякі занепокоєння виникають через відсутність взаємодії між розгорнутим обладнанням та пристроями. Цими проблемами є:

а) Труднощі з інтеграцією та розгортанням пристроїв та обладнання, виготовлених різними виробниками, із різними типами роз’ємів, використанням різних форматів даних та підтримкою різних комунікаційних протоколів. витягувати та передавати інформацію з /на ці пристрої;

б) Відсутність загальних методів та підходів до тестування інтерфейсів прикладного програмування (API) цих пристроїв;

в) Складність у використанні програмного забезпечення для захисту, пропонованого третіми сторонами для захисту пристроїв. Взаємодія у IoT вирішується за допомогою розгортання проміжних компонентів, що підтримують кілька протоколів, для вирішення вищезазначених питань. Посередницький компонент, такий як шлюз IoT, зазвичай виконує кілька важливих функцій - від перекладу протоколів до об'єднання, фільтрації, шифрування, обробки та управління даними. У деяких випадках використання обчислень туману вирішують недоліки хмарних моделей, які мають серйозні проблеми із затримкою, пропускною здатністю мережі, географічною спрямованістю, надійністю та безпекою. Пристрої можуть залишатися можуть передавати дані до централізованих або розподілених програм для туману, які знаходяться поруч з ними, і, отже, можуть відповідати вимогам, яким не можуть відповідати далекі централізовані програми.

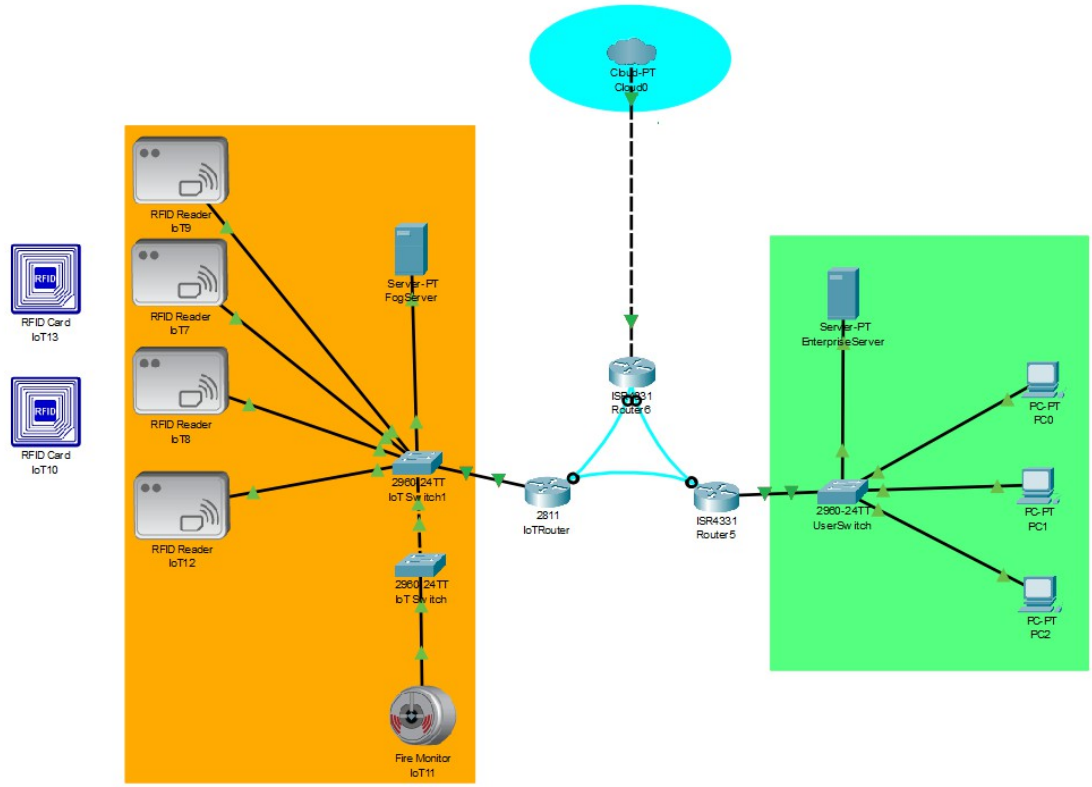

Рисунок 3.4 – Змодельована мережа в середовищі Cisco Packet Tracer

На цьому етапі злиття даних, отриманих від різних пристроїв та датчиків на об'єктах розумного міста, може використовувати відомі підходи до інтеграції даних, включаючи поширення даних, консолідацію даних, об'єднання даних та інтеграцію семантичних даних. Крайові шлюзи зазвичай виконують цю задачу збору даних. Поширення даних - це передача зібраних даних з джерел даних на цільові сервери. Консолідація даних - це процес інтеграції даних, отриманих з декількох джерел даних, і постійне їх зберігання в одному сховищі даних. Інтеграція семантичних даних включає контрольовані словники, які вимагають стандартизованої термінології для представлення елементів даних у сховищах даних.

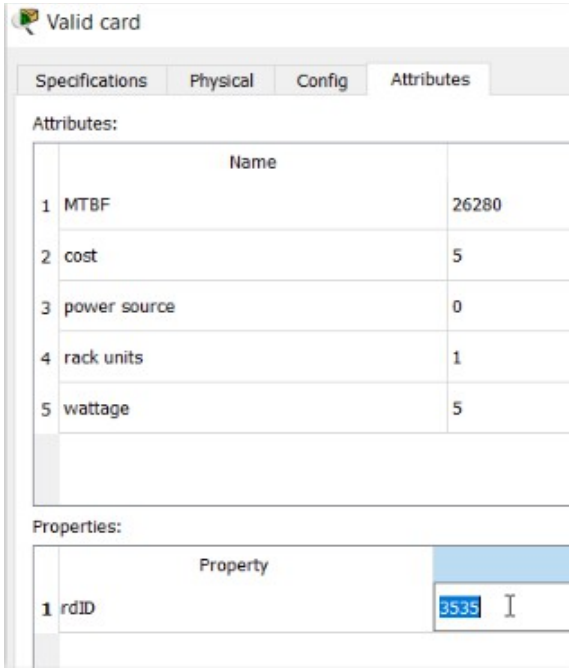

Рисунок 3.5 – Модифікація атрибуту мітки

Прикладами контрольованих словників є різні онтології, розроблені для розумних міст. Для обробки потоків даних в основному використовуються механізми потокової обробки, такі як ApacheFlume, Apache Flink, Apache Storm, Amazon Kinesis та Apache SparkStreaming.

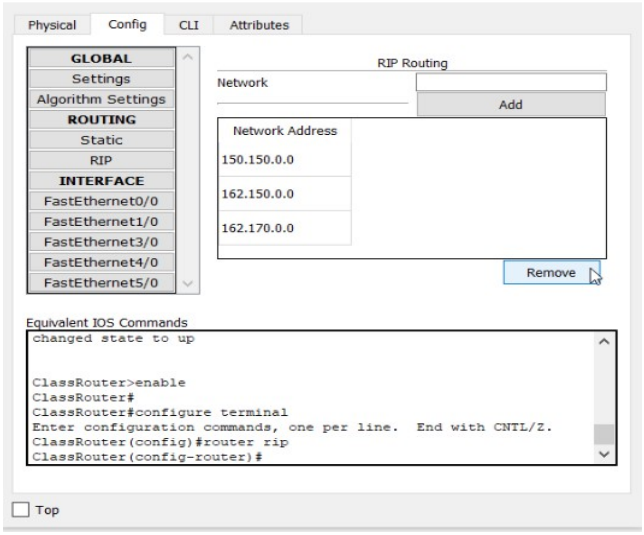

Рисунок 3.6 – Налаштування одного роутера

Ці потокові двигуни зазвичай підтримують багато бібліотек машинного навчання, таких як Tensorflow, Keras, PyTorch, Scikit-learn та багато іншого, щоб зрозуміти неструктуровані дані.

ВИСНОВКИ

В рамках дослідницької роботи були розглянуті поняття Туманних обчислень, Сховищ даних, можливості взаємодії пристроїв “Інтернету речей” з сучасними системами обліку та управління. Були досліджені існуючі проблеми, що виникають під час проектування корпоративних сховищ даних, шляхи їх вирішення, а також актуальні підходи до створення аналітичних систем і систем збору обробки і завантаження даних. Поставлені цілі та задачі були виконані в повному обсязі, а саме:

а) спроектована і реалізована система, що представляє собою корпоративне сховище даних;

б) в рамках реалізації КСД проаналізовані вимоги і виконане моделювання процесу ETL.

в) виконано моделювання системи з використанням Fog-обчислень

г) були виконані перевірки роботоспроможності системи, що підтвердило повну роботоспроможність сховища даних і можливість його використання в подальшому.

Для первинного проектування сховища даних було використано Cloudera, Hive, і середовище CISCO Packet Tracer для моделювання поведінки компонентів IoT. В результаті роботи було представлено функціонуючу модель корпоративного сховища даних, яке виконує задачі вилучення інформації з обраних джерел, очистки і приведення їх до відповідного рівня гранулярності і стандарту а також завантаження в цільові сутності в БД. Були запропоновані напрями розвитку створеної системи для забезпечення повної відповідності світовим стандартам подібних систем. Розглянуті аспекти покращення та розширення функціоналу системи, а саме: безпеки, гнучкості, масштабування та швидкості роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lucas N. Building a Test Bed for Simulation Analysis for the Internet of Things / Lucas Novelli., 2008. - 26c.
2. Jinhui Z. A fog computing model for implementing motion guide to visually impaired / Jinhui Zhu., 2020. - 91c.
3. Said E.K. Efficient and Dynamic Scaling of Fog Nodes for IoT Devices / Said El Kafhai., 2017. - 14c.
4. Andras M. A survey and taxonomy of simulation environments modelling fog computing / Andras Markus., 2020. - 26c.
5. Fog Computing Application for Effective Fronthaul Management in Fifth Generation Networks [Електронний ресурс] // IEEE. - 2020. - Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/8658911(дата звернення: 10.02.2021). - Назва з екрана
6. Fernanda F. Integrating an IoT Application Middleware with a Fog and Edge Computing Simulator / Fernanda Fama., 2020. - 30c.
7. Definition of Enterprise Resource Planning (ERP) [Електронний ресурс] // Oracle. - 2020. Режим доступу до ресурсу: https://www.oracle.com/erp/what-is-erp/ (дата звернення: 28.03.2021). - Назва з екрана
8. Ralph K. The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data / Ralph Kimball., 2019.
9. Ralph K. The Data Warehouse Concept / Ralph Kimball., 2019.
10. ERP - планування ресурсів підприємства [Електронний ресурс] // it - 2020. Режим доступу до ресурсу:https://www.it.ua/knowledge-base/technology-innovation/enterprise-resource-planning-erp(дата звернення: 10.02.2021). - Назва з екрана
11. 3 Ways to Build An ETL Process with Examples [Електронний ресурс] // IEEE. - 2020. - Режим доступу до ресурсу: https://panoply.io/data-warehouse-guide/3-ways-to-build-an-etl-process (дата звернення: 10.02.2021). - Назва з екрана

12. David P.A. A comparative analysis of simulators for the Cloud to Fog continuum / David Perez Abreu., 2018. - 33c.
13. Opnet Simulation manual [Електронний ресурс] // IEEE. - 2020. - Режим доступу до ресурсу: https://doc.omnetpp.org/omnetpp/SimulationManual (дата звернення: 10.02.2021). - Назва з екрана
14. A comparative node evaluation model for highly heterogeneous massive-scale Internet of Things-Mist networks [Електронний ресурс] // IEEE. - 2020. - Режим доступу до ресурсу: https://onlinelibrary.wiley.com/doi/full/10.1002/ett.3924 (дата звернення: 10.02.2021). - Назва з екрана
15. J. Son, A.V. Dastjerdi, R.N. Calheiros, X. Ji, Y. Yoon, R. Buyya, CloudSimSDN - GitHub Repository[Електронний ресурс] 2019, (https://github.com/Cloudslab/cloudsimsdn).(дата звернення: 10.02.2021). - Назва з екрана
16. J. Son, A.V. Dastjerdi, R.N. Calheiros, R. Buyya Sla-aware and energy-efficient dynamic overbooking in sdn-based cloud data centers // IEEE Trans. Sustainable Comput., 2017. - 89c.
17. C. Cicconetti, M. Conti, A. Passarella An architectural framework for serverless edge computing: design and emulation tools / IEEE International Conference on Cloud Computing Technology and Science (CloudCom)., 2018, 55c.
18. Сукачев Д. В. Инфракрасные датчики движения и присутствия[Електронний ресурс] / Д. В. Сукачев // Энергосовет. – 2010. – Режим доступу до ресурсу: http://www.energosovet.ru/bul_stat.php?idd=43 (дата звернення: 18.11.2020). – Назва з екрана.
19. Garcia M. L. Design and Evaluation of Physical Protection Systems / MaryLynn Garcia., 2007. – 376 с.
20. Halonen T. GSM, GPRS and EDGE Performance: Evolution Towards3G/UMTS / T. Halonen, R. Javier, M. Juan., 2004. – 656 с.

21. Is IoT Security a Ticking Time Bomb? [Електронний ресурс] //SecurityIntelligence. – 2016. – Режим доступу доресурсу:https://securityintelligence.com/is-iot-security-a-ticking-bomb/ (дата звернення:06.12.2020). – Назва з екрана.
22. Stuckmann P. The GSM Evolution: Mobile Packet Data Services / PeterStuckmann., 2003. – 256с.