Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Підтримка методів інтелектуального аналізу даних засобами реляційних систем
(тема)

Виконав:
здобувач другого року навчання,
групи ДСм-23-1
Кесарєва М.О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Науки про дані (Data Science)
(повна назва спеціалізації)
Керівник проф. В.О. Філатов
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ О.В. Золотухін
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

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

Освітня програма Науки про дані (Data Science)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кесарєвій Марії Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи Підтримка методів інтелектуального аналізу даних засобами реляційних систем

затверджена наказом університету від 22 листопада 20 24 р. № 1238Ст

2. Термін подання студентом роботи до екзаменаційної комісії 21 січня 20 25 р.

3. Вихідні дані до роботи аналіз можливостей реляційних баз даних (РБД) у підтримці методів інтелектуального аналізу даних (data mining), таких як кластеризація, класифікація, прогнозування, асоціативні правила, створення демонстративної програми з асоціацією, приклад роботи асоціативного аналізу у фармацевтичній сфері.

4. Перелік питань, що потрібно опрацювати в роботі:

1) Методи інтелектуального аналізу баз даних;

2) Огляд структурованих моделей даних;

3) Підтримка методів інтелектуального аналізу даних засобами PC.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	виконано
2	Вивчення предметної області дослідження	01.11.2024	виконано
3	Проектування, розробка ПЗ	01.11.2024 А18.12.2024	виконано
4	Написання пояснювальної записки	01.11.2024 – 18.12.2024	виконано
5	Перевірка пояснювальної записки керівником	18.12.2024	виконано
6	Нормоконтроль	23.12.2024	
7	Рецензування	25.12.2024	
8	Захист кваліфікаційної роботи	21.01.2025	

Дата видачі завдання 25 листопада 20 24 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. В.О. Філатов
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 86 с., 18 рис., 1 дод., 27 джерел.

АПТЕКА, АСОЦІАЦІЯ, БАЗА ДАНИХ, ГРАФОВІ МОДЕЛІ, ДОКУМЕНТНІ МОДЕЛІ, ІНТЕЛЕКТУАЛЬНІЙ АНАЛІЗ, ІЄРАРХІЧНІ МОДЕЛІ, КЛАСИФІКАЦІЯ, КЛАСТЕРИЗАЦІЯ, КЛЮЧ-ЗНАЧЕННЯ МОДЕЛІ, МАРКЕТИНГ, МЕРЕЖЕВІ МОДЕЛІ, НЕЧІТКІ ДАНІ, НЕЧІТКІ МОДЕЛІ, ОБ'ЄКТНО-ОРІЄНТОВАНІ МОДЕЛІ, РЕЛЯЦІЙНІ МОДЕЛІ, РСУБД, СТРУКТУРОВАНІ МОДЕЛІ ДАНИХ, СУБД, HOLAP, MOLAP, OLAP, ROLAP.

Об'єкт дослідження – реляційні системи управління базами даних, які використовуються для зберігання, управління та аналізу даних, а також їх інтеграція із методами інтелектуального аналізу даних.

Предмет дослідження – методи інтелектуального аналізу даних, зокрема їх реалізація та підтримка засобами реляційних систем управління базами даних.

Мета роботи – аналіз можливостей реляційних систем управління базами даних для підтримки методів інтелектуального аналізу даних, а також розробка рекомендацій щодо їх ефективного використання для вирішення завдань аналізу великих обсягів даних.

Методи дослідження – вивчення наукових праць, методологій і технологій, що стосуються РСУБД і інтелектуального аналізу даних, оцінка ефективності різних РСУБД у виконанні задач інтелектуального аналізу, оцінка точності, продуктивності та масштабованості рішень.

ABSTRACT

Master’s thesis contains: 86 pp., 18 fig., 1 ann., 27 references.

ASSOCIATION, CLASSIFICATION, CLUSTERING, DATABASE, DBMS, DOCUMENT MODELS, FUZZY DATA, FUZZY MODELS, GRAPH MODELS, HIERARCHICAL MODELS, HOLAP, INTELLIGENT ANALYSIS, KEY-VALUE MODELS, MARKETING, MOLAP, NETWORKS MODELS, OBJECT-ORIENTED MODELS, OLAP, PHARMACY, RELATIONAL MODELS, ROLAP, STRUCTURED DATA MODELS.

The object of research is relational database management systems used for data storage, management and analysis, as well as their integration with methods of intelligent data analysis.

The subject of research is methods of intellectual data analysis, in particular their implementation and support by means of relational database management systems.

The purpose of the work is to analyze the capabilities of relational database management systems to support methods of intelligent data analysis, as well as to develop recommendations for their effective use to solve the problems of analyzing large volumes of data.

Research methods – study of scientific works, methodologies and technologies related to RDBMS and intelligent data analysis, evaluation of the effectiveness of various RDBMS in performing tasks of intellectual analysis, assessment of accuracy, productivity and scalability of solutions.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 7
Вступ 8
1 Методи інтелектуального аналізу баз даних 9
1.1 Загальні відомості 9
1.2 Кластеризація 13
1.3 Класифікація 19
1.4 Асоціація 23
1.5 OLAP 29
2 Огляд структурованих моделей даних 37
2.1 Загальні відомості 37
2.2 Ієрархічні моделі 38
2.3 Мережеві моделі 41
2.4 Реляційні моделі 44
2.5 Об'єктно-орієнтовані моделі 51
2.6 Документні моделі 56
2.7 Ключ-значення моделі 59
2.8 Графові моделі 63
2.9 Нечіткі моделі в структурованих базах даних 66
3 Підтримка методів інтелектуального аналізу даних засобами РС 68
3.1 Потреби до даних для аналізу 68
3.2 Підтримка методів засобами РС 70
3.3 Застосування у фармацевтичній галузі 74
Висновки 79
Перелік джерел посилання 82
Додаток А Відомість кваліфікаційної роботи 86

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

РБД – реляційна база даних;

РСУБД – реляційні системи управління базами даних;

СУБД – системи управління базами даних;

DB – database – база даних;

DBMS – Database Management Systems – системи управління базами даних;

DOLAP – Desktop OLAP – десктопний OLAP;

HOLAP – Hybrid OLAP – гібридний OLAP;

MOLAP – Mobile OLAP – мобільний OLAP;

MOLAP – Multidimensional OLAP – багатовимірний OLAP;

OLAP – Online Analytical Processing – онлайн аналітична обробка;

RDB – Relational Database – реляційна база даних;

RDBMS – Relational Database Management Systems – реляційні системи управління базами даних;

ROLAP – Relational OLAP – реляційний OLAP;

SOLAP – Spatial OLAP – просторовий OLAP;

WOLAP – Web-Enabled OLAP – веб-орієнтований OLAP.

ВСТУП

Сучасний ринок, та сучасні інформаційні системи характеризуються високою динамікою розвитку, високим рівнем конкуренції та складністю умов використання. Для забезпечення ефективного прийняття рішень у багатофакторних умовах необхідна адаптація методів аналізу даних до специфіки реляційних систем управління базами даних.

Методи інтелектуального аналізу даних дозволяють обробляти великі обсяги інформації, включаючи нечіткі та неповні дані, які можуть виникати в різних сферах діяльності.

Використання нечітких моделей є актуальним у контексті інтеграції інтелектуального аналізу даних із реляційними базами.

Нечіткі множини дозволяють моделювати невизначеність і розмитість інформації, що сприяє більш точному аналізу та прийняттю рішень.

Нечіткі моделі надають можливість обробляти та інтерпретувати дані в умовах невизначеності, що є особливо актуальним у завданнях продаж (у даному випадку, у фармацевтичній сфері), де вподобання споживачів, їхні фінансові можливості, очікування від продукту тощо можуть бути представлені як нечіткі множини.

Загалом, огляд структурованих моделей даних допомагає краще зрозуміти, як обробляти і зберігати ці дані, а використання нечітких моделей у структурованих базах даних дозволяє зберігати нечіткі оцінки та запити, що сприяє більш гнучкому та адаптивному прийняттю рішень.

Таким чином, підтримка методів інтелектуального аналізу даних у реляційних системах сприяє розвитку адаптивних і масштабованих рішень для обробки складних і неоднозначних даних.

1 МЕТОДИ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ БАЗ ДАНИХ

1.1 Загальні відомості

Інтелектуальний аналіз баз даних, або, як його часто називають – Data Mining – це процес аналізу великих обсягів даних з метою виявлення прихованих закономірностей, трендів та корисної інформації. Застосовуючи різноманітні методи, такі як статистичний аналіз, машинне навчання та штучний інтелект, компанії можуть використовувати ці знання для прийняття більш оптимальних рішень, прогнозування можливих подій та ризиків та оптимізації ручних процесів. Це підрозділ аналізу даних, який зосереджується на виявленні закономірностей у великих наборах даних.

Інтелектуальний аналіз даних має широке застосування у різних сферах завдяки своїй здатності виявляти приховані закономірності та тенденції у великих масивах даних. У бізнесі його використовують для аналізу поведінки клієнтів, наприклад, для сегментації на основі купівельних звичок, що дозволяє створювати персоналізовані маркетингові кампанії. Також, це допомагає прогнозувати продажі, виявляти шахрайство у транзакціях чи розробляти рекомендаційні системи, наприклад у Amazon або Netflix, інших великих мережах.

У фінансовій галузі методи інтелектуального аналізу даних застосовуються для управління ризиками, наприклад, оцінки кредитоспроможності клієнтів або прогнозування можливої несплати. Також, це ефективний інструмент для аналізу ринкових трендів і автоматизації процесів торгівлі на біржі. У цій сфері інтелектуальний аналіз даних допомагає у виявленні шахрайства та оптимізації бюджетного планування.

В охороні здоров’я методи інтелектуального аналізу даних активно використовуються для діагностики захворювань. Аналізуючи великі обсяги медичних даних, можна виявити закономірності, які допомагають у

ранньому виявленні захворювань або в прогнозуванні ризиків. Це також сприяє вибору індивідуалізованих методів лікування для пацієнтів. Крім того, інтелектуальний аналіз даних застосовується для відстеження поширення інфекційних захворювань та ефективного управління ресурсами в медицині.

У транспорті й логістиці його застосовують для оптимізації маршрутів доставки, прогнозування попиту на транспортні послуги та аналізу даних про дорожній рух для покращення інфраструктури. У виробництві інтелектуальний аналіз даних допомагає виявляти дефекти в продукції, прогнозувати потребу в сировині чи технічному обслуговуванні обладнання.

В освіті та науці ці методи застосовуються для аналізу навчальних досягнень студентів, виявлення факторів, що впливають на успішність, а також для обробки великих масивів наукових даних з метою виявлення нових трендів.

Таким чином, інтелектуальний аналіз даних є універсальним інструментом, який знаходить застосування в багатьох галузях, де необхідно аналізувати великі обсяги даних для прийняття обґрунтованих рішень.

Головними етапами інтелектуального аналізу даних є:

- отримання даних;
- підготовка даних;
- дослідження даних;
- вибір предикторів;
- вибір моделі;
- навчання моделі;
- оцінка моделі;
- впровадження моделі;
- підтримка моделі.

Після визначення проблеми та того, які ключові питання потрібно вирішити за допомогою цього аналізу, відбувається збір даних із релевантних доступних джерел: баз даних, інших файлів, API чи онлайн-ресурсів. Зібрані дані мають бути якісними, точними, повними та релевантними для конкретної задачі. Якщо цього не буде, аналіз буде недостовірним.

Наступним кроком є підготовка даних. Цей етап вимагає видалення дублікатів, заповнення пропущених значень, корекції різноманітних невідповідностей та перетворення інформації у зручний формат для аналізу. Це одна з дуже важливих частин процесу, бо якість даних напряму впливає на точність результатів, і погано очищені дані можуть зіпсувати усі результати аналізу.

Далі йде дослідження даних. Дослідження включає в себе аналіз розподілу, статистичних характеристик і візуалізацію для виявлення закономірностей, аномалій чи трендів. Цей етап дозволяє отримати попереднє уявлення про структуру даних і їхні особливості.

Наступним кроком, на основі даних визначаються ключові змінні – предиктори, які мають найбільший вплив на результати. Це також називають відбором або інженерією ознак. На цьому етапі можуть додаватися нові ознаки, які краще відповідають задачі, або прибиратися зайві, що не дають цінної інформації.

Після підготовки даних і ознак обирається модель. Тип моделі залежить від задачі: для класифікації беруться, наприклад, рішення дерев або SVM, для регресії – лінійна регресія чи нейронні мережі, а для кластеризації – алгоритм k-means чи подібні. Вибір моделі визначається потребами в точності та загалом від конкретної задачі.

Окрім описаних нижче кластеризації, класифікації та асоціації, важливою задачею є регресія, зокрема завдання прогнозування, яку буде коротко описано тут. Встановлення залежності безперервних вихідних від вхідних змінних. У завданнях регресії та класифікації потрібно визначити

значення залежної змінної об'єкта на підставі значень інших змінних, що характеризують цей об'єкт.

Нехай дано кінцевий безліч об'єктів $I = \{i1,i2,\ldots,ij,\ldots in\}$. Кожен із об'єктів характеризується деяким ознаковим описом $(x1,x2,\ldots,xk,\ldots,xm,xm+1)$. Нехай значення ознак $(x1, x2, \ldots, xk, \ldots, xm)$ відомі. Тоді завдання полягає у визначенні невідомої ознаки $xm+1$.

Інші задачі та методи будуть описані у наступних пунктах роботи.

Далі йде ще один дуже важливий етап – навчання моделі. Це один із ключових етапів процесу, саме на якому алгоритм отримує можливість аналізувати патерни та закономірності у підготовлених даних. Для цього дані зазвичай розділяються на навчальний і тестовий набори. Навчальний набір використовується для тренування моделі, тобто для оптимізації її параметрів, тоді як тестовий набір потрібен для оцінки її продуктивності на даних, які модель раніше не використовувала. У деяких випадках використовують також валідаційний набір, щоб налаштувати гіперпараметри та зменшити ризик перенавчання.

Під час навчання алгоритм поступово налаштовує свої параметри, щоб мінімізувати похибку. Наприклад, у лінійній регресії налаштовуються коефіцієнти, які визначають вагу кожного предиктора, а у нейронних мережах – оптимізуються ваги та зсуви через алгоритм зворотного поширення помилки – backpropagation. Навчання може бути ітеративним – наприклад, у градієнтному спуску, коли модель поступово покращується за декілька епох, або відбуватися за один крок – наприклад, у деяких алгоритмах кластеризації.

Етап оцінки моделі спрямований на перевірку її здатності вирішувати поставлену задачу. Основні метрики оцінки залежать від типу проблеми: для класифікації це можуть бути точність (accuracy), повнота (recall), специфічність та F1-міра, а для регресії – середньоквадратична похибка (MSE), середня абсолютна похибка (MAE) або коефіцієнт

детермінації (R^2). У випадку кластеризації застосовуються такі метрики, як силуетний коефіцієнт чи індекс Дейвіса-Боулдіна.

При незадовільних результатах, можна переналаштувати гіперпараметри моделі, обрати інший алгоритм, якщо той виявився занадто неефективним, чи розширити дані чи покращити їх попередню обробку.

Впровадження моделі у середовище є наступним кроком після успішної оцінки. На цьому етапі модель інтегрується в бізнес-процеси або системи для використання у реальних, а не тестових, сценаріях. Це може включати створення API для обробки нових даних у реальному часі, інтеграцію в існуючі аналітичні платформи чи налаштування автоматичного прийняття рішень на основі прогнозів моделі. Наприклад, модель може автоматично прогнозувати ризики шахрайства в банківських транзакціях або рекомендувати товари у системах електронної комерції.

Моніторинг і підтримка моделі є останнім, але не менш важливим етапом. В умовах реального використання модель може поступово втрачати свою точність через зміну вхідних даних або бізнес-контексту – дрейф даних. Для запобігання такої проблеми, треба регулярно перевіряти продуктивність моделі на нових даних. Якщо виявляється, що модель більше не дає коректних прогнозів, проводиться її донавчання на більш актуальних даних або повна переробка.

Крім цього, важливо забезпечити автоматичне оновлення даних, звітність про роботу моделі та реакції на зміни в її середовищі. У складних системах може застосовуватись поєднання кількох моделей або резервна модель, щоб уникнути збоїв у роботі.

1.2 Кластеризація

Однією з актуальних та важливих задач аналізу даних є кластеризація. Кластеризація – це метод аналізу даних, що використовується для групування схожих об'єктів у кластери (групи) без заздалегідь визначених

категорій чи міток. Цей метод належить до некерованого навчання, оскільки він не потребує попередньо маркованих даних. Основна ідея кластеризації полягає в тому, щоб об'єкти в межах одного кластеру були максимально схожими між собою, а об'єкти з різних кластерів – максимально відрізнялися.

На нижче показаному рисунку 1.1, взятого з статті [1] на сайті serokell, можна побачити візуалізацію роботи алгоритму кластеризації, у даному випадку – алгоритму k-means.

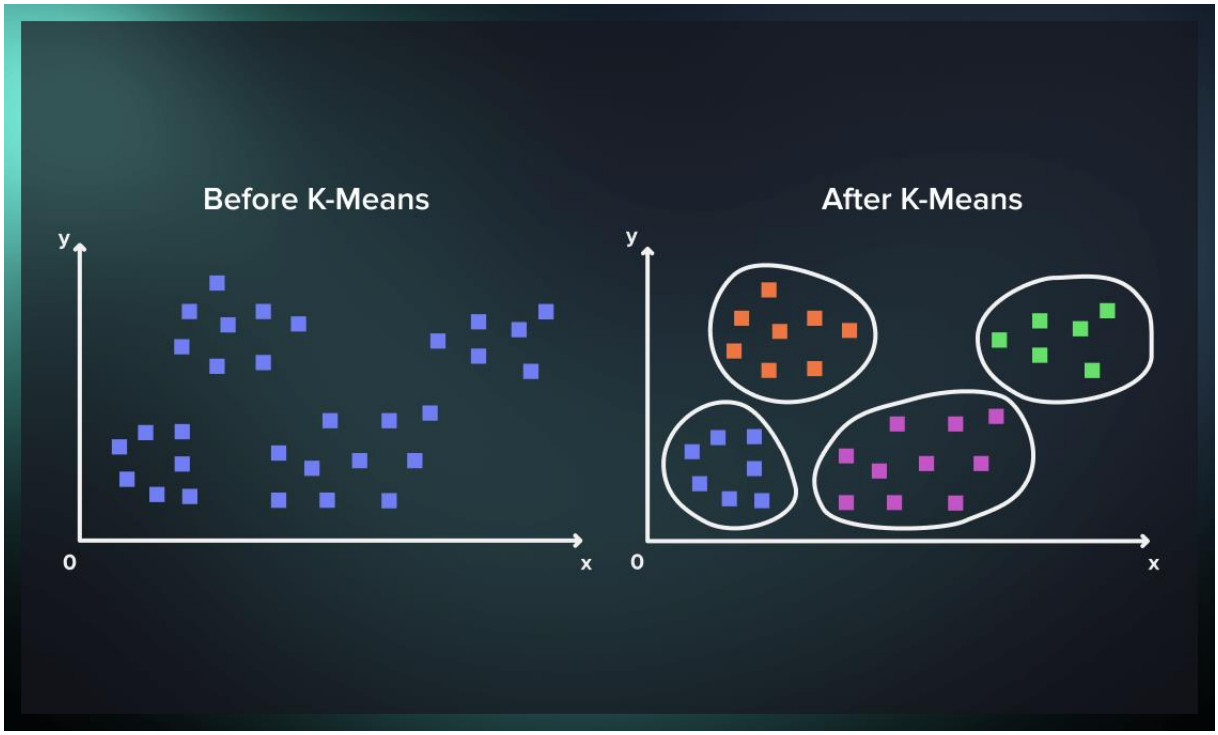

Рисунок 1.1 – Приклад роботи алгоритму кластеризації k-means

Процес кластеризації починається з вибору характеристик, які описують дані. Важливо, щоб ці ознаки точно відображали властивості об'єктів, адже їхня якість безпосередньо впливає на точність кластеризації.

Після визначення ознак вибирається алгоритм кластеризації. Найпоширенішим є алгоритм k-means, який працює шляхом поділу об’єктів на k кластерів, причому кожен об’єкт належить до кластеру з найближчим

центром (центроїдом). Робота алгоритму починається з вибору кількості кластерів k, що задається користувачем. Цей вибір є важливим і часто залежить від попереднього аналізу даних. Наприклад, кількість кластерів можна визначити за допомогою методу «лікоть», аналізуючи, як змінюється сума квадратів відстаней між об’єктами та їх центроїдами при збільшенні k.

Алгоритм ініціалізує k центрів кластерів, вибираючи їх випадково або за допомогою вдосконаленого підходу k-means++, який підбирає центри так, щоб вони були максимально віддаленими один від одного. Далі кожен об'єкт прив'язується до найближчого центру за метрикою, наприклад, евклідовою відстанню – це створює початкові кластери.

Після цього алгоритм обчислює нові центроїди для кожного кластера, використовуючи середнє значення координат об'єктів у цьому кластері. На основі оновлених центроїдів знову проводиться розподіл об'єктів між кластерами. Цей процес повторюється ітеративно до досягнення стабільності, а саме – коли розподіл об'єктів перестає змінюватися, або виконується завчасно задана кількість ітерацій.

Однією з переваг k-means є його простота та швидкість роботи, що робить його придатним для аналізу великих наборів даних. Однак алгоритм має і свої недоліки. Наприклад, він є чутливим до початкових умов, через що результати можуть змінюватися залежно від початкового розташування центроїдів. Для розв'язання цієї проблеми алгоритм часто запускають кілька разів з різними початковими точками, вибираючи розв'язок із найкращою сумарною відстанню.

Іншим обмеженням є те, що k-means погано працює із даними нерівномірної щільності або нестандартної форми кластерів. У таких випадках об’єкти можуть бути неправильно розподілені між групами.

Крім того, алгоритм потребує нормалізації даних, якщо ознаки мають різний масштаб, оскільки відстань між точками буде домінувати за характеристиками з більшими значеннями.

Загалом k-means широко використовується для аналізу даних завдяки своїй ефективності, особливо в задачах, де кластери мають сферичну форму та приблизно однакову щільність. Цей метод застосовують у маркетинговій сегментації клієнтів, кластеризації зображень, виявленні аномалій, аналізі текстів та інших випадках.

Іншим популярним підходом є алгоритм агломеративної кластеризації, який будує певну ієрархію кластерів. Спочатку кожен об'єкт розглядається як окремий кластер, і поступово об'єднує їх у більші групи, поки всі об'єкти не опиняться в одному загальному кластері або не буде досягнуто заданої кількості кластерів.

Процес кластеризації базується на визначенні відстані між кластерами, що може розраховуватися за різними методами. Найбільш поширені способи включають мінімальну відстань між точками двох кластерів (single linkage), максимальну відстань між точками (complete linkage) і середню відстань між усіма парами точок (average linkage). Інший популярний підхід – використання центроїдів, де відстань між кластерами розраховується як відстань між їхніми центроїдами. Вибір методу вимірювання впливає на форму та структуру отриманих кластерів.

Алгоритм працює ітеративно. На кожному етапі два найближчі кластери об'єднуються в один, і це повторюється, поки не буде досягнуто бажаного стану. Результати зазвичай представляються у вигляді дендрограми – графічного дерева, яке відображає порядок ієрархічного об'єднання кластерів. Дендрограма дозволяє візуально визначити, на якому етапі об'єднання слід зупинитися, щоб отримати оптимальну кількість кластерів.

Агломеративна кластеризація добре підходить для даних, де не відома наперед кількість кластерів або де потрібно дослідити багаторівневу структуру груп. Вона є особливо корисною для аналізу невеликих або середніх за розміром наборів даних, оскільки обчислювальна складність може швидко зростати зі збільшенням кількості об'єктів.

Однак цей метод має певні обмеження. Він чутливий до шуму та викидів, оскільки процес об'єднання незворотний – якщо два кластери були об'єднані, їх неможливо розділити в подальших кроках. Крім того, агломеративна кластеризація передбачає обчислення відстаней між усіма парами кластерів, що, як уже було сказано, робить її менш ефективною для дуже великих наборів даних.

Агломеративна кластеризація знаходить застосування в багатьох галузях. У біології її використовують для аналізу генетичних даних і класифікації видів. У соціальних науках вона допомагає вивчати схожість між групами людей або культур. У бізнесі цей метод може застосовуватися для сегментації клієнтів чи аналізу ринків. Завдяки своїй здатності виявляти багаторівневу структуру, агломеративна кластеризація є цінним інструментом для дослідження складних даних.

Також широко використовується алгоритм DBSCAN [2], який групує об'єкти на основі щільності. На відміну від k-means, цей метод не потребує попереднього визначення кількості кластерів і добре справляється з даними нерівномірної щільності. DBSCAN визначає «ядрові точки», які мають достатню кількість сусідів, і об'єднує їх у кластери, ігноруючи «шум» – об'єкти, що не належать до жодного кластеру.

Процес кластеризації починається з вибору будь-якої точки. Якщо вона є ядровою, утворюється новий кластер, і всі її сусіди додаються до цього кластеру. Потім алгоритм рекурсивно перевіряє сусідів і розширює кластер до тих пір, доки не будуть охоплені всі точки, які відповідають критеріям щільності. Якщо точка не є ядровою і не належить до жодного існуючого кластеру, вона позначається як шумова.

Однією з головних переваг DBSCAN є його здатність виявляти кластери довільної форми, включаючи конічні, лінійні або кільцеподібні структури, які важко розпізнати методами на зразок k-means. Алгоритм також стійкий до шуму, що дозволяє йому ефективно працювати з

реальними даними, які часто містять аномалії або точки, що не вписуються в загальну картину.

Не зважаючи на це, DBSCAN все одно має свої обмеження. Він дуже чутливий до вибору параметрів – неправильно підібрані значення можуть призвести до утворення занадто великих або занадто малих кластерів, або до неправильного визначення шумових точок. Для великих багатовимірних даних обчислювальна складність алгоритму також може стати проблемою, оскільки він вимагає розрахунку відстаней між всіма парами точок, що може бути повільним.

Для будь якого методу, оцінка якості кластеризації є важливим етапом процесу. Зазвичай використовується така метрика, як силуетний коефіцієнт, яка оцінює, наскільки добре кожен об'єкт узгоджується зі своїм кластером порівняно з іншими. Інші підходи включають візуалізацію, наприклад, побудову графіків за допомогою методів зниження розмірності, таких як PCA або t-SNE, щоб перевірити якість групування.

Загалом, кластеризація широко застосовується у багатьох галузях. У маркетингу вона використовується для сегментації клієнтів, у біології – для класифікації генів або білків, у фінансах – для виявлення груп активів з подібною динамікою. Її також застосовують для аналізу соціальних мереж, визначення регіонів із різною інтенсивністю злочинності, кластеризації зображень та багатьох інших задач.

Та, таким чином, кластеризація є потужним і універсальним інструментом аналізу даних, який дозволяє отримувати цінні результати без потреби в попередньо визначених категоріях. Вона допомагає виявляти структуру даних, що є важливою основою для прийняття обґрунтованих рішень.

1.3 Класифікація

Наступною широко використовуваною задачею є класифікація. Класифікація – це один із основних типів задач у машинному навчанні, яка має за мету розподіл об’єктів або даних у заздалегідь визначені категорії чи класи. Вона базується на створенні моделі, здатної прогнозувати клас нових об’єктів на основі наявних даних, які використовуються для навчання.

Цей метод є широко застосовуваним у різних сферах, таких як медицина, маркетинг, фінанси, розпізнавання образів та обробка текстів, як і інші методи інтелектуального аналізу даних.

На нижче показаному рисунку 1.2, взятого з статті [3] на сайті botpenguin, можна побачити візуалізацію роботи алгоритму класифікації – показано розділення даних на 2 різних класи – A та B.

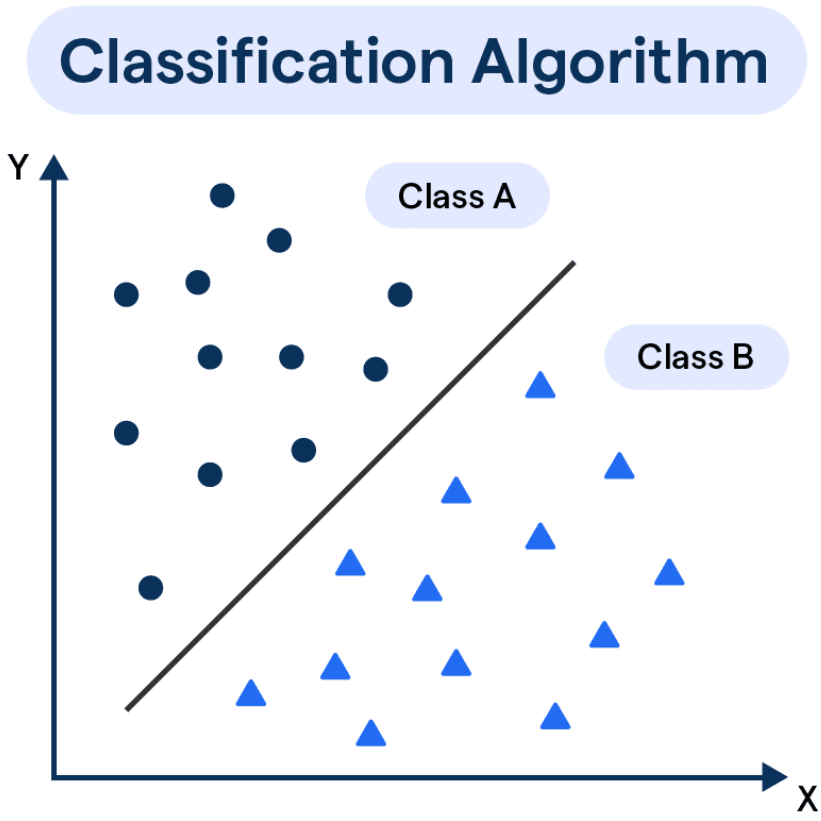

Рисунок 1.2 – Приклад роботи алгоритму класифікації

Процес класифікації починається зі збору та підготовки даних. Мета полягає в отриманні якісного набору даних, який складається з прикладів із позначеними класами. Кожен приклад містить набір характеристик і відповідний клас або мітку. Цей набір використовується для навчання моделі, яка намагається знайти закономірності між характеристиками й класами.

Модель будується шляхом вибору алгоритму класифікації. Існує багато підходів, таких як логістична регресія, дерева рішень, метод опорних векторів (SVM), наївний байєсівський класифікатор, нейронні мережі та ансамблеві методи (наприклад, Random Forest або Gradient Boosting). Кожен із цих методів має свої переваги залежно від задачі, структури даних і вимог до інтерпретованості або точності.

Логістична регресія є одним із найпростіших і найпоширеніших алгоритмів класифікації. Вона використовується для задач, де потрібно передбачити ймовірність належності до певного класу. Алгоритм обчислює ймовірність приналежності об'єкта до класу за допомогою логістичної функції (сигмоїди), яка переводить значення з лінійної моделі у діапазон від 0 до 1.

Перевагою логістичної регресії є її інтерпретованість – вагу кожної характеристики можна аналізувати, щоб зрозуміти її вплив на результати. Цей метод добре працює для задач із лінійними розділеннями класів, але може виявитися недостатньо гнучким для складніших випадків.

Дерева рішень є інтуїтивно зрозумілими моделями, які використовують ієрархічний підхід до класифікації. У їхній основі лежить розподіл даних на підгрупи за допомогою правил, які базуються на значеннях характеристик. Кожне розгалуження дерева відповідає певній умові, а листя дерева представляють кінцеві класи.

Дерева рішень легко візуалізувати, що робить їх корисними для інтерпретованих моделей. Однак, з недоліків, вони мають тенденцію до перенавчання, особливо якщо дерево занадто глибоке. Для покращення

результатів часто використовуються методи обрізання (pruning) та інші регуляризації.

Метод опорних векторів (SVM) шукає гіперплощину, яка максимально розділяє класи в багатовимірному просторі. Цей метод ідеально підходить для задач із чітким розмежуванням класів. У разі, якщо класи не є лінійно роздільними, SVM використовує ядрові функції (kernel trick), які проектують дані у вищий вимір, де розділення стає можливим. SVM є потужним інструментом, особливо для задач із малими наборами даних і високою розмірністю.

Однак, він може бути обчислювально дорогим для великих наборів даних, а також складно налаштовується через вибір параметрів, таких як тип ядра та параметр регуляризації.

Наївний байєсівський класифікатор базується на теоремі Байєса і припущенні про незалежність характеристик. Він обчислює ймовірність приналежності до кожного класу на основі значень характеристик і вибирає клас із найвищою ймовірністю.

Цей метод є надзвичайно швидким і ефективним для великих наборів даних, а також добре працює з текстовими даними, як у випадку класифікації електронної пошти чи аналізу тональності тексту. Головним недоліком є його обмеження, пов’язані з припущенням про незалежність характеристик, яке не завжди відповідає реальності.

Нейронні мережі є складними нелінійними моделями. Вони складаються з шарів нейронів, де кожен нейрон отримує вхідні значення, обробляє їх за допомогою ваги і функцій активації та передає результат на наступний шар. Нейронні мережі надзвичайно гнучкі та можуть моделювати складні залежності в даних.

Вони особливо корисні для задач із великою кількістю даних, таких як розпізнавання образів, обробка природної мови та прогнозування. Однак, їхнє навчання вимагає значних обчислювальних ресурсів і може бути складним через налаштування великої кількості параметрів.

Ансамблеві методи, такі як Random Forest і Gradient Boosting, поєднують результати кількох моделей для підвищення точності та стійкості класифікації.

Random Forest використовує велику кількість дерев рішень, побудованих на випадкових підвибірках даних і характеристик, а потім об'єднує їх прогнози шляхом голосування або середнього значення. Це робить його стійким до перенавчання та зменшує чутливість до шуму.

Gradient Boosting, на відміну від Random Forest, створює дерева послідовно, де кожне наступне дерево фокусується на корекції помилок попереднього. Gradient Boosting забезпечує високу точність, але є більш чутливим до перенавчання, тому потребує ретельної налаштування гіперпараметрів.

У навчанні моделі використовуються мітки класів і відповідні їм характеристики. Модель оптимізує свої параметри для зменшення помилок класифікації на навчальних даних. Для перевірки якості моделі зазвичай використовуються тестові дані або метод крос-валідації. Це допомагає оцінити її здатність до узагальнення, тобто правильно класифікувати нові, раніше невідомі приклади.

Після навчання модель використовують для прогнозування класу нових даних. Наприклад, у задачі класифікації електронних листів модель може визначати, чи є новий лист спамом, чи ні. У випадку медичних задач модель може класифікувати пацієнтів на основі їхніх симптомів, визначаючи, чи є у них певне захворювання чи травма.

Для оцінки ефективності моделі використовують такі метрики, як точність, повнота, F1-міра, ROC-крива та площа під нею (AUC). Вибір метрики залежить від задачі. Наприклад, у задачах, де важливо уникнути помилок в одному класі, більший акцент роблять на метриках, що враховують баланс між точністю і повнотою.

Класифікація може бути бінарною, коли є лише два класи, або багатокласовою, коли об’єкти належать до одного з багатьох класів. У

складніших випадках застосовують багатоміткову класифікацію, коли один об'єкт може належати до кількох класів одночасно.

Сучасні класифікаційні моделі мають широкий спектр застосувань. Наприклад, у фінансах їх використовують для прогнозування кредитоспроможності клієнтів, виявлення шахрайських операцій та аналізу ризиків. У медицині класифікаційні моделі допомагають у діагностиці захворювань, аналізі зображень (наприклад, для виявлення пухлин) і персоналізованій медицині. У маркетингу моделі класифікації використовують для сегментації клієнтів, аналізу поведінки користувачів і прогнозування їхньої реакції на рекламні кампанії.

Таким чином, класифікація є універсальним і потужним інструментом для вирішення задач, пов’язаних із прогнозуванням, розпізнаванням і прийняттям рішень. Завдяки розвитку методів машинного навчання та доступності великих обсягів даних, її значення та ефективність продовжують зростати.

1.4 Асоціація

Задачею, на яку буде поставлений акцент в цій роботі є асоціація. Асоціація – це метод аналізу даних, який виявляє приховані зв’язки або закономірності між елементами в наборах даних. Цей підхід широко використовується для дослідження транзакцій або взаємозв'язків між елементами, що часто з'являються разом. Найпоширенішим прикладом є аналіз кошика покупок, де мета полягає у виявленні того, які товари купуються разом. У прикладі цієї роботи це буде показано на основі покупки лік у аптеці.

На нижче показаному рисунку 1.3, взятого з статті [4] на сайті kdnuggets, можна побачити візуалізацію роботи алгоритму асоціації, а саме алгоритму Apriori, який буде описаний нижче у цьому підрозділі.

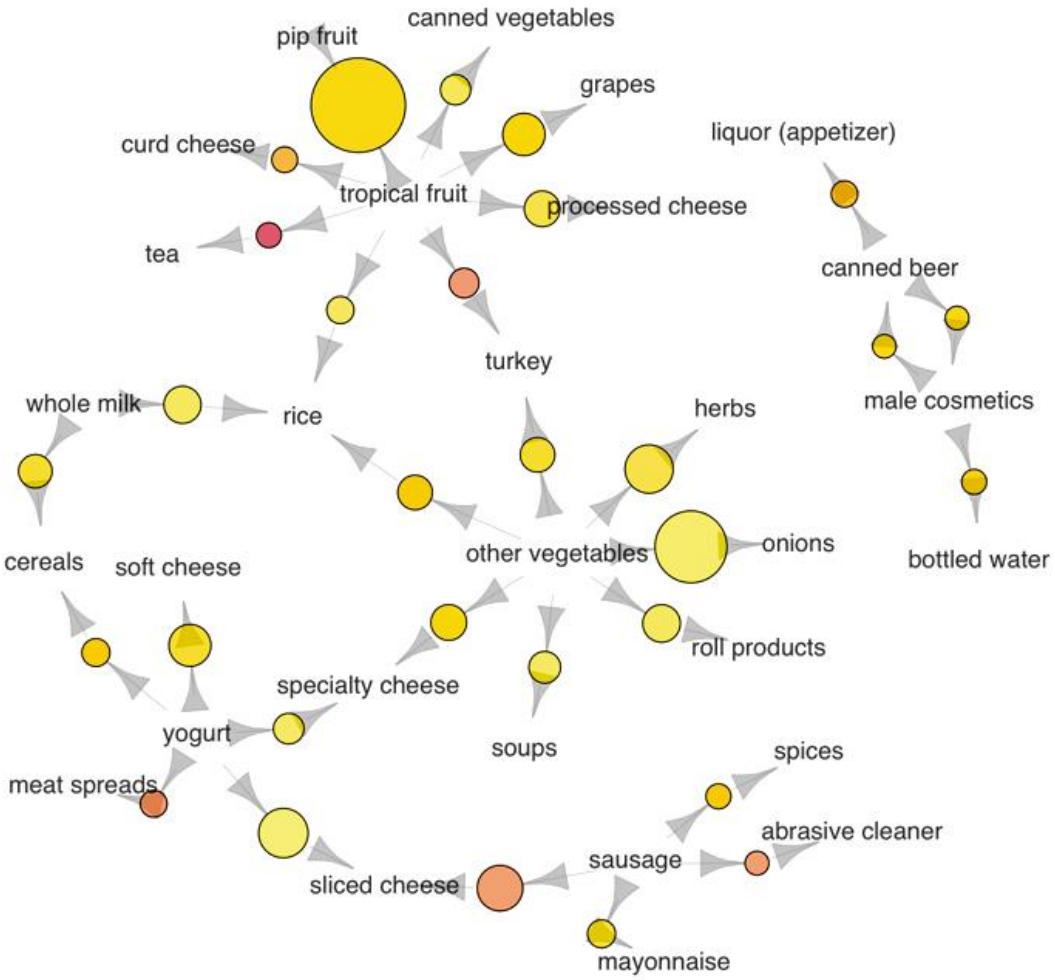

Рисунок 1.3 – Приклад роботи алгоритму асоціації Apriori

Процес аналізу асоціації починається зі створення транзакційного набору даних, де кожен запис представляє унікальну транзакцію, що містить перелік елементів. На цьому етапі особливу увагу приділяють якості даних, адже будь-які пропущені або недостовірні дані можуть вплинути на результат.

Головною метою є пошук асоціативних правил, які зазвичай мають вигляд «якщо A, то B». Кожне правило оцінюється за допомогою трьох основних метрик: підтримка, довіра та ліфт.

Підтримка (Support) визначає, наскільки часто конкретна група елементів зустрічається в наборах транзакцій. Вона обчислюється як

відношення кількості транзакцій, що містять ці елементи, до загальної кількості транзакцій. Це допомагає визначити популярність правила.

Якщо визначати це як формулу, то

$$Support\ (A \rightarrow B) = \frac{\text{Кількість транзакцій, що містять } A \cup B}{\text{Загальна кількість транзакцій}}. \quad (1.1)$$

Довіра (Confidence) вимірює ймовірність того, що елемент B буде придбаний у тих транзакціях, де вже є елемент A. Висока довіра свідчить про сильний зв'язок між елементами.

Якщо визначати це як формулу, то

$$Confidence\ (A \rightarrow B) = \frac{Support(A \cup B)}{Support(A)}. \quad (1.2)$$

Ліфт (Lift) порівнює фактичну частоту спільної появи елементів із тією, яку можна було б очікувати за випадкових умов. Значення ліфта більше за одиницю свідчить про позитивну залежність між елементами, тобто вони частіше з'являються разом, ніж окремо.

Якщо визначати це як формулу, то

$$Lift\ (A \rightarrow B) = \frac{Confidence\ (A \rightarrow B)}{Support(B)} \quad (1.3)$$

або

$$Lift\ (A \rightarrow B) = \frac{Support(A \cup B)}{Support(A) \times Support(B)}. \quad (1.4)$$

Серед алгоритмів, що використовуються для аналізу асоціацій, найбільш популярним є алгоритм Apriori, приклад роботи якого було показано на рисунку 1.3 вище. Його основна ідея базується на так званій

властивості анти-мономності частих наборів: якщо набір елементів є частим, тоді всі його підмножини також будуть частими. Це означає, що для відсіювання рідкісних комбінацій можна аналізувати лише ті набори, які мають часті підмножини.

Процес роботи алгоритму починається з пошуку частих одноелементних наборів. Наприклад, якщо у транзакціях часто зустрічається якийсь елемент, то цей елемент буде включено до подальшого аналізу. Далі ці набори розширюються шляхом додавання інших елементів, які також часто зустрічаються разом із початковими. Цей процес повторюється, доки не буде знайдено всі часті набори, що відповідають встановленому порогу підтримки.

Основним кроком алгоритму є побудова кандидатних наборів (тобто потенційно частих наборів), які генеруються шляхом комбінування вже знайдених частих наборів. Для кожного з цих кандидатів обчислюється підтримка, і якщо вона перевищує заданий поріг, набір вважається частим. Завдяки цьому механізму Apriori значно скорочує кількість обчислень, оскільки уникає аналізу рідкісних комбінацій.

Наступним етапом є побудова асоціативних правил із частих наборів. Для кожного частого набору виділяються всі можливі розділення на «ліву» і «праву» частини правила. Для набору {один, два, три} можуть бути згенеровані такі правила, як «якщо один і три, то два». Для кожного такого правила обчислюються метрики довіри та ліфта, щоб оцінити силу й корисність знайдених закономірностей.

Apriori особливо добре працює для невеликих і середніх за розміром наборів даних. Проте для великих транзакційних баз його ефективність може знижуватися через велику кількість ітерацій і високі обчислювальні витрати, особливо якщо поріг підтримки встановлено дуже низьким. У таких випадках використання алгоритму FP-Growth, який буде описаний нижче, або інших оптимізованих підходів може бути більш доцільним.

У реальних застосуваннях Apriori часто використовується для аналізу купівельних звичок клієнтів, створення рекомендаційних систем, оптимізації бізнес-процесів, виявлення шахрайства та інших задач. Його простота й інтуїтивна зрозумілість роблять його популярним інструментом для вивчення закономірностей у даних.

Іншим популярним, та вже згаданим вище алгоритмом є FP-Growth [5], який створює структуру даних у вигляді дерева частих елементів (FP-tree). Цей метод дозволяє працювати ефективніше, ніж Apriori, особливо для великих наборів даних, адже не потребує багаторазового проходу через весь транзакційний набір.

FP-Growth використовує інший підхід до зберігання та обробки даних, що дозволяє зменшити кількість необхідних проходів через транзакційний набір і виключити потребу у створенні великої кількості кандидатних наборів.

Основною особливістю FP-Growth є використання дерева частих патернів (FP-дерева) як структури даних. FP-дерево компактно представляє транзакційний набір, об'єднуючи елементи, що часто зустрічаються разом, у вигляді вузлів, пов'язаних між собою. Для побудови дерева алгоритм спочатку проходить по даних і визначає частоту кожного елемента. Потім відсіюються ті елементи, частота яких менша за заданий поріг підтримки. Залишкові елементи сортуються за спаданням частоти, і на основі цього створюється FP-дерево.

У FP-дереві кожна транзакція додається як шлях, а загальні елементи об’єднуються в загальні вузли. Це значно скорочує обсяг пам’яті, необхідний для зберігання даних, особливо якщо транзакції містять багато спільних елементів. У результаті, навіть великі набори даних можуть бути представлені в компактному вигляді.

Після побудови дерева алгоритм рекурсивно аналізує його, щоб виділити часті патерни. Для цього використовується метод розділення дерева на умовні піддерева для кожного елемента. Наприклад, якщо елемент

«один» є частим, аналізуються всі транзакції, які його містять, для виявлення додаткових частих патернів, що з ним асоціюються, таких як «один та два» або «один та три».

FP-Growth особливо ефективний для великих і щільних транзакційних баз, де Apriori може виявитися надто повільним через потребу в багаторазовому генеруванні й перевірці кандидатних наборів. FP-Growth працює швидше, оскільки використовує лише два проходи через дані: один для обчислення частоти елементів і створення дерева, а другий для аналізу дерева.

Алгоритм широко застосовується в задачах рекомендацій, аналізу поведінки користувачів, маркетингу та бізнес-аналітики. Наприклад, у роздрібній торгівлі FP-Growth може допомогти виявити, які товари найчастіше купують разом, щоб оптимізувати їхнє розташування на полицях чи створити акційні набори. Також він корисний у телекомунікаціях для аналізу послуг, які часто замовляються разом.

Хоча FP-Growth є швидким і пам'яттєво ефективним, його реалізація може бути складнішою, ніж у Apriori. Однак переваги у швидкості й продуктивності роблять його частим вибором для аналізу великих обсягів транзакційних даних.

Загалом, асоціаційний аналіз знаходить застосування в багатьох галузях. У роздрібній торгівлі він допомагає оптимізувати розташування товарів на полицях, створювати акційні набори та прогнозувати поведінку клієнтів. У медицині цей метод використовується для виявлення комбінацій симптомів або побічних ефектів лікарських препаратів. У сфері маркетингу він дозволяє сегментувати аудиторію та створювати персоналізовані рекомендації.

Однак, незважаючи на широке застосування, аналіз асоціацій має свої обмеження. Наприклад, великий набір даних із багатьма елементами може призводити до генерування величезної кількості правил, більшість із яких будуть нерелевантними. Тому важливо правильно налаштувати пороги

підтримки й довіри та фільтрувати результати, щоб отримати корисну інформацію.

Окрім зазначених обмежень, варто зазначити, що аналіз асоціацій може стикатися з проблемою рідкісних елементів у наборі даних, які мають низьку частотність і тому часто ігноруються через високі пороги підтримки. Це може призводити до втрати потенційно цікавих закономірностей. Також значною перешкодою є обчислювальна складність, особливо для великих наборів даних, оскільки пошук асоціативних правил вимагає перевірки великої кількості можливих комбінацій елементів, що може бути дуже ресурсозатратним.

Таким чином, асоціація є потужним інструментом для виявлення зв’язків у даних, дозволяючи бізнесу та науковцям приймати обґрунтовані рішення на основі знайдених закономірностей.

1.5 OLAP

Важливою технологією є OLAP – OnLine Analytical Processing. OLAP – це технологія для інтерактивного аналізу великих обсягів багатовимірних даних, яка широко використовується для бізнес-аналітики, планування, прийняття рішень і дослідження тенденцій. Вона забезпечує швидкий доступ до структурованих даних, дозволяючи користувачам виконувати складні аналітичні запити в реальному часі.

Основна концепція OLAP полягає в роботі з багатовимірними структурами даних, які називаються кубами. Куби OLAP організовують дані за кількома вимірами, такими як час, продукти, регіони чи клієнти. Наприклад, продажі можуть бути представлені у вигляді куба, де однією вимірністю є регіон, іншою – час, а третьою – категорії продуктів. Така структура дозволяє швидко й легко виконувати аналітичні операції, такого типу як порівняння продажів між регіонами чи аналіз змін у часі.

На нижче показаному рисунку 1.4, взятого з статті [6] на сайті techtarget, можна побачити візуалізацію роботи OLAP.

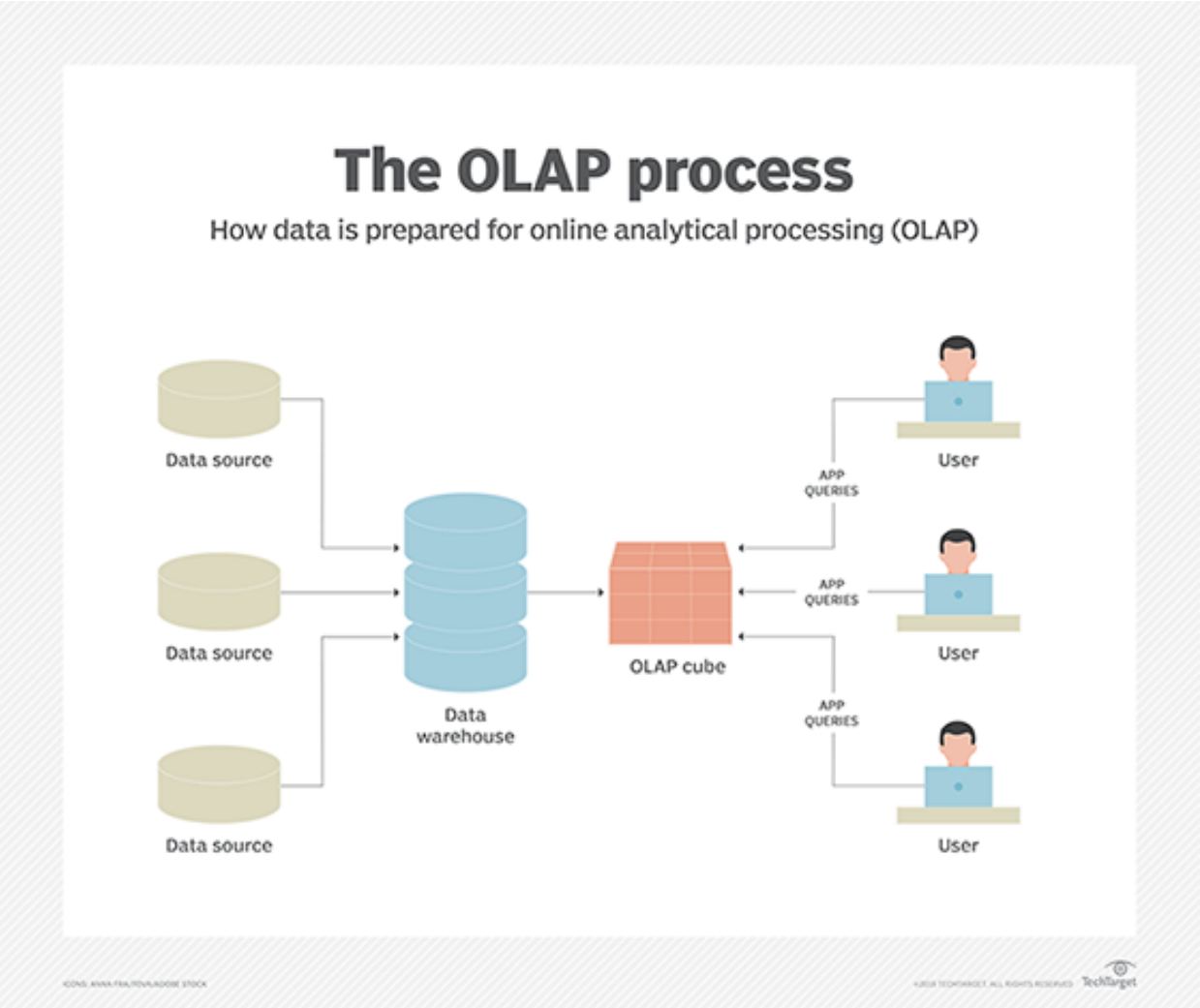

Рисунок 1.4 – Приклад роботи OLAP

OLAP підтримує декілька основних операцій для роботи з даними. Вони забезпечують динамічний підхід до дослідження даних, дозволяючи змінювати рівень деталізації, вибирати конкретні сегменти або змінювати ракурс аналізу.

Серед операцій можна визначити такі:

- roll-up;
- drill-down;
- slice;

- dice;
- pivot.

Операція roll-up агрегує дані на більш високому рівні абстракції, зменшуючи деталізацію та узагальнюючи інформацію. Ця операція дозволяє отримувати загальну картину, представляючи підсумкові дані.

Операція drill-down, навпаки, деталізує інформацію, переходячи до більш низького рівня, що дає змогу аналізувати дані в контексті окремих частин.

Операція slice фокусує увагу на одному шарі багатовимірних даних, обираючи певне значення для одного з вимірів. Це дозволяє аналізувати дані у вибраній площині.

Операція dice працює з підмножинами даних, застосовуючи кілька критеріїв одночасно. Це дозволяє звузити аналіз, досліджуючи конкретний набір умов.

Операція pivot змінює представлення даних, переставляючи місцями виміри для зміни перспективи. Це дає можливість переглядати інформацію під іншим кутом, що може сприяти глибшому розумінню взаємозв’язків у даних.

OLAP-архітектури поділяються на три основні типи залежно від того, як зберігаються дані. Основними типами таких архітектур є MOLAP – Multidimensional OLAP, ROLAP – Relational OLAP і HOLAP – Hybrid OLAP, кожен із яких має свої особливості.

На нижче показаному рисунку 1.5, взятого з статті [7] на сайті techtarget, можна побачити розділення типів OLAP, з MOLAP, ROLAP, HOLAP, що будуть описані детальніше у цьому підрозділі, і іншими менш використованими типами як WOLAP – Web-Enabled OLAP, DOLAP – Desktop OLAP, SOLAP – Spatial OLAP, та іншим типом, що теж має абревіацію MOLAP, але означає не Multidimensional, а Mobile OLAP, про які буде розказано більш кратко.

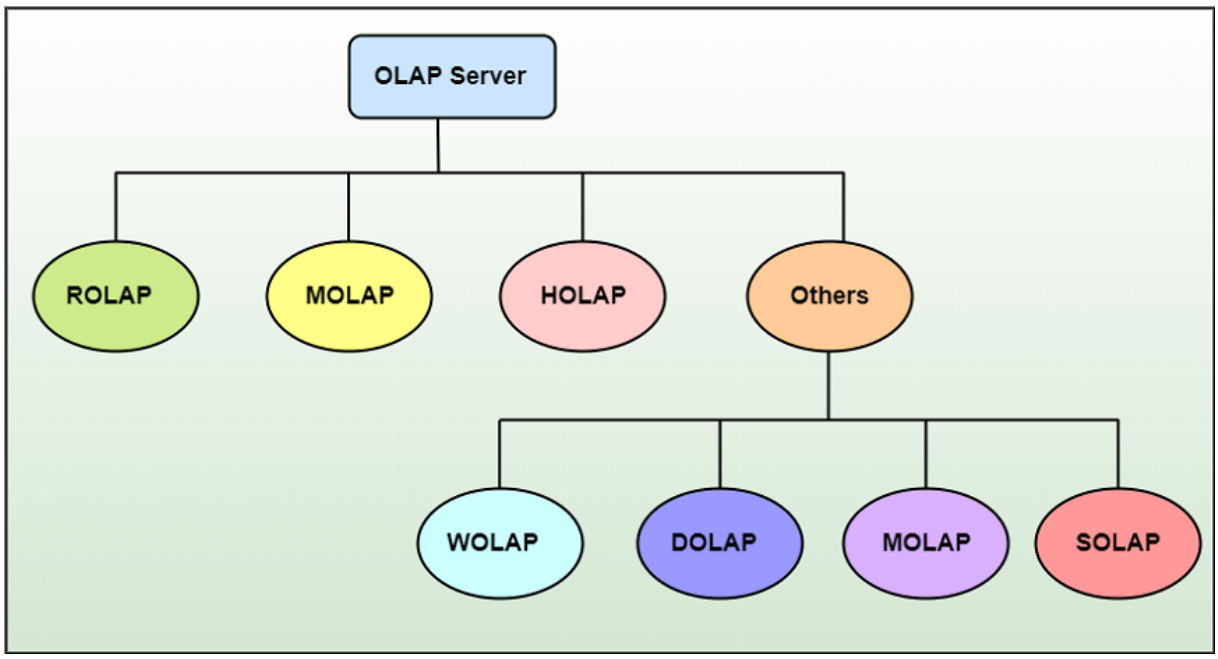

Рисунок 1.5 – Типи OLAP

MOLAP, або багатовимірна OLAP, використовує спеціалізовані багатовимірні масиви для зберігання даних. Цей підхід передбачає попереднє обчислення агрегованих значень і збереження їх у кубах даних, що забезпечує високу швидкість доступу до результатів. Завдяки цьому MOLAP добре підходить для швидкого виконання складних запитів, оскільки всі необхідні обчислення виконуються заздалегідь. Проте цей підхід має певні обмеження, зокрема потребу у значному обсязі пам’яті для зберігання великих багатовимірних масивів, а також складність у масштабуванні для дуже великих наборів даних.

ROLAP, або реляційна OLAP, ґрунтується на використанні реляційних баз даних для зберігання інформації. У цьому підході агреговані дані не зберігаються попередньо, а обчислюються в реальному часі під час виконання запитів. Це дозволяє ефективно працювати з великими обсягами даних, не обмежуючи їх розмір заздалегідь створеними багатовимірними структурами. ROLAP є більш масштабованим, ніж MOLAP, оскільки використовує можливості реляційних баз даних, проте швидкість доступу

до даних може бути нижчою через необхідність обчислювати агрегати в момент запиту.

HOLAP, або гібридна OLAP, поєднує найкращі риси MOLAP і ROLAP, використовуючи комбінований підхід до зберігання даних. У HOLAP частина даних, зазвичай найбільш часто використовувані або агреговані, зберігається у багатовимірних кубах, як у MOLAP, що забезпечує високу швидкість доступу до них. Решта даних залишається в реляційних базах, як у ROLAP, що дозволяє ефективно обробляти великі обсяги інформації, зберігаючи гнучкість. Це робить HOLAP збалансованим рішенням для організацій, яким потрібно поєднувати швидкість обробки і здатність працювати з великими даними.

Ці три типи архітектури забезпечують різні рівні продуктивності, гнучкості та масштабованості, що дозволяє організаціям обирати підхід залежно від їхніх аналітичних потреб і доступних ресурсів.

Існують також менш популярні типи стилів OLAP, на які можна натрапити час від часу.

WOLAP, або веб-орієнтований OLAP, відноситься до OLAP-додатків, які доступні через веб-браузер. На відміну від традиційних OLAP-додатків у форматі клієнт/сервер, WOLAP має трирівневу архітектуру, що складається з трьох компонентів: клієнтської частини, проміжного програмного забезпечення та сервера бази даних.

DOLAP, або десктопний OLAP, дозволяє користувачам завантажувати частину даних із бази даних чи джерела та працювати з цим набором даних локально, на своєму комп’ютері.

Інший MOLAP, а даному випадку – мобільний OLAP, забезпечує доступ до OLAP-даних та додатків віддалено за допомогою мобільних пристроїв.

SOLAP, або просторовий OLAP, поєднує можливості геоінформаційних систем (GIS) та OLAP в одному інтерфейсі користувача. Він дозволяє управляти як просторовими, так і непросторовими даними.

Однією з головних переваг OLAP є можливість швидко відповідати на складні запити без потреби в глибоких знаннях баз даних. Завдяки цьому бізнес-користувачі можуть аналізувати дані безпосередньо, не залучаючи технічних фахівців. Інша перевага – здатність працювати з агрегованими даними, що прискорює аналіз і зменшує обсяг оброблюваних даних.

OLAP широко застосовується в бізнес-аналітиці для дослідження фінансових показників, аналізу продажів, прогнозування попиту, моніторингу продуктивності. У фінансовій сфері OLAP допомагає аналізувати витрати та доходи, виявляти тенденції й аномалії. У маркетингу використовується для оцінки ефективності кампаній і сегментації клієнтів. У логістиці OLAP дозволяє відстежувати операційну ефективність і оптимізувати ланцюги постачання.

Проте OLAP має і певні обмеження. Складність у проектуванні кубів може призвести до труднощів у налаштуванні, особливо якщо дані змінюються. Крім того, для роботи з великими обсягами даних потрібне попереднє агрегування, що може впливати на гнучкість аналізу. Незважаючи на це, OLAP залишається основним інструментом для швидкого й інтуїтивного аналізу багатовимірних даних.

OLAP-системи реалізовані в різноманітних програмних рішеннях, які надають можливості для багатовимірного аналізу даних. Вони використовуються для бізнес-аналітики, фінансового планування, прогнозування, оптимізації ресурсів і багатьох інших завдань.

Серед найпопулярніших програмних рішень, які працюють з OLAP, можна виділити наступні:

- Microsoft SQL Server Analysis Services (SSAS);
- Oracle Essbase;
- IBM Cognos Analytics;
- SAP BusinessObjects Analysis;
- Tableau;
- Qlik Sense;

- QlikView;
- Pentaho BI;
- Apache Kylin;
- Mondrian.

Microsoft SQL Server Analysis Services (SSAS) є частиною Microsoft SQL Server і забезпечує розробку OLAP-кубів, а також реалізацію багатовимірного аналізу. SSAS підтримує MOLAP, ROLAP і HOLAP-архітектури, що дозволяє обирати оптимальний підхід для конкретного сценарію. Він інтегрується з іншими продуктами Microsoft, такими як Excel і Power BI, що робить його зручним для корпоративного використання.

Oracle Essbase – потужна OLAP-платформа від Oracle, яка забезпечує гнучкість і масштабованість для аналізу великих наборів даних. Essbase часто використовується у фінансовому плануванні, бюджетуванні та управлінні ресурсами. Вона підтримує MOLAP і дозволяє легко інтегруватися з іншими продуктами Oracle.

IBM Cognos Analytics включає OLAP-компоненти для аналізу даних та створення інтерактивних звітів. Cognos дозволяє створювати OLAP-куби, налаштовувати запити та візуалізувати багатовимірні дані. Ця система використовується в різних галузях для прийняття стратегічних рішень.

SAP BusinessObjects Analysis є частиною екосистеми SAP, що надає інструменти для багатовимірного аналізу та інтеграції з SAP HANA. Це рішення часто використовується у великих корпораціях для управління даними, фінансової звітності та прогнозування.

Tableau є популярною платформою для візуалізації та аналізу даних, яка також підтримує OLAP-запити. Tableau інтегрується з різними джерелами даних, включаючи OLAP-куби, і забезпечує зручний інтерфейс для створення інтерактивних дашбордів.

Qlik Sense і QlikView обидва є інструментами бізнес-аналітики, які підтримують OLAP-функціональність. Вони забезпечують інтерактивний

аналіз даних, що зберігаються в багатовимірних структурах, і пропонують ефективні засоби візуалізації.

Pentaho BI – це рішення з відкритим кодом, яке включає OLAP-інструменти для аналізу даних. Pentaho дозволяє створювати OLAP-куби та проводити багатовимірний аналіз із застосуванням різних архітектур.

Крім того, існує багато інших платформ і бібліотек, які підтримують OLAP-операції, зокрема open-source рішення, такі як Apache Kylin і Mondrian, які дозволяють реалізовувати OLAP-функціональність для великих обсягів даних. Користувачі обирають відповідне програмне забезпечення залежно від своїх потреб, бюджету та інтеграційних вимог.

2 ОГЛЯД СТРУКТУРОВАНИХ МОДЕЛЕЙ ДАНИХ

2.1 Загальні відомості

Цілком кажучи, структуровані моделі даних – це систематизовані методи представлення й організації інформації для зберігання, обробки та аналізу даних. Вони використовуються для роботи з даними, які мають чітко визначену структуру та формат, що дозволяє легко зберігати їх у вигляді таблиць або інших впорядкованих форматів. Основна мета структурованих моделей – забезпечити ефективне управління та доступ до даних, зберігаючи цілісність і взаємозв'язки між ними. Вони широко застосовуються в бізнесі, фінансовій сфері, наукових дослідженнях та урядових структурах для зберігання інформації про клієнтів, фінансові операції, медичні дані та багатьох інших напрямках.

Серед основних принципів структурованих моделей можна визначити:

- використання стовпців та рядків для організації;
- фіксована схема;
- підтримка запитів;
- відповідність реальності.

Використання стовпців та рядків для організації полегшує зберігання, сортування, пошук і фільтрацію даних. Зазвичай це структура таблиць з заздалегідь визначеними полями, де стовпець визначає якийсь атрибут, а рядок – запис стосовно цього атрибуту, чи об’єкт.

Фіксована схема у структурованих моделях визначає, які типи даних зберігаються чи які можна зберігати, зв’язки між об’єктами, обмеження щодо даних. За допомогою такої схеми зручніше та стабільніше працювати із даними, але також для деяких проектів чи ситуацій є недоліком. Така схема не дає зручно використовувати дані з невизначеною структурою чи

різнотипними форматами – наприклад медіафайли. Також, є труднощі з масштабуванням схем для великих обсягів даних.

Підтримка стандартизованих мов для запитів, по типу SQL, дозволяє таким користувачам таких структур легко виконувати операції різних ступіней складності над даними – поєднання, сортування, агрегацію та фільтрування, тощо. Це робить обробку та отримання інформації досить швидкою.

Відповідність реальності також є важливою частиною таких моделей, відображаючи реальні зв’язки та об’єкти – в залежності від області, до якої відносяться дані, це можуть буті персональні дані клієнтів, інформація про замовлення чи предмети, тощо.

Популярними структурованими моделями, і які будуть описані нижче, є:

- ієрархічні моделі;
- мережеві моделі;
- реляційні моделі;
- об'єктно-орієнтовані моделі;
- документні моделі;
- ключ-значення моделі;
- графові моделі.

Основними серед них є ієрархічні, мережеві та реляційні, інші – менш широко використовані, але все одно існуючі моделі.

2.2 Ієрархічні моделі

Ієрархічна модель даних організована у вигляді дерева. Кожен елемент в ієрархії має один «батьківський» елемент (крім кореневого), і може мати кілька «дочірніх» елементів. Часто вони використовуються для папок та файлів в операційних системах девайсів – де кожен каталог може містити вкладені каталоги, але всі вони прив’язані до одного кореня, для

представлення підрозділів, відділів та команд компаній та установ, географічних та біологічних класифікацій, в системах управління документами бібліотек та архівів для сортування документів, книг та інших ресурсів. Цією моделлю користуються бази даних на основі IBM’s (International Business Machines) Information Management System (IMS), для зберігання даних в основних банківських та страхових системах. Загалом, ієрархічна модель ідеально підходить для представлення даних з природною ієрархією (каталоги, класифікація товарів, організаційні структури тощо).

Щодо даних, для яких ця модель погано підходить, то оскільки кожен елемент може мати лише одного батьківський, несумісними є ситуації, де потрібна більш складна взаємодія, наприклад у соціальних мережах або торгових платформах, де один елемент може бути пов'язаний із багатьма іншими елементами.

Серед основних принципів ієрархічних моделей можна визначити:

- деревоподібну структуру;
- підтримку відношень один-до-багатьох;
- швидкий доступ до даних;
- обмежені зв’язки.

Дані у такій структурі, як вже було сказано вище, представлені у вигляді дерева – починаючи з кореневого об’єкта, від якого розходяться зв’язки, що пов’язують батьківські об’єкти з дочірніми елементами. Кожен батьківський елемент може мати кілька дочірніх, але кожен дочірній елемент має лише один батьківський.

Оскільки ієрархічна модель має жорстку структуру, це ускладнює внесення змін. Наприклад, якщо потрібно додати новий тип зв'язку або змінити структуру, може знадобитися суттєво перебудувати модель. Якщо, за якихось причин дочірній елемент має бути пов’язаний більше чим з одним батьківським, головним перетворенням є заміна одного дерева

кількома, що призводить до надмірності в базі даних через дублювання даних.

Така модель забезпечує швидкий доступ до даних, оскільки структура передбачає єдиний шлях від кореня до кожного елемента. Це дозволяє швидко шукати та витягувати дані, особливо в сценаріях із частим доступом до певної частини ієрархії. Завдяки однозначному шляху від кореня до будь-якого елемента, доступ до даних здійснюється швидше, ніж у багатьох інших моделях. Це особливо важливо в сценаріях із великою кількістю звернень до певної частини ієрархії. Однак, при дуже глибоких ієрархіях робота з даними стає важчою, оскільки кожен елемент залежить від свого попередника. Це може ускладнювати навігацію, підтримку та масштабування такої структури.

Для візуального прикладу ієрархічної моделі можна навести рисунок 2.1, що у даному випадку відображає структуру сторінок теоретичного веб-сайту.

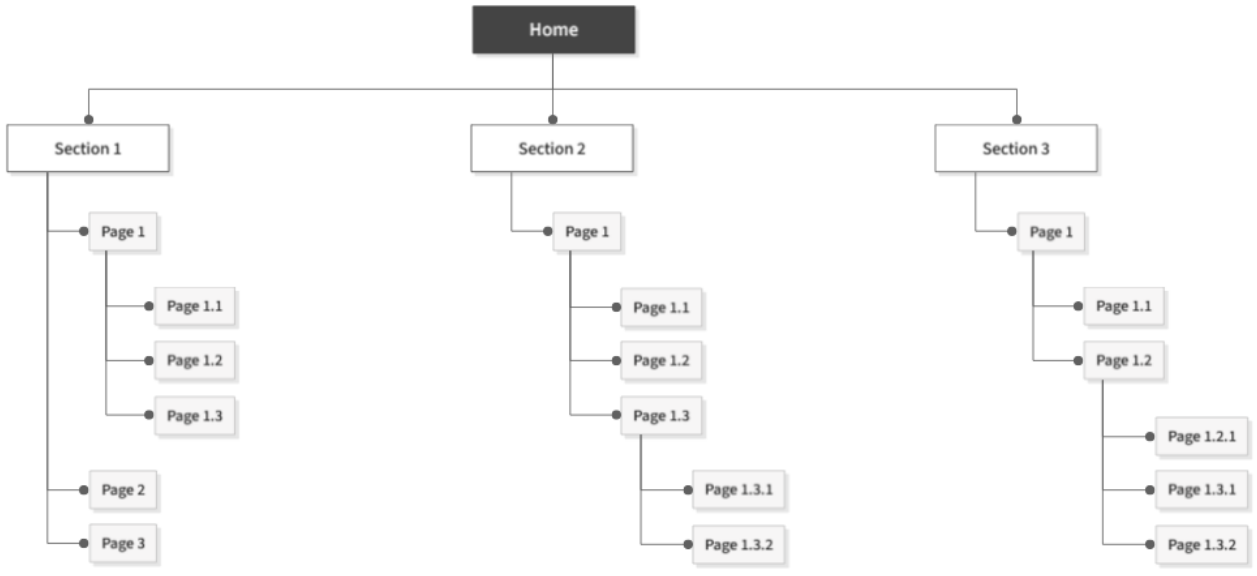

Рисунок 2.1 – Приклад ієрархічної моделі

На цьому зображенні можна побачити структуру даних, поданих у ієрархічній моделі.

До основних елементів ієрархічної структури належать рівень, елемент (вузол) і зв’язок. Вузол – це сукупність атрибутів, що описують певний об’єкт. На схемі ієрархічного дерева вузли зображені як вершини графа. Кожен вузол нижчого рівня пов’язаний лише з одним вузлом, розташованим на рівень вище.

Ієрархічне дерево має єдину кореневу вершину, що не підпорядкована жодній іншій і розташована на найвищому (першому) рівні. Підпорядковані вузли розміщені на другому, третьому, тощо рівнях. Для кожного запису бази даних існує лише один ієрархічний шлях від кореневого запису.

Загалом, ієрархічна модель даних, як і мережева (що буде розписана у наступному пункті), базується на графовій структурі, але є частковим випадком мережевої моделі на концептуальному рівні. У цій моделі вершини графу відповідають типам сегментів (або просто сегментам), а дуги представляють типи зв’язків предок – нащадок. В ієрархічній структурі кожен сегмент-нащадок має рівно одного предка.

Таким чином можна сказати, що ієрархічна модель – це зв'язний, неорієнтований граф з деревоподібною структурою, що поєднує сегменти. Ієрархічна база даних складається з впорядкованого набору дерев.

2.3 Мережеві моделі

У свою чергу, мережеві моделі даних – це моделі даних, в яких інформація організована у вигляді мережі або графу з множинними зв'язками між елементами. На відміну від попередньо описаної ієрархічної моделі, де кожен дочірній елемент має одного батьківського, у мережевій моделі кожен елемент може мати кілька батьківських елементів, що забезпечує більшу гнучкість і дозволяє ефективніше представляти складні зв’язки між об’єктами.

І ієрархічні і мережеві бази даних часто називають навігаційними через специфіку технології доступу до даних, яка використовується при

написанні програм обробки на мові маніпулювання даними. Доступ до даних по шляхах, які не були передбачені під час створення бази, може займати непропорційно багато часу.

Принцип навігації підвищує ефективність доступу до даних і скорочує час відповіді на запит, але водночас посилює залежність програм від структури даних. Програми обробки даних стають тісно пов’язаними з поточною структурою бази даних і потребують переписування при її зміні. Операції модифікації та видалення даних вимагають переналаштування покажчиків, а робота з даними залишається орієнтованою на записи. Крім того, навігаційний підхід обмежує можливість підвищення рівня мови маніпулювання даними, що робить його недоступним для користувачів без професійної підготовки. Для пошуку потрібного запису в ієрархічній або мережевій структурі програміст спочатку визначає шлях доступу, а потім покроково переглядає всі записи, що лежать на цьому шляху.

Використовуються такі моделі в системах, де потрібна підтримка складних зв’язків і високий рівень гнучкості. Наприклад, у банківських системах – одним з яких є Universal Datenbank System (UDS) від Siemens, наукових дослідженнях, тощо.

Серед основних принципів мережевих моделей можна визначити:

- структуру у вигляді графу;
- підтримку відношень багато-до-багатьох;
- підтримку відношень один-до-багатьох;
- підтримку складних структур даних.

Як і ієрархічна, мережева модель має структуру графа, що складається з вузлів (або записів) та зв’язків між ними. Кожен вузол представляє певний об’єкт або сутність, а зв’язки відображають відношення між ними. Це дозволяє моделювати відношення багато-до-багатьох, що є важливою особливістю мережевої моделі.

На відміну від ієрархічної моделі, що має в собі лише відношення один-до-одного та один-до-багатьох, мережева починає підтримувати

багато-до-багатьох. У мережевій моделі один елемент може бути пов'язаний з кількома іншими елементами, навіть із кількома батьківськими елементами. Це дозволяє краще представляти реальні, складніші взаємозв'язки.

Завдяки можливості створення множинних зв’язків між елементами, мережева модель може підтримувати більш складні структури даних, ніж ієрархічна модель. Це робить її ефективнішою для представлення об’єктів із широкими міжоб’єктними відносинами.

Все так же як і ієрархічна модель, завдяки прямим зв'язкам між елементами, мережева модель забезпечує швидкий доступ до пов'язаних даних, що підвищує продуктивність у системах із великим обсягом запитів на взаємопов'язані дані.

Однак, вона все одно складніша в проектуванні й управлінні, оскільки потребує детального налаштування множин і зв'язків. Це може ускладнювати розробку та підтримку бази даних. Зміни ж у моделі можуть призвести до необхідності оновлення численних зв'язків, що вимагає багато часу та зусиль, особливо в великих базах даних.

Мережеві моделі підходять для систем, де потрібно виконувати складні запити до даних з численними зв'язками між ними. Однак, їх важливим недоліком є те, що мережева модель не має такої поширеної підтримки мов запитів, як SQL у реляційній моделі, яка буде описана у наступному розділі.

Це може обмежити її використання, оскільки складні запити можуть вимагати специфічних мов або підходів до написання.

Для візуального прикладу мережевої моделі можна навести рисунок 2.2, що у даному випадку відображає теоретичний розподіл студентів між учбовими проектами за різними предметами.

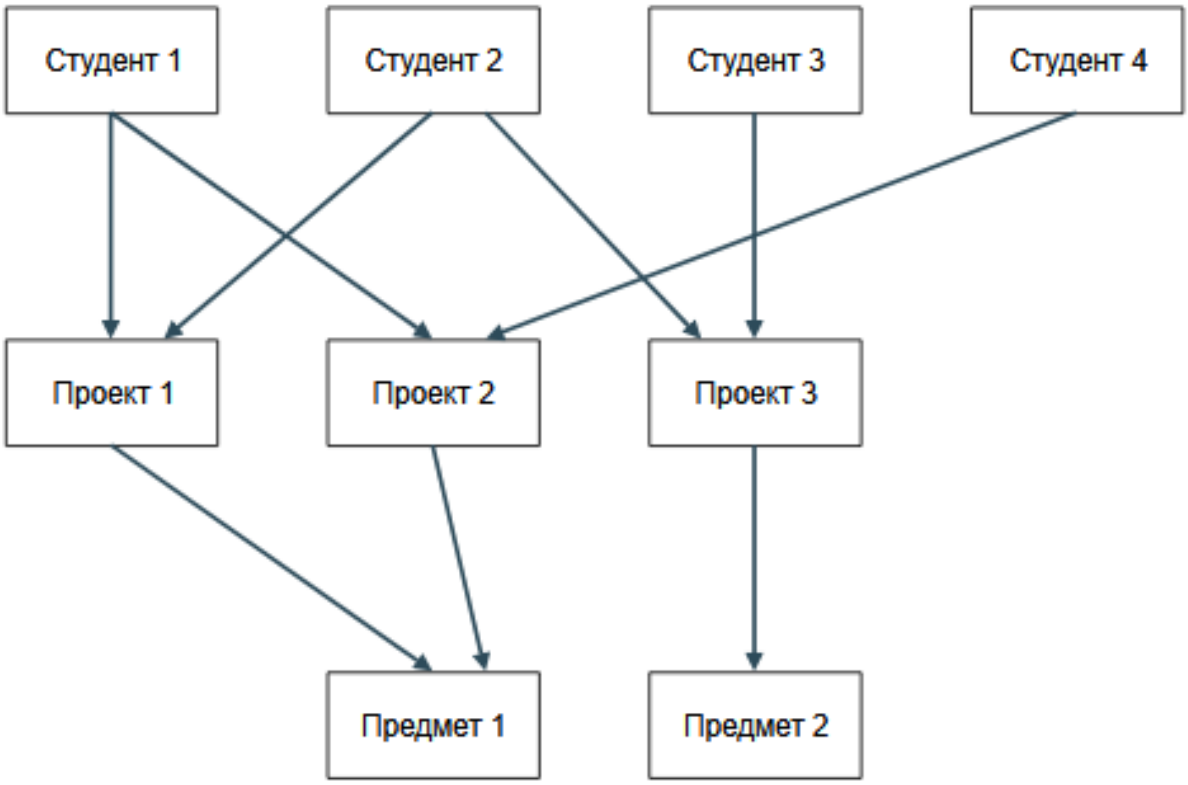

Рисунок 2.2 – Приклад мережевої моделі

На цьому зображенні можна побачити структуру даних, поданих у мережевої моделі. У порівнянні з попередньою ієрархічною структурою, тут можна побачити елементи з декількома батьківськими елементами, і усі зв’язки один-до-одного, один-до-багатьох, багато-до-одного та багато-до-багатьох.

2.4 Реляційні моделі

Реляційна модель даних, у свою чергу, є однією з найпопулярніших і широко використовуваних моделей для організації, зберігання та маніпулювання даними в базах даних. Вона заснована на концепції математичних відношень і забезпечує структурований, логічний і уніфікований спосіб роботи з даними. Вона використовується у безлічі різноманітних систем, що потребують структурованого та надійного

зберігання інформації. Реляційна модель даних є потужним і надійним інструментом для організації, управління та зберігання даних. Її структурованість, підтримка складних запитів, забезпечення цілісності даних і масштабованість роблять її однією з найпопулярніших моделей для бізнес-систем і додатків. Попри свої обмеження, реляційна модель залишається стандартом для багатьох організацій завдяки своїй ефективності та простоті інтеграції.

Серед основних понять реляційних моделей можна визначити:

- таблицю;
- атрибут;
- кортеж;
- первинний ключ;
- вторинний ключ;
- зовнішній ключ;
- відношення.

Дані зберігаються у вигляді таблиць, де кожна таблиця представляє собою сукупність пов’язаних даних, відому як «відношення». Кожен рядок таблиці називається «кортежем» і відображає один запис або об’єкт, а кожен стовпець – «атрибутом» або полем, що відповідає певній характеристиці об’єкта.

Атрибут – це окрема колонка таблиці, що визначає певну властивість або характеристику об’єктів, які описуються в цій таблиці. Атрибути мають тип даних (наприклад, число, текст, дата), що визначає допустимі значення.

Кожен рядок таблиці називається кортежем, і він представляє один конкретний запис. Кортеж містить значення для кожного атрибута таблиці.

Первинний ключ (Primary Key) – це атрибут або комбінація атрибутів, які унікально ідентифікують кожен запис у таблиці. Первинний ключ допомагає уникнути дублювання записів і є основою для встановлення зв'язків між таблицями.

Вторинний ключ (Secondary Key) – це атрибут, кожному значенню якого може відповідати більш ніж один екземпляр індексованих даних

Зовнішній ключ (Foreign Key) – це атрибут або набір атрибутів у таблиці, який створює зв’язок з іншою таблицею, посилаючись на її первинний ключ. Зовнішні ключі дозволяють створювати зв’язки між таблицями, забезпечуючи зв’язність даних.

Відношення в реляційній моделі означає зв'язок між різними таблицями. Ці зв'язки можуть бути типу «один-до-одного», «один-до-багатьох» або «багато-до-багатьох». Відношення допомагають організувати дані так, щоб уникнути надлишковості.

Реляційна модель використовує мову SQL (Structured Query Language) для доступу, маніпулювання та управління даними, який є стандартною мовою для взаємодії з реляційними СУБД (системами управління базами даних.

Основними операціями над даними в SQL є:

- витяг (SELECT);
- поєднання (JOIN);
- видалення (DELETE);
- оновлення (UPDATE);
- додавання (INSERT).

Операція SELECT використовується для отримання даних з однієї або декількох таблиць. Вона дозволяє вибирати потрібні стовпці, фільтрувати рядки, застосовувати функції, сортувати результати, а також групувати записи для підрахунку або агрегування.

Операція JOIN дозволяє об’єднувати рядки з двох або більше таблиць на основі пов’язаних між ними стовпців. Вона може бути використана для отримання повної картини даних, що розподілені по різних таблицях. Існують різні типи поєднань, наприклад INNER JOIN, LEFT JOIN, RIGHT JOIN і FULL JOIN, кожне з яких обробляє відсутні дані по-своєму.

Операція DELETE використовується для видалення рядків із таблиці, які відповідають умовам, заданим у команді. При видаленні можна вказати умови WHERE, щоб уникнути видалення всіх даних з таблиці.

Операція UPDATE використовується для оновлення існуючих записів у таблиці. За допомогою SET задаються значення, які потрібно змінити, а WHERE дозволяє вибрати конкретні рядки для оновлення.

Операція INSERT використовується для додавання нових записів у таблицю. За допомогою INSERT можна додавати як один запис, так і кілька одночасно.

Для візуального прикладу реляційної моделі можна навести рисунок 2.3, що у даному випадку відображає теоретичну схему бази даних з даними про робітників якоїсь фірми та її асети.

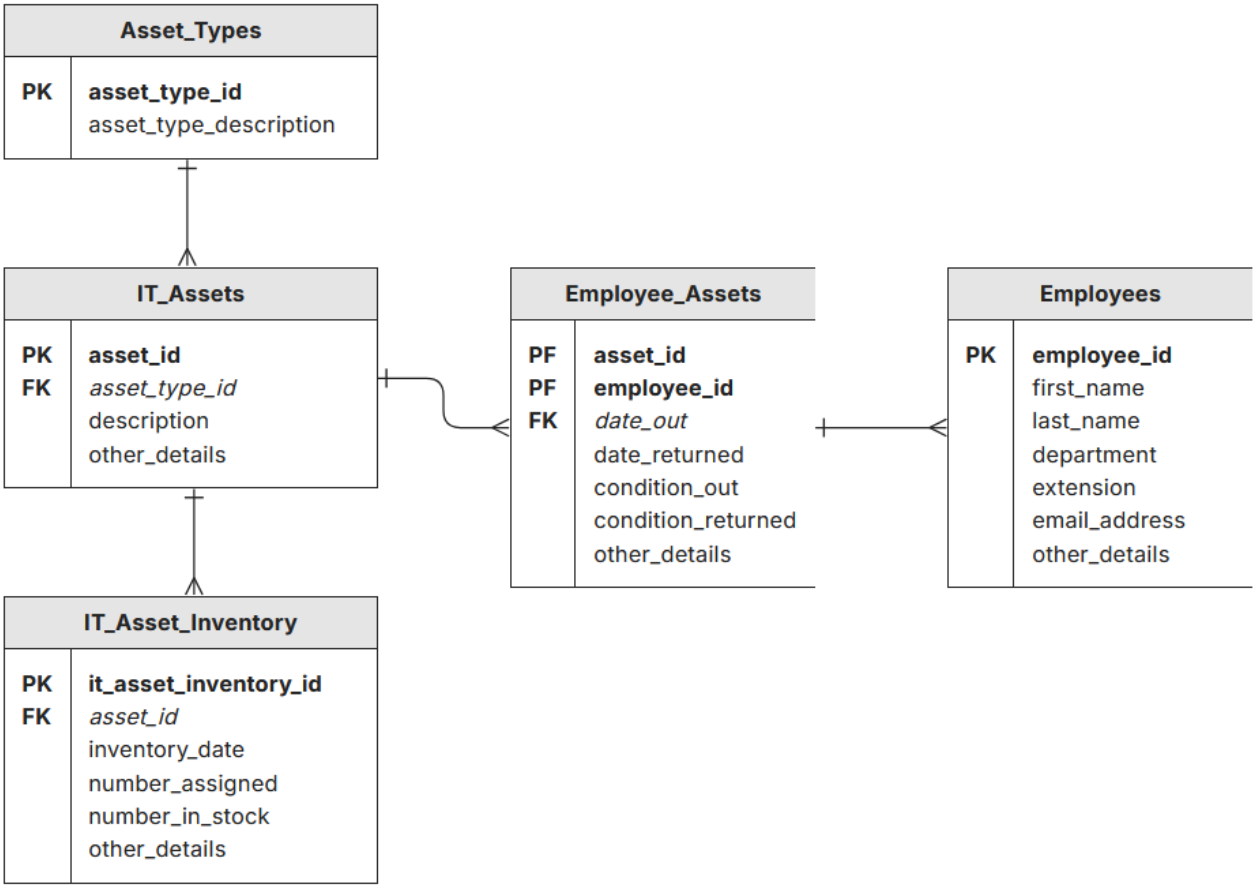

Рисунок 2.3 – Приклад реляційної моделі

На цьому зображенні можна побачити структуру даних, поданих у реляційній моделі. На прикладі цієї теоретичної бази даних можна побачити використання головних елементів реляційної структури, а саме таблиць, кортежів, атрибутів, первинних та зовнішніх ключей, а також відношень, у даному випадку – один-до-багатьох.

Реляційна модель даних зазвичай включає теорію нормалізації. За визначенням Крістофера Дейта, реляційна модель даних складається з таких трьох основних компонентів:

- структурний;
- маніпуляційний;
- цілісний.

Структурний компонент визначає, що основною структурою є нормалізоване n-арне відношення. Це відношення можна уявляти як таблицю, де рядки – це кортежі, а стовпці – атрибути, кожен з яких визначений на певному домені.

Його можна представити як відношення:

$$R \subseteq D_1 \times D_2 \times \cdots \times D_n, \tag{2.1}$$

де D_i – домен i-го атрибута, тобто множина можливих значень для цього атрибута;

R – відношення, що є скінченною множиною k-кортежів, а саме:

$$R = \{t_1, t_2, \cdots, t_k\}, \tag{2.2}$$

де кожен кортеж t – це впорядкована n-ка:

$$t = \{v_1, v_2, \cdots, v_n\},\ v_i \in D_i\ . \tag{2.3}$$

Кожен кортеж t відповідає рядку в таблиці, а кожен атрибут – стовпцю.

Такий підхід дає інтуїтивно зрозуміле представлення реляційної бази даних у вигляді кінцевого набору таблиць.

Маніпуляційний компонент задає два ключові механізми для роботи з даними: реляційну алгебру і реляційне числення. Цей компонент гарантує, що будь-яка мова реляційних баз даних має рівень виразності, не менший за реляційну алгебру або реляційне числення.

Реляційна алгебра визначає операції над відношеннями. Основні операції мають у собі об’єднання, перетин, різницю, декартовий добуток, проекцію та вибірку.

Формула об’єднання:

$$R_1 \cup R_2 = \{t \mid t \in R_1 \text{ або } t \in R_2\}. \quad (2.4)$$

Формула перетину:

$$R_1 \cap R_2 = \{t \mid t \in R_1, t \in R_2\}. \quad (2.5)$$

Формула різниці:

$$R_1 - R_2 = \{t \mid t \in R_1, t \notin R_2\}. \quad (2.6)$$

Формула декартового добутку:

$$R_1 \times R_2 = \{(t_1, t_2) \mid t_1 \in R_1, t_2 \in R_2\}. \quad (2.7)$$

Формула проекції:

$$\pi_{A_1,A_2,\dots,A_m}(R) = \{t[A_1, A_2, \dots, A_m] \mid t \in R\}, \quad (2.8)$$

де $A_1, A_2, \ldots, A_m$ – підмножина атрибутів відношення R.

Формула вибірки:

$$\sigma_{\text{умова}}(R) = \{t \mid t \in R \text{ і умова}(t)\text{виконується}\}. \quad (2.9)$$

Реляційне числення визначає запити як логічні формули. Наприклад:

$$\{t \mid \phi(t)\}, \quad (2.10)$$

де ϕ(t) – логічна формула, що описує умови, яким має відповідати кортеж t.

Цілісний компонент включає вимоги до цілісності сутностей і посилань. Цілісність сутностей означає, що кожен кортеж у відношенні унікальний, що забезпечується первинним ключем.

Нехай P – набір атрибутів, які утворюють первинний ключ, тоді:

$$\forall t_1, t_2 \in R, t_1[P] = t_2[P] \Longrightarrow t_1 = t_2. \quad (2.11)$$

Цілісність посилань (або вимога зовнішнього ключа) вимагає, щоб значення зовнішнього ключа (у формулі – FK), яке вказує на інше відношення, знаходило відповідний кортеж з таким самим первинним ключем (у формулі – PK) або залишалося порожнім (без зв'язку, у формулі – $NULL$). А саме, маємо формулу:

$$\forall t \in R_1, t[FK] \neq NULL \Longrightarrow \exists t' \in R_2, t[FK] = t'[PK]. \quad (2.12)$$

Таким чином, реляційна модель даних базується на строгій математичній основі, використовуючи множини, функції та логіку для визначення структур, операцій і обмежень, що забезпечує високий рівень формалізму і надійності в управлінні даними.

2.5 Об'єктно-орієнтовані моделі

Об’єктно-орієнтована модель даних – ООМД – є підходом до організації та маніпуляції даними, що базується на принципах об'єктно-орієнтованого програмування. Вона забезпечує представлення даних у вигляді об'єктів, що дозволяє моделювати реальні об'єкти та їхні зв'язки більш природним способом у порівнянні з традиційними реляційними моделями, описаними у розділі вище. Цей підхід дозволяє використовувати наслідування, інкапсуляцію та поліморфізм для більшої гнучкості і зручності структури бази даних. Її гнучкість і здатність працювати з об'єктами роблять об’єктно-орієнтовану модель даних цінною для розробників, які прагнуть відобразити реальні об'єкти й їхню поведінку в базах даних. Проте складність у реалізації та обмеження в продуктивності роблять її менш популярною для проектів, де потрібна швидка обробка великих обсягів даних.

Зазвичай об’єктно-орієнтовані моделі використовуються для зберігання комп’ютерних файлів зображень, відео та аудіо, для зберігання даних про складні об’єкти, для роботи з інженерними об’єктами, кресленнями та проектами, для об’єктів з оновленням даних в реальному часі.

Існує кілька типів об'єктно-орієнтованих баз даних. Мультимедійна база даних включає медіафайли, такі як зображення, які не можуть зберігатися в реляційній базі даних.

Гіпертекстова база даних дозволяє зв’язувати будь-який об’єкт з будь-яким іншим об’єктом. Це корисно для організації великої кількості різнорідних даних, але не ідеально для числового аналізу.

Об’єктно-орієнтована модель бази даних є найвідомішою постреляційною моделлю бази даних, оскільки вона включає таблиці, але не

обмежується ними, як реляційна. Такі моделі також відомі як гібридні моделі баз даних.

Серед основних понять об’єктно-орієнтованих моделей можна визначити:

- об'єкт;
- клас;
- екземпляр класу;
- наслідування;
- інкапсуляцію;
- поліморфізм.

Об'єкти є основними будівельними блоками об'єктно-орієнтованої моделі. Кожен об'єкт складається з властивостей (атрибутів) і методів (функцій), які визначають його поведінку.

Клас – це шаблон або «схема» для створення об'єктів з певним набором властивостей і методів. Клас визначає загальну структуру та поведінку для групи схожих об'єктів.

Клас дозволяє інкапсулювати дані (властивості або атрибути) та функціональність (методи), що працюють із цими даними, у межах єдиної сутності. Завдяки цьому об'єкти, створені на основі класу, можуть зберігати стан і виконувати операції, характерні для своєї природи.

Класи є фундаментальною концепцією в об’єктно-орієнтованому програмуванні, що дозволяє створювати модульний, зрозумілий і повторно використовуваний код.

Екземпляр класу – це конкретний об'єкт, створений на основі класу. Кожен екземпляр зберігає свої власні значення атрибутів, хоча всі екземпляри одного класу мають спільну структуру і методи.

Для візуального прикладу об’єктно-орієнтованої моделі можна навести рисунок 2.4, взятий з сайту lucidchart з статті про типи баз даних [8], що у даному випадку відображає теоретичну схему з об’єктами і даними на тему продажі-купівлі.

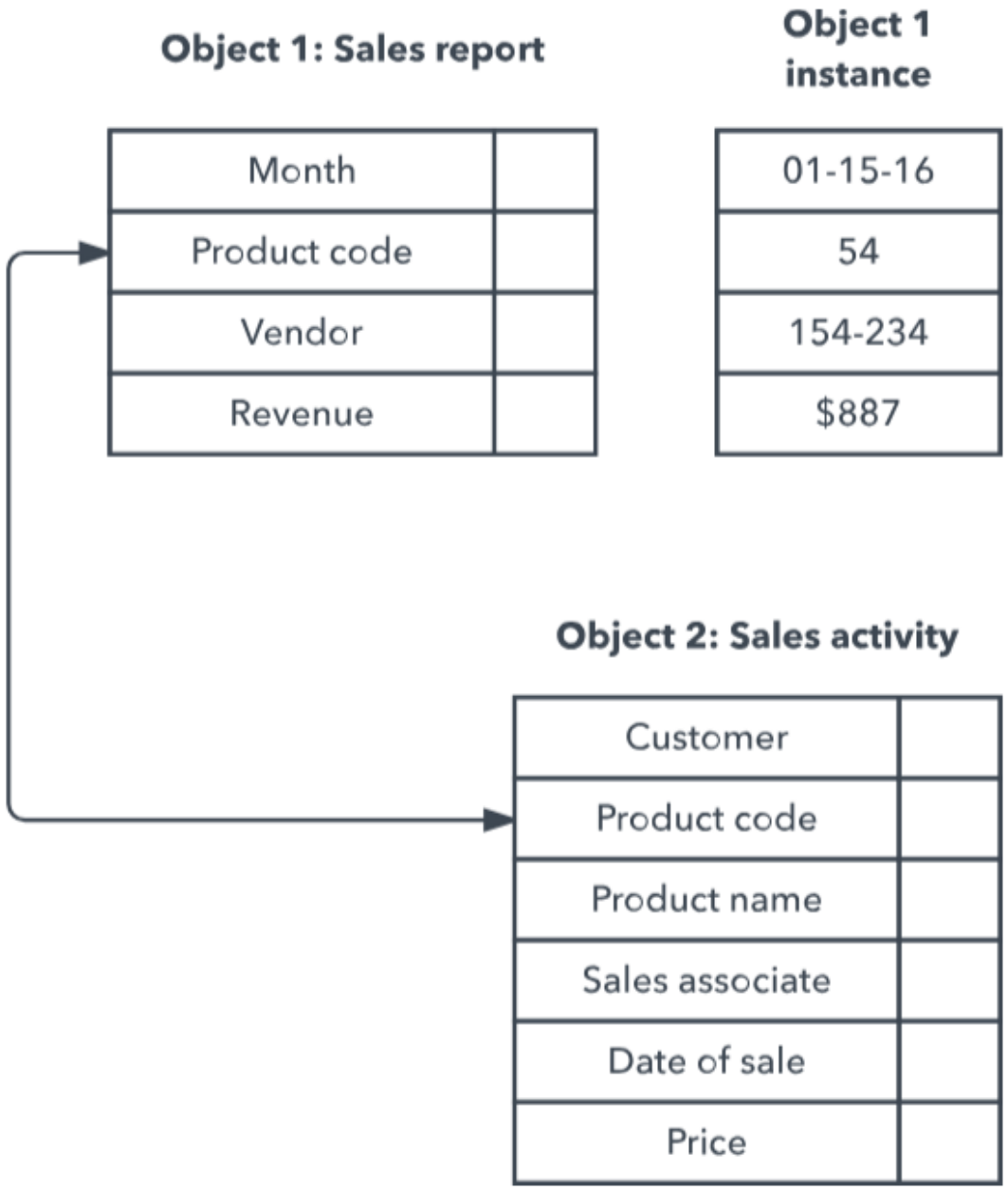

Рисунок 2.4 – Приклад об’єктно-орієнтованої моделі

На цьому зображенні можна побачити структуру даних, поданих у об’єктно-орієнтованій моделі.

Наслідування дозволяє створювати нові класи на основі існуючих, успадковуючи їхні властивості та методи. Це дозволяє уникати дублювання коду. Наслідуючий клас може успадковувати властивості і методи першого класу, але також мати власні унікальні атрибути.

Інкапсуляція обмежує доступ до внутрішніх даних об'єктів, дозволяючи звертатися до них тільки через певні методи. Це сприяє захисту даних і запобігає їхньому випадковому або небажаному зміненню.

Поліморфізм дозволяє використовувати один і той самий метод для різних об'єктів або класів. Це підвищує гнучкість і дозволяє виконувати дії з різними типами об'єктів у єдиний спосіб.

Наслідування та поліморфізм сприяють повторному використанню коду, що знижує витрати на розробку та обслуговування.

Об’єктно-орієнтована модель даних дозволяє моделювати складні системи та реальні об'єкти більш природно. Класи і наслідування полегшують роботу з даними, особливо для великих, складних проектів. Вона підтримує зберігання складних і вкладених типів даних, таких як мультимедійні об'єкти, зображення, документи тощо.

У об'єктно-орієнтованій моделі зв’язки між об'єктами можуть відображати складні взаємовідносини, що є аналогом зв'язків між таблицями в реляційній моделі. Вони відображають різний ступінь залежності між об'єктами, який варіюється від слабкої асоціації до тісного зв’язку композиції.

Серед основних зв’язків між об’єктами у об’єктно-орієнтованих моделей можна визначити:

- асоціацію;
- композицію;
- агрегацію.

Асоціація є базовим типом зв’язку між об’єктами, коли один об'єкт пов’язаний з іншим без жорсткої залежності. У цьому випадку об’єкти можуть існувати незалежно один від одного. Асоціація дозволяє одному об’єкту використовувати інший для виконання своїх завдань, але зв’язок між ними не є обов’язковим для існування цих об'єктів. Асоціація може мати різну множинність: один-до-одного, один-до-багатьох та багато-до-багатьох.

Агрегація представляє собою слабший тип зв’язку, при якому один об’єкт є частиною іншого, але при цьому може існувати незалежно. Це відображає стосунки «частина-ціле», коли компоненти можуть існувати окремо від контейнера, до якого вони входять. Агрегація вказує на те, що об'єкти мають логічний зв’язок, але їх існування не залежить одне від одного.

Композиція, у свою чергу, – це найсильніший тип зв’язку між об'єктами, який показує стосунки «ціле-частина», при яких частини не можуть існувати окремо від цілого. У випадку композиції компоненти створюються і знищуються разом із головним об'єктом. У композиції підлеглі об’єкти не мають незалежного існування від головного об'єкта – залежність між об'єктами є такою тісною, що їх не можна розділити.

Об'єктно-орієнтована модель може бути складною у впровадженні і підтримці через необхідність підтримувати об'єкти і їхні зв'язки в актуальному стані. Вона може мати меншу продуктивність при обробці великих обсягів даних у порівнянні з реляційними базами. Оптимізація запитів і взаємодії між об'єктами може бути складнішою.

Дуже важливим недоліком є то, що, на відміну від описаної вище реляційної моделі, яка використовує стандартизовану мову SQL, об'єктно-орієнтовані бази даних не мають єдиного загальноприйнятого стандарту для мови запитів. Це може створювати труднощі з перенесенням даних між різними базами та з загальною зручністю обробки даних.

Однак, об'єктно-орієнтовані моделі добре інтегруються з об'єктно-орієнтованими мовами програмування, такими як Java, C++, Python тощо, що дозволяє розробникам працювати з базами даних у звичній парадигмі, незважаючи на проблеми з використанням SQL.

2.6 Документні моделі

Переходячи до менш популярних моделей, і, в даному випадку, NoSQL моделі, наступними є документні моделі. Документні моделі даних є одним з типів моделей NoSQL, що орієнтуються на зберігання даних у вигляді документів, зазвичай в форматах JSON, BSON, XML або подібних, а також у бінарних формати, такі як PDF чи документи Microsoft Office (MS Word, Excel, тощо). У таких базах даних інформація зберігається в структурованих або напівструктурованих документах, які об'єднують дані у вкладені структури – ключі та значення. Це дозволяє зберігати складну ієрархію даних без потреби в суворій схемі, як у реляційних базах.

Часто такі системи використовуються для комерційного зберігання інформації про продукти, їхні характеристики, описи та відгуки, що можуть значно варіюватися від продукту до продукту, соціальних мереж – з профілями, що містять багато різних елементів та налаштувань, для зберігання статей, сторінок тощо. Документні моделі даних є ідеальним вибором для застосунків з динамічною структурою даних, де зміни відбуваються часто. Гнучкість і масштабованість документних баз дозволяють забезпечити високу продуктивність та адаптованість до нових вимог без значних витрат на реорганізацію даних.

Серед основних наборів способів організації документних моделей можна визначити:

- колекції;
- теґи;
- невидимі метадані;
- ієрархію директорій;
- комірки.

Для візуального прикладу документної моделі можна навести рисунок 2.5, взятий з сайту blog.usu з статті про документні бази даних [9],

що у даному випадку відображає теоретичну схему інформації про людину, порівнюючи SQL відображення з NoSQL документом.

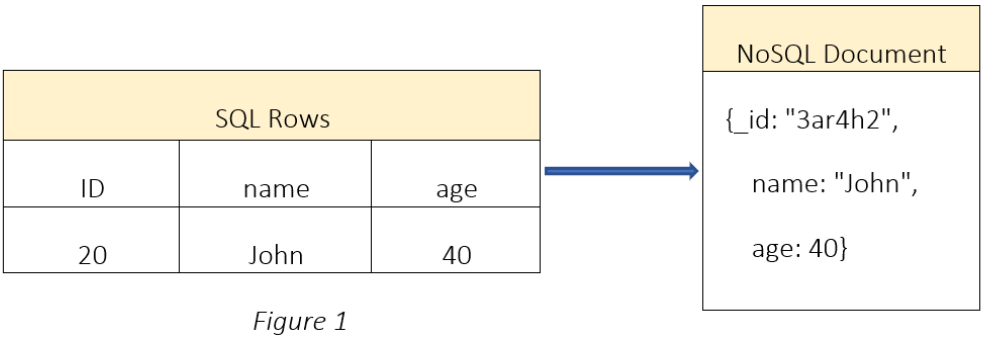

Figure 1

Рисунок 2.5 – Приклад документної моделі

На цьому зображенні можна побачити структуру даних, поданих у документній моделі.

Серед основних принципів документних моделей можна визначити:

- гнучку структуру даних;
- ієрархічні та вкладені структури;
- відсутність жорсткої схеми;
- високу масштабованість і продуктивність.

Документні бази не вимагають фіксованої структури, тому різні документи можуть мати різні набори полів. Це означає, що кожен документ можна адаптувати під специфічні дані, що зберігаються, не змінюючи загальну структуру бази. Це особливо корисно в динамічних системах, де дані часто змінюються.

Дані зберігаються у вигляді вкладених структур, що дозволяє відображати складні об'єкти в одному документі. Деякі документні бази можуть мати обмежені можливості для складних запитів, таких як приєднання кількох документів, що є більш простим у реляційних базах даних.

Документні моделі не мають обов'язкової схеми, що дозволяє додавати нові поля в документ без необхідності змінювати всі інші записи і дає більшу гнучкість і можливість швидко адаптуватися до нових вимог. Це дозволяє швидко вносити зміни в модель даних без складних міграцій, як у реляційних базах даних. Однак, через відсутність суворої схеми та жорсткої нормалізації можлива наявність дубльованих або непослідовних даних.

Також, документні бази даних можуть обробляти великі обсяги даних, розподілені по кількох серверах. Горизонтальне масштабування дозволяє збільшувати обсяг даних і обробку запитів, додаючи нові сервери до системи.

Документні бази рідко підтримують транзакції на рівні кількох документів або колекцій, що може бути недоліком для систем, де потрібна висока надійність транзакцій.

Поширеними операціями в документних базах даних є створення нового документа у колекції, читання документа за допомогою унікального ідентифікатора або запитів, що підтримують фільтрацію та сортування, оновлення документа за допомогою додавання нових полів чи зміни значень та видалення одного або кількох документів на основу умов фільтрації.

Головними, та сучасними документними базами даних є MongoDB [10], Amazon DocumentDB [11], CouchDB [12] та Couchbase [13]. Вони відрізняються підходами до масштабування, зберігання, реплікації та обробки даних і підходять для різних типів застосунків, які вимагають високої гнучкості та продуктивності.

MongoDB – це гнучка документно-орієнтована NoSQL база даних, яка зберігає дані у форматі BSON (схожого до більш широко використовуваного JSON), що робить її зручною для зберігання складних та вкладених даних. Вона підтримує шардінг для горизонтального масштабування, реплікацію для високої доступності та інтеграцію з різними мовами програмування. MongoDB ідеально підходить для сучасних веб- та мобільних застосунків, які потребують динамічних схем даних.

Amazon DocumentDB – це керована документна база даних від AWS, сумісна з API MongoDB, що дозволяє легко переносити MongoDB-застосунки. DocumentDB забезпечує автоматичне масштабування, резервне копіювання та інтеграцію з AWS-сервісами, пропонуючи високу безпеку та зручність адміністрування для додатків у хмарі. Це рішення підходить для робочих навантажень, які потребують високої надійності та мінімального адміністрування.

CouchDB – це документно-орієнтована NoSQL база даних від Apache, яка використовує JSON для зберігання даних, JavaScript для обробки запитів та HTTP як інтерфейс API. Вона підтримує принципи реплікації та синхронізації, що робить її чудовим вибором для розподілених систем, де потрібно забезпечити синхронізацію даних між різними пристроями чи серверами, навіть за наявності часткових відключень мережі. CouchDB реалізує концепцію «Master-Master» реплікації, що дозволяє одночасно записувати дані у різні екземпляри бази без конфліктів. Завдяки цьому, CouchDB ідеально підходить для мобільних та веб-додатків, де критично важлива офлайн-підтримка з подальшою синхронізацією.

Couchbase же, у свою чергу, – це високопродуктивна NoSQL база даних, яка поєднує в собі можливості роботи з документами та кешування. Вона забезпечує низьку затримку та високий рівень доступності через реплікацію і кластеризацію, що робить її особливо зручною для мобільних та IoT-додатків. Couchbase підтримує SQL-подібні запити (N1QL) для роботи з JSON, що спрощує роботу з даними.

2.7 Ключ-значення моделі

Продовжуючи перелік NoSQL моделей, ключ-значення (Key-Value) модель даних є однією з найпростіших моделей зберігання у NoSQL базах даних. Вона передбачає зберігання даних у вигляді пар «ключ-значення», де кожному унікальному ключу відповідає певне значення. Цей підхід є

ефективним для систем, які потребують швидкого доступу до даних і високої продуктивності при великих обсягах інформації.

Структура моделі є найпростішою, маючи в собі пари ключей та значень, де ключ – унікальний ідентифікатор (часто представлений як рядок), який використовується для пошуку відповідного значення, а значення може бути будь-яким типом даних, включаючи рядки, числа, JSON-документи, зображення або навіть складні об’єкти. Значення не має обмеженої структури і може бути вільноформатним, що дозволяє зберігати складні і вкладені дані.

Для візуального прикладу ключ-значення моделі можна навести рисунок 2.6, взятий з сайту redis.io з статті про ключ-значення бази даних [14], що у даному випадку відображає теоретичну схему інформації про номер телефону людей.

Рисунок 2.6 – Приклад ключ-значення моделі

На цьому зображенні можна побачити структуру даних, поданих у ключ-значення моделі.

Основні операції, такі як зчитування, додавання або видалення записів, виконуються за допомогою ключів. Це забезпечує дуже швидкий

доступ, оскільки операції доступу за ключем мають складність O(1) в багатьох реалізаціях, що є суттєвою перевагою при роботі з великими обсягами даних.

Модель не вимагає фіксованої структури значень. Кожне значення може бути унікальним за своєю структурою і не обов’язково узгоджуватися з іншими значеннями у базі. Це дозволяє гнучко зберігати різнорідні дані, що особливо корисно для застосунків, де структура інформації може динамічно змінюватися.

Ключ-значення бази даних легко масштабуються горизонтально, тобто шляхом додавання нових серверів до кластера. Це досягається за допомогою розподіленого хешування або шардінгу, де дані розподіляються на основі хеш-функції ключів, що дозволяє балансувати навантаження між вузлами.

Завдяки своїй простоті, бази даних типу ключ-значення є дуже швидкими і можуть обробляти великий обсяг операцій введення-виведення за секунду. Вони підходять для випадків, де важливі низька затримка та висока пропускна здатність, наприклад, для кешування веб-контенту або для обробки аналітичних даних у реальному часі.

Серед основних сценаріїв використання ключ-значення моделей можна визначити:

- кешування;
- сеансні дані;
- конфігураційні дані;
- обробка аналітики в реальному часі.

Якщо розповідати по порядку, то ключ-значення бази, такі як Redis та Memcached, часто використовуються для кешування, щоб швидко витягати дані, які нечасто змінюються, але часто запитуються. Кешування покращує продуктивність системи, зменшуючи кількість звернень до основної бази даних або зовнішніх API.

У багатьох веб-додатках користувацькі сесії зберігаються в ключ-значення базах, що дозволяє швидко записувати й зчитувати дані користувача під час взаємодії з додатком. Це забезпечує високу швидкодію і дозволяє зберігати індивідуальні налаштування чи інші тимчасові дані сесії.

Ключ-значення бази зручні для зберігання конфігураційних параметрів, які використовуються у багатьох додатках, таких як налаштування компонентів або змінні середовища, які повинні бути швидкодоступними для програми.

Бази даних ключ-значення добре підходять для збору та обробки даних аналітики, де потрібно швидко фіксувати численні події (наприклад, кліки користувачів або перегляди сторінок) з подальшим аналізом.

Однак, ключ-значення моделі не підходять для складних запитів чи аналізу, оскільки вони не підтримують складні структури відносин між даними. Також, модель є недостатньо гнучкою для використання у випадках, де потрібні зв'язки між даними, такі як у реляційних базах даних.

Серед основних ключ-значення баз даних можна визначити Redis [14], Memcached [15], RocksDB [16] та DynamoDB [17].

Redis – це високопродуктивна база даних ключ-значення зберігання в оперативній пам'яті, яка підтримує різноманітні структури даних, такі як списки, множини та хеші. Вона використовується для кешування, обробки черг та обчислень у реальному часі. Redis забезпечує низьку затримку і високу швидкість, що робить її чудовим вибором для програм із високими вимогами до продуктивності.

Memcached – це простий і ефективний кеш у пам’яті, призначений для прискорення роботи веб-додатків за рахунок зниження навантаження на основні бази даних. Memcached зберігає тимчасові дані у вигляді ключ-значення і використовується для швидкого доступу до часто запитуваної інформації. Воно підходить для додатків, яким необхідна проста кешуюча система без додаткових складних структур даних.

RocksDB – це вбудована база даних, створена на основі LevelDB, яка оптимізована для швидкого зберігання на твердотільних накопичувачах (SSD). Вона підтримує великі обсяги записів і особливо підходить для робочих навантажень з високою частотою записів, таких як логування та потокові дані. RocksDB використовується у багатьох системах для зберігання даних на диску із низькою затримкою та високою пропускною здатністю.

Amazon DynamoDB – це керована, розподілена база даних від AWS, яка підтримує ключ-значення та вище описану документну моделі. Вона забезпечує горизонтальне масштабування і автоматичне керування даними, що робить її ідеальною для великих застосунків із динамічними навантаженнями. DynamoDB популярна для зберігання великих обсягів даних і має функції для забезпечення високої доступності та надійності.

2.8 Графові моделі

Остання у переліку модель – графова. Графові моделі даних – це тип NoSQL баз даних, що використовують графову структуру для зберігання та обробки даних. У графових моделях дані представлені у вигляді графів, де вузли (вершини) графа представляють сутності (об'єкти), а ребра (з'єднання між вузлами) описують зв'язки між цими сутностями. Така структура дозволяє ефективно моделювати і працювати з даними, що мають складні взаємозв'язки, такі як соціальні мережі, рекомендаційні системи, обробка шляхів, логістика тощо. Система підходить для представлення складних відносин між даними, таких як родинні зв'язки, соціальні мережі або ланцюги постачань.

Оскільки графові бази даних оптимізовані для роботи з великими обсягами зв'язків, вони дозволяють швидко знаходити шляхи між вузлами, що є важливим для таких задач, як пошук друзів у соцмережах, для виявлення найкоротших шляхів або пошуку маршрутів у мережах

транспортних або логістичних компаній, або рекомендації на основі схожості між користувачами чи продуктами.

Серед основних компонентів графових моделей можна визначити:

- Вузли;
- Ребра;
- Атрибути.

Для візуального прикладу ключ-значення моделі можна навести рисунок 2.7, взятий з публікації «Application of graph databases for transport purposes» [18], що у даному випадку відображає теоретичну схему інформації про номер телефону людей.

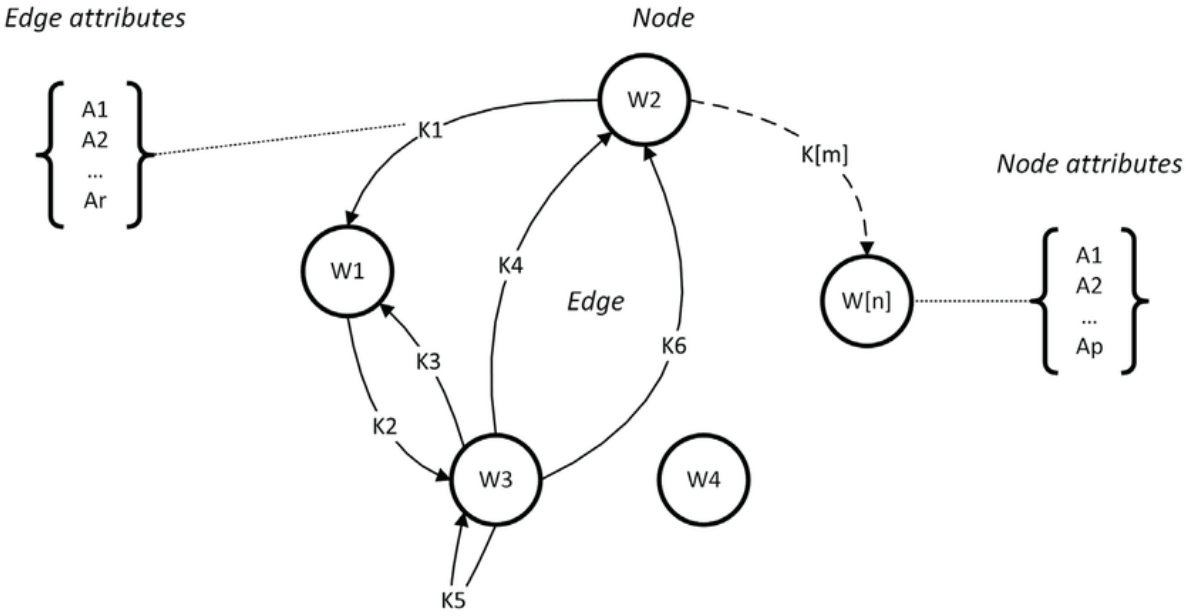

Рисунок 2.7 – Приклад графової моделі

На цьому зображенні можна побачити структуру даних, поданих у графовій моделі.

Кожен вузол представляє об'єкт або сутність у системі. Ребра представляють зв'язки між вузлами. Вони можуть бути орієнтованими (від одного вузла до іншого) або неорієнтованими. Як вузли, так і ребра можуть мати атрибути, які описують додаткову інформацію про зв'язки чи сутності.

Графові моделі легко розширюються для нових типів зв'язків та сутностей, що робить їх корисними для динамічних, швидко змінюваних систем.

Графові бази даних можуть мати вищу складність налаштування та адміністрування порівняно з традиційними реляційними базами даних, а також можуть бути менш ефективними для простих або структурованих даних, де зв'язки між елементами не є критичними.

Серед основних графових баз даних можна визначити Neo4j [19], Amazon Neptune [20], ArangoDB [21] та JanusGraph [22].

Neo4j – це одна з найбільш популярних графових баз даних, яка використовує модель графа для зберігання та обробки даних. Вона використовує спеціалізовану мову запитів Cypher для створення складних запитів і має високу продуктивність при обробці великих обсягів взаємозв'язків. Neo4j часто використовується для аналізу соціальних мереж, рекомендаційних систем і виявлення прихованих патернів у даних.

Amazon Neptune – це керована графова база даних від AWS, яка підтримує дві популярні моделі графів: Property Graph та RDF (Resource Description Framework). Вона оптимізована для масштабованості та інтеграції з іншими сервісами AWS, що робить її ідеальним вибором для організацій, що потребують надійної, безперервної роботи з графами у хмарі. Neptune підтримує високу доступність і автоматичне масштабування.

ArangoDB – це мульти-модельна база даних, яка поєднує графову модель, документну та модель ключ-значення. Вона дозволяє ефективно працювати з різними типами даних і забезпечує високу продуктивність при збереженні гнучкості в проектуванні схем. ArangoDB використовується для складних додатків, таких як аналітика великих даних, графові запити та зберігання великих наборів документів.

Нарешті, JanusGraph – це розподілена графова база даних з відкритим кодом, яка підтримує масштабування через різні системи зберігання, такі як Apache Cassandra, HBase або Google Cloud Bigtable. Вона спеціалізується на

обробці великих графів з мільйонами вузлів і ребер, і є ідеальним вибором для складних сценаріїв, де необхідно забезпечити надійність, масштабованість і високу продуктивність. JanusGraph підтримує популярні бібліотеки для аналізу графів, такі як Apache TinkerPop.

2.9 Нечіткі моделі в структурованих базах даних

Нечіткі моделі в структурованих базах даних є потужним інструментом для роботи з неточними та неповними даними, які часто зустрічаються у реальних системах. На відміну від традиційних чітких (класичних) моделей баз даних, де кожен елемент або належить певному класу, або не належить, нечіткі моделі дозволяють враховувати ступінь належності та невизначеність.

Нечітка логіка була розроблена для роботи з інформацією, яка є нечіткою або неоднозначною. Вона ґрунтується на понятті нечітких множин, де кожен елемент має певну ступінь належності (зазвичай в межах [0,1]), а не просто належить чи не належить множині.

У класичних реляційних базах даних таблиці містять чітко визначені дані (наприклад, чіткі значення «істина» чи «хибність»). Натомість у нечітких базах даних значення атрибутів можуть мати певну ймовірність або ступінь впевненості.

Нечіткі моделі дозволяють враховувати інформацію, яка не є точною або повною, наприклад, нечіткі часові інтервали, приблизні значення кількісних характеристик тощо.

Також, нечіткі моделі дозволяють працювати з частково відомими або двозначними даними. Наприклад, у медичних базах даних для діагностики можуть бути використані нечіткі моделі, що враховують можливість різних станів пацієнта на основі нечітких симптомів.

В умовах невизначеності нечіткі бази даних можуть допомогти в прийнятті рішень, оскільки вони дозволяють оперувати градієнтами належності, а не просто булевими значеннями.

Однак, обробка нечітких даних потребує додаткових обчислень та спеціальних алгоритмів для інтерпретації нечітких значень, а у великих базах даних обробка нечітких запитів може вимагати більше часу та ресурсів у порівнянні з класичними запитами. Також, що є немаловажним, робота з нечіткими моделями вимагає спеціалізованих знань в області нечіткої логіки та теорії нечітких множин.

Нечіткі типи даних офіційно не підтримуються більшістю класичних систем керування базами даних, однак, деякі системи дозволяють використовувати нечіткі дані або наближені до них концепції.

Наприклад, PostgreSQL можна використовувати для роботи з нечіткими типами даних за допомогою розширень, наприклад, pg_trgm для пошуку за схожістю (триграми). Це дозволяє здійснювати пошук нечітких рядків, використовуючи рівень подібності між текстовими полями. У ній можна створювати спеціалізовані функції та типи даних з додаванням логіки для нечітких значень за допомогою PL/pgSQL або інших підтримуваних мов.

Також, Oracle має потужний інструментарій для нечітких пошуків і обробки текстів. Зокрема, використовується розширення Oracle Text для пошуку за подібністю в текстових даних. Пошук за схожістю в Oracle можна виконувати за допомогою функції SOUNDEX для обробки текстових полів.

Можливо адаптувати до нечітких даних і вищеописані Neo4j та MongoDB. Але з самого зручного, FuzzyDB – це спеціалізована база даних, призначена саме для обробки нечітких даних. Вона дозволяє зберігати і обробляти нечіткі множини та логіку без потреби у додаткових бібліотеках чи плагінах.

Можна використовувати зовнішні бібліотеки для обробки нечітких даних і їх інтеграції з базами даних через API або збережені процедури.

3 ПІДТРИМКА МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ ЗАСОБАМИ РС

3.1 Потреби до даних для аналізу

Для аналізу даних існує кілька основних форматів, кожен з яких має свої особливості, переваги та недоліки. Найпоширеніші формати даних включають реляційні бази даних, файли CSV/Excel, NoSQL-бази даних, JSON/XML, інші текстові файли, а також формати для великих даних, наприклад Parquet або Avro.

Текстові файли, такі як CSV (comma-separated values) і TSV (tab-separated values), є популярними завдяки своїй простоті та сумісності з більшістю аналітичних інструментів. Вони підходять для зберігання простих табличних даних і швидкого обміну інформацією. Проте ці формати мають значні обмеження. Вони не підтримують ієрархічну структуру чи зв’язки між даними, що ускладнює роботу з великими та складними наборами інформації. Крім того, у великих файлах може виникати проблема з продуктивністю під час читання та запису.

JSON і XML – формати, які часто використовуються для передачі структурованих даних у веб-додатках. JSON є компактнішим і легшим для обробки, тоді як XML забезпечує більшу гнучкість, але за рахунок збільшення розміру файлу. Обидва формати добре підходять для зберігання напівструктурованих даних, але потребують додаткових ресурсів для перетворення у формат, зручний для аналізу. YAML є схожим на JSON, але орієнтований на зручність читання людьми, що робить його менш ефективним для великих обсягів інформації.

NoSQL-бази даних, такі як MongoDB, Cassandra або Redis, створені для роботи з неструктурованими або напівструктурованими даними. Вони забезпечують гнучкість у зберіганні інформації та добре підходять для сценаріїв із швидкими змінами схеми даних. Проте відсутність суворих

зв’язків і стандартних інструментів для аналітики робить їх менш придатними для традиційного аналізу даних.

Формати для великих даних, типу Parquet, Avro або ORC, оптимізовані для ефективного зберігання й обробки об’ємних даних у розподілених системах, таких як Hadoop чи Spark. Вони зберігають дані в колонному форматі, що знижує споживання пам’яті та підвищує швидкість запитів. Однак ці формати вимагають спеціалізованого програмного забезпечення та технічних знань, що обмежує їх використання у звичайних аналітичних завданнях.

У порівнянні з цими форматами, реляційні бази даних пропонують оптимальне співвідношення зручності, гнучкості та сумісності з аналітичними платформами. Вони дозволяють легко управляти зв’язками між даними, проводити складні розрахунки та інтегрувати результати в сучасні інструменти бізнес-аналітики. Інші формати мають свої сфери застосування, але для систематичного аналізу структурованих даних РБД залишаються найкращим вибором.

Реляційні бази даних є основою аналітики завдяки чіткій структурі даних. Вони організовують інформацію у вигляді таблиць з рядками й стовпцями, дозволяють встановлювати зв’язки між таблицями та забезпечують підтримку складних запитів за допомогою мови SQL. Завдяки строгим правилам цілісності, дані завжди залишаються якісними, а вбудована підтримка транзакцій забезпечує точність результатів навіть у багатокористувацьких середовищах. Реляційні бази даних масштабуються як горизонтально, так і вертикально, що робить їх ідеальними для аналітичних систем як малого, так і великого масштабу.

Для ефективного аналізу даних структура бази даних повинна відповідати кільком ключовим вимогам. Насамперед, вона має забезпечувати чітку організацію інформації, де дані легко класифікуються і розподіляються між логічними сутностями. Це дозволяє зручно працювати з великими наборами даних і підтримувати цілісність інформації. Реляційні

бази даних реалізують цю вимогу за допомогою таблиць, де кожна таблиця відповідає окремій категорії даних.

Нормалізація є важливим аспектом структури бази, оскільки вона усуває дублювання даних і мінімізує ризик виникнення помилок. У реляційних базах даних нормалізація реалізується шляхом поділу даних на окремі таблиці з встановленням чітких зв’язків між ними. Це сприяє оптимальному використанню пам’яті та спрощує оновлення даних без ризику втрати їхньої узгодженості.

Ще однією вимогою є підтримка складних зв’язків між даними, таких як відносини «один до одного», «один до багатьох» чи «багато до багатьох». Реляційні бази даних забезпечують це за допомогою первинних і зовнішніх ключів, які дозволяють зв’язувати таблиці й виконувати складні запити. Це робить їх ідеальними для аналітики, де часто необхідно об'єднувати й аналізувати інформацію з різних джерел.

Важливою також є підтримка стандартизованих запитів і сумісність з аналітичними інструментами. Реляційні бази даних використовують мову SQL, яка є універсальним стандартом для роботи з даними. Завдяки цьому аналітики можуть швидко отримувати необхідну інформацію, створювати звіти та інтегрувати базу даних з аналітичними платформами.

Реляційні бази даних відповідають усім цим вимогам, забезпечуючи структурованість, зв’язність і ефективність роботи з інформацією. Їхній підхід до організації даних сприяє як оперативному доступу до інформації, так і можливості масштабування для роботи з великими обсягами даних. Це робить їх оптимальним вибором для завдань аналізу, особливо у складних і динамічних середовищах.

3.2 Підтримка методів засобами PC

Застосування агрегатних функцій до ключових атрибутів дає змогу будувати логічні залежності між інформаційними одиницями.

Запроваджено поняття функціональних асоціативних правил. Семантична мережа (СМ), створена на основі запропонованого підходу, сприяє підвищенню ефективності систем підтримки прийняття рішень.

У процесі перенесення знань у пам’ять необхідно включати інформацію про структуру інформаційних одиниць. Наприклад, машинне слово може бути представлено як структура, що містить інформацію про те, в яких розрядах зберігаються дані про групи та спеціальності наукових працівників. При цьому слід використовувати спеціальні словники, такі як «довідник груп» і «довідник спеціальностей», де зазначені всі наявні в пам’яті інформаційної системи групи та спеціальності. Ці атрибути слугують іменами для машинних слів, що відповідають рядкам таблиці, і дозволяють здійснювати пошук необхідної інформації. Кожен рядок таблиці є окремим екземпляром такої структури.

Сучасні технології обробки інформації, де інформаційні одиниці поділяються на дані та команди, створили ситуацію, у якій дані залишаються пасивними, тоді як команди є активними. Усі процеси запускаються командами, які звертаються до даних лише за необхідності.

Метою проведених досліджень є аналіз особливостей інформаційних одиниць і структур даних, що впливають на технологію вилучення знань. Бази даних (БД) можна розглядати як структури баз знань (БЗ), адже на сьогодні немає таких баз знань, які повністю забезпечують внутрішню інтерпретованість, структурованість, зв’язність, наявність семантичної міри та активність знань.

Для забезпечення однозначної інтерпретації інформаційні одиниці повинні мати гнучку структуру. Вони можуть бути частиною інших одиниць або містити інші складові, утворюючи взаємозв’язки типу «частина – ціле», «елемент – клас» тощо. У базах даних зв’язки між інформаційними одиницями відображають їхні відносини, які можуть бути декларативними чи процедурними.

Наприклад, декларативні знання описуються відношеннями на кшталт «у певний момент часу», «причина – наслідок» або «еквівалентність». Процедурні знання виникають у відношеннях типу «аргумент – функція», які визначають операції, що виконуються для обчислення певних функцій.

Агрегатні функції – це функції, які підраховують кількість записів у таблиці, визначають кількість значень у стовпці, знаходять мінімальне або максимальне значення, обчислюють суму чи середнє значення. До них належать функції COUNT, SUM, MAX, MIN, AVG тощо, залежно від функціональності, запропонованої розробником системи.

Для застосування агрегатних функцій до групи значень використовується параметр угруповання GROUP BY. Цей параметр об’єднує однакові значення заданого атрибута в один рядок, що містить підсумкові результати.

Останні роки спостерігається активний розвиток і гібридизація методів інтелектуальної обробки інформації. Концепція м’яких обчислень об'єднує такі напрями, як нечітка логіка, штучні нейронні мережі, видобування знань, бази даних, імовірнісні підходи, еволюційні алгоритми та інші. Ці підходи доповнюють один одного, що дозволяє створювати гібридні інтелектуальні системи, використовуючи різноманітні їх комбінації.

Дослідники баз даних також активно працюють у цьому напрямку. Зокрема, розробляється нечітка реляційна алгебра та спеціалізовані розширення структурованої мови запитів (SQL) для роботи з нечіткими даними. Таким чином, формуються сучасні перспективи у сфері обробки інформації, зокрема, у вигляді нечітких запитів до баз даних (fuzzy queries).

Актуальними питаннями в цьому контексті є дві ключові проблеми: як проектувати системи з нечіткими даними та в яких структурах їх найкраще зберігати. Рішення цих завдань відкриває можливості для інтеграції реляційних баз даних, які містять величезні обсяги інформації, із системами, що базуються на нечіткій логіці.

Розглянемо задачу в загальному вигляді. Нехай $U(R_1,\ldots, R_n)$ – БД, що зберігає основні дані, $R^f(A_1, A_2, A_3)$ – відношення фаззіфікації. Задача має сенс, якщо в БД U існує параметр, для якого виконана фаззіфікація.

Щоб організувати спільну роботу з базами даних U і R, формалізуємо процедуру інтеграції, спираючись на поетапну нормалізацію. Структура БД U отримана на підставі функціональних залежностей $F = \{M_i \to N_i\}$, де M_i, $N_i \in U$. Виділимо одну залежність, яка містить атрибут з параметрами фаззіфікації, і позначимо її як $W \to V$, причому W і V можуть бути множинами. Відношення R^f містить одну залежність виду $F' = \{A_1, A_2, A_3 \to A_1, A_2, A_3\}$. Спираючись на аксіоми виводу, можна одержати еквівалентну множину

$$F' = \{A_1, A_2, A_3 \to A_1;\ A_1, A_2, A_3 \to A_2;\ A_1, A_2, A_3 \to A_3\}.$$

Нехай параметр фаззіфікації відповідає атрибуту A_2, тоді для визначення типу зв'язку необхідно одержати множину $F = F \cup F'$ і розглянути два випадки, що впливають на правила нормалізації.

Перший – $A_2 \in W$ – пошук неповних залежностей: якщо виконуються функціональні залежності $\xi \to \zeta$ і $\omega \to \zeta$, причому $\omega \subseteq \xi$, тоді залежність $\omega \to \zeta$ є неповною.

Другий – $A_2 \in V$ – пошук транзитивно залежних елементів: якщо виконуються функціональні залежності $\xi \to \omega$ і $\omega \to \zeta$, тоді елемент ζ є транзитивно залежним.

Існування таких залежностей дозволить виконати коректну декомпозицію й встановити зв'язок між базами даних U і R^f.

Якщо $A_2 = W$ або $A_2 = V$, то процес декомпозиції призводить до другої або третьої нормальної форми. Якщо рівняння не виконуються, то неможливо організувати підтримку однозначності зв'язаних даних, тому що асоціація між відношеннями буде відповідати типу «багато-до-багатьох».

Як правило, на практиці умови рівняння не виконуються і для нормалізації необхідно виділити базис F і повторити процедуру

декомпозиції. Враховуючи той факт, що структура БД не повинна змінюватися, необхідно зв'язати відношення фаззіфікації R^f і БД U без реструктуризації схеми даних.

Для усунення зв'язку «N:N» запровадимо додаткову сполучну сутність, яка вирішить проблему цілісності даних за рахунок визначення нових типів зв'язків. «Сутність-зв'язок» буде містити один атрибут – сполучний для R^f і U, причому з об'єктивних причин він буде ключовим.

Виходячи з опису концептуальної схеми ПО можна бачити, що для коректного з'єднання R^f і U необхідно побудувати проміжну таблицю. Такий підхід гарантує погодженість даних для будь-яких параметрів фаззіфікації.

Для даної задачі цілком коректні результати при виконанні з'єднання відношень з асоціацією типу «N:N». Можливі значення атрибута $A_1 \in U$ можуть повторюватися стільки разів, скільки це значення перетинає границі діаграми фаззіфікації по осі ординат. Тобто кожному значенню атрибута A_1 відповідає рядок унікальних даних. Якщо A_1 не є ключем, і значення повторюються, то, за визначенням множини, у рядку повинно бути хоча б одне відмінне значення. У термінах розв'язуваної задачі необхідно аналізувати всі такі рядки. В атрибуті A_1 відношення R^f також повторюються значення, які необхідно проаналізувати, причому в різних комбінаціях.

Таким чином, у загальному вигляді для аналізу даних, що накопичуються в реляційних базах даних, досить побудувати відношення фаззіфікації й встановити зв'язок з атрибутом (атрибутами), за значеннями якого необхідно провести відповідний аналіз.

3.3 Застосування у фармацевтичній галузі

На прикладі усе це можна побачити у програмі у застосунку Microsoft Access, зробленій для демонстрації. Програма показує можливість

використання асоціативної логіки в фармацевтичній галузі, а саме – рекомендуючи, на основі часто обираних разом ліків, схожі товари для покупця.

Дані записані у вигляді реляційної бази даних, чию перевагу для використання було описано у розділах вище, що можна побачити на рисунку 3.1.

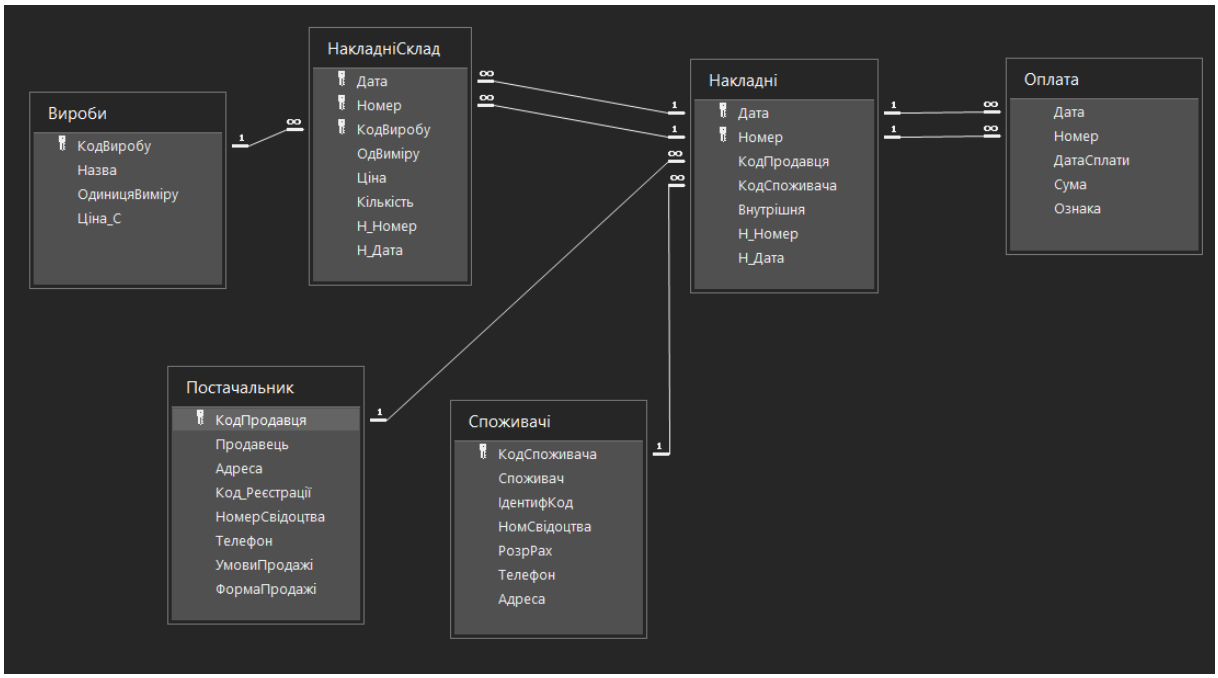

Рисунок 3.1 – Таблиці БД

Для даного прикладу існує 6 таблиць.

Таблиця Вироби складається з полей:

- КодВиробу (що є ключем),
- Назва,
- ОдиницяВиміру,
- Ціна_С.

Таблиця НакладніСклад складається з полей:

- Дата (що є ключем),
- Номер (що є ключем),

- КодВиробу (що є ключем),
- ОдВиміру,
- Ціна,
- Кількість,
- Н_Номер,
- Н_Дата.

Таблиця Постачальник складається з полей:

- КодПродавця (що є ключем),
- Продавець,
- Адреса,
- Код_Реєстрації,
- НомерСвідоцтва,
- Телефон,
- УмовиПродажі,
- ФормаПродажі.

Таблиця Споживачі складається з полей:

- КодСпоживача (що є ключем),
- Споживач,
- ІдентифКод,
- НомСвідоцтва,
- РозрРах,
- Телефон,
- Адреса.

Таблиця Накладні складається з полей:

- Дата (що є ключем),
- Номер (що є ключем),
- КодПродавця,
- КодСпоживача,
- Внутрішня,
- Н_Номер,

– Н_Дата.

Таблиця Оплата складається з полей:

– Дата (що є ключем),

– Номер (що є ключем),

– ДатаСплати,

– Сума,

– Ознака.

Між собою таблиці пов’язані зв’язками «один-до-багатьох».

Дані про покупки-продажі та накладні можна подивитися за допомогою форм та звітів, наприклад, на рисунку 3.2 можна побачити усі податкові.

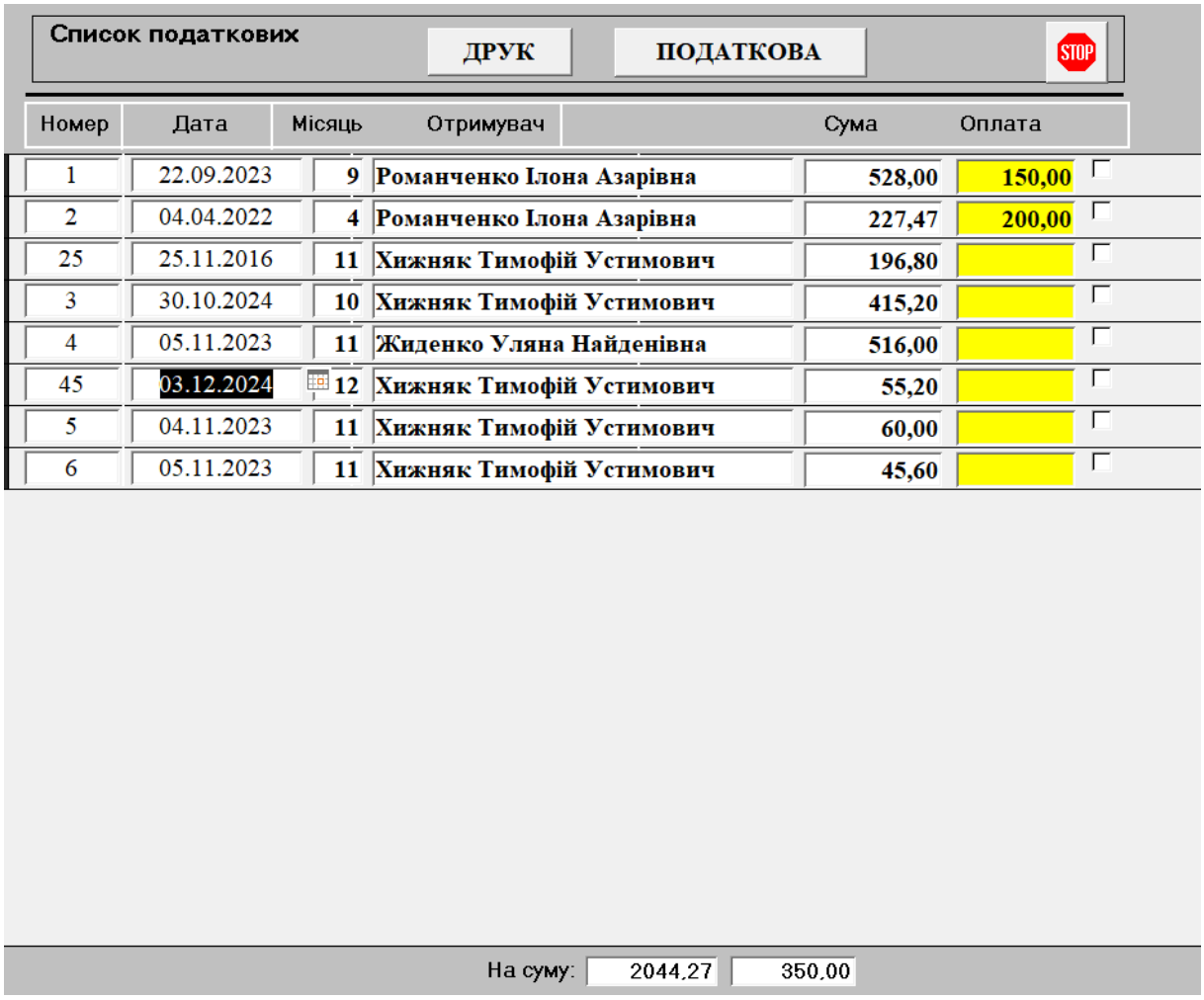

Рисунок 3.2 – Список податкових

Таким же чином, на рисунку 3.3, можна побачити і усі накладні, на основі яких можна далі провести аналіз товарів, які часто покупають разом.

Список накладних

Отримувач: Жиденко Уляна Найденівна

накладна № 4 **Дата** ####### ☐ — **430,00** **86,00** **516,00**

Назва	од.вим.	ціна	к-сть	сума	НДС	сума з НДС
Глід	л	80.00	2	160.00	32.00	192.00
Демідрол	уп	54.00	5	270.00	54.00	324.00

Отримувач: Романченко Ілона Азарівна

накладна № 2 **Дата** ####### ☐ — **189,56** **37,91** **227,47**

Назва	од.вим.	ціна	к-сть	сума	НДС	сума з НДС
Валідол	уп	3,00	3	9,00	1,80	10,80
Настойка валеріани	б.	2,34	4	9.36	1,87	11.23
Корвалол	таб.	4,00	2	8.00	1,60	9,60
Валокордін	уп.	2,64	55	145,20	29,04	174,24
Бінт	уп	3,60	5	18,00	3,60	21,60

накладна № 1 **Дата** ####### ☐ — **440,00** **88,00** **528,00**

Назва	од.вим.	ціна	к-сть	сума	НДС	сума з НДС
Валідол	уп	7,00	5	35,00	7,00	42,00
Настойка валеріани	б.	3.00	10	30.00	6.00	36.00
Корвалол	таб.	5.00	10	50.00	10,00	60.00
Зеленка	бан	5,00	5	25,00	5,00	30,00
Демідрол	уп	50,00	6	300,00	60,00	360,00

Отримувач: Хижняк Тимофій Устимович

накладна № 6 **Дата** ####### ☐ — **38,00** **7,60** **45,60**

Назва	од.вим.	ціна	к-сть	сума	НДС	сума з НДС
Зеленка	бан	5,00	4	20,00	4,00	24,00
Бінт	уп	3,00	6	18,00	3,60	21,60

накладна № 5 **Дата** ####### ☐ — **50,00** **10,00** **60,00**

Назва	од.вим.	ціна	к-сть	сума	НДС	сума з НДС
Зеленка	бан	5,00	6	30,00	6,00	36,00
Бінт	уп	4,00	5	20,00	4,00	24,00

накладна № 45 **Дата** ####### ☐ — **46,00** **9,20** **55,20**

Назва	од.вим.	ціна	к-сть	сума	НДС	сума з НДС

Рисунок 3.3 – Усі податкові

Аналіз же проходить за допомогою запиту у базу даних, який можна побачити на рисунку 3.4.

Валідол	5	1	1	1	1		1		
Настойка валеріани	3	1	1	1					
Корвалол	4	1	1	1			1		
Валокордін	1		1						
Зеленка	3	1						1	1
Бінт	3		1					1	1
Глід	1					1			
Демідрол	3	1			1	1			

Рисунок 3.4 – Перший пункт аналізу за товарами

Цей пункт виділяє, незважаючи на саму кількість куплених товарів, які товари обрані разом. Наприклад, можна побачити, що у двох випадках брали разом зеленку та бінт, а у чотирьох випадках – валідол та корвалол. Робиться це за допомогою запиту у базу даних, а саме:

TRANSFORM Count(НакладніСклад.Дата) AS [Count-Дата]

SELECT НакладніСклад.КодВиробу, Count(НакладніСклад.Дата) AS [Підсумкове значення Дата]

FROM НакладніСклад

GROUP BY НакладніСклад. КодВиробу

PIVOT НакладніСклад.Номер;

Далі, на наступному пункту (рисунок 3.5), робиться висновок, у яких накладних є якісь пари товарів.

Номер	Count-КодИ
1	2
2	2
25	2
3	1
45	2

Рисунок 3.5 – Другий пункт аналізу за товарами

Робиться це за допомогою запиту у базу даних, а саме:

SELECT [3_АССОЦІАЦІЯ В НАКЛАДНИХ].Номер, Count([3_ АССОЦІАЦІЯ В НАКЛАДНИХ]. КодВиробу) AS [Count-КодВиробу]

FROM [3_ АССОЦІАЦІЯ В НАКЛАДНИХ]

GROUP BY [3_ АССОЦІАЦІЯ В НАКЛАДНИХ].Номер;

Далі, на наступному пункту (рисунок 3.6), робиться висновок, про то, у якій кількості, і на яку суму, бралися парні товари у оброблених накладних.

45	Корвалол	Корвалол	7	4
45	Валідол	Валідол	6	3
3	Валідол	Валідол	5	2
25	Корвалол	Корвалол	33	4
25	Валідол	Валідол	4	3
2	Корвалол	Корвалол	2	4
2	Валідол	Валідол	3	3
1	Корвалол	Корвалол	10	5
1	Валідол	Валідол	5	7
*				

Рисунок 3.6 – Третій пункт аналізу за товарами

Робиться це за допомогою запиту у базу даних, а саме:

SELECT НакладніСклад.Номер, НакладніСклад.КодВиробу, НакладніСклад. КодВиробу, НакладніСклад.Кількість, НакладніСклад.Ціна

FROM НакладніСклад

WHERE (((НакладніСклад. КодВиробу)=45 Or (НакладніСклад. КодВиробу)=43))

ORDER BY НакладніСклад.Номер DESC;

Таким чином, проводиться асоціація між, у даному випадку, покупками Корвалола та Валідола, та дає інформацію, що користувачу, який замовляє один з цих товарів, є сенс рекомендувати другий як можливу покупку.

ВИСНОВКИ

Була описана актуальність інтеграції методів інтелектуального аналізу даних із реляційними системами управління базами даних. У сучасних умовах, коли інформаційні системи характеризуються динамічним розвитком, високою конкуренцією та складністю умов експлуатації, ефективне управління великими обсягами даних потребує адаптації методів аналізу до специфіки РСУБД.

Методи інтелектуального аналізу дозволяють обробляти великі обсяги інформації, зокрема нечіткі та неповні дані, що є поширеними у багатьох сферах діяльності. Інтеграція нечітких моделей у реляційні бази даних сприяє підвищенню точності аналізу, дозволяючи моделювати невизначеність та розмитість інформації. Це особливо актуально для завдань, пов’язаних із прогнозуванням поведінки споживачів, аналізом їхніх уподобань, фінансових можливостей та очікувань, які можуть бути представлені у вигляді нечітких множин.

Розгляд структурованих моделей даних допоміг краще зрозуміти підходи до зберігання та обробки нечітких оцінок і запитів у базах даних, що забезпечує гнучкість і адаптивність у прийнятті рішень. У результаті підтримка методів інтелектуального аналізу в РСУБД сприяє створенню адаптивних і масштабованих рішень для роботи зі складними та неоднозначними даними, що має значний потенціал для підвищення ефективності сучасних інформаційних систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Logunova I., Bolgurtseva O. K-Means Clustering Algorithm in ML. *K-Means Clustering Algorithm in ML*. URL: https://serokell.io/blog/k-means-clustering-in-machine-learning (дата звернення: 12.12.2024).

2. DBSCAN Clustering in ML | Density based clustering – GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/ (дата звернення: 12.12.2024).

3. Classification Algorithm. *Free Chatbot maker | Chatbot for Website, WhatsApp | BotPenguin*. URL: https://botpenguin.com/glossary/classification-algorithm (дата звернення: 12.12.2024).

4. Ng A. Association Rules and the Apriori Algorithm: A Tutorial. *kdnuggets*. URL: https://www.kdnuggets.com/2016/04/association-rules-apriori-algorithm-tutorial.html (дата звернення: 12.12.2024).

5. FP Growth Algorithm in Data Mining – Javatpoint. *www.javatpoint.com*. URL: https://www.javatpoint.com/fp-growth-algorithm-in-data-mining (дата звернення: 12.12.2024).

6. Biscobing J. What is OLAP (online analytical processing)? | Definition from TechTarget. *Search Data Management*. URL: https://www.techtarget.com/searchdatamanagement/definition/OLAP (дата звернення: 12.12.2024).

7. Data Warehouse | Types of OLAP – javatpoint. *www.javatpoint.com*. URL: https://www.javatpoint.com/data-warehouse-types-of-olap (дата звернення: 12.12.2024).

8. What is a Database Model. *Lucidchart*. URL: https://www.lucidchart.com/pages/database-diagram/database-models (дата звернення: 11.11.2024).

9. Jafarzadeh M. Schema Design and Relationship in NoSQL Document-based databases. *Wissen im Service | USU Blog*. URL:

https://blog.usu.com/en-us/schema-design-and-relationship-in-nosql-document-based-databases (дата звернення: 11.11.2024).

10. W3Schools.com. *W3Schools Online Web Tutorials*. URL: https://www.w3schools.com/mongodb/ (дата звернення: 11.11.2024).

11. What is Amazon DocumentDB (with MongoDB compatibility) – Amazon DocumentDB. *docs.aws.amazon*. URL: https://docs.aws.amazon.com/documentdb/latest/developerguide/what-is.html (дата звернення: 11.11.2024).

12. 1. Introduction – Apache CouchDB® 3.4 Documentation. *Overview – Apache CouchDB® 3.4 Documentation.* URL: https://docs.couchdb.org/en/stable/intro/index.html (дата звернення: 11.11.2024).

13. Couchbase Documentation | Couchbase Docs. *Couchbase Documentation | Couchbase Docs.* URL: https://docs.couchbase.com/home/index.html (дата звернення: 11.11.2024).

14. What is a Key-Value Database? – Redis. *Redis*. URL: https://redis.io/nosql/key-value-databases/ (дата звернення: 11.11.2024).

15. Memcached Overview. *Memcached Documentation*. URL: https://docs.memcached.org/ (дата звернення: 11.11.2024).

16. RocksDB Basics | Speedb Documentation. *About | Speedb Documentation.* URL: https://docs.speedb.io/rocksdb-basics (дата звернення: 11.11.2024).

17. Amazon DynamoDB Documentation. *docs.aws.amazon*. URL: https://docs.aws.amazon.com/dynamodb/ (дата звернення: 11.11.2024).

18. Czerepicki A. Application of graph databases for transport purposes. *Bulletin of the Polish Academy of Sciences Technical Sciences.* 2016. Т. 64, № 3. С. 457–466. URL: https://doi.org/10.1515/bpasts-2016-0051 (дата звернення: 11.11.2024).

19. Welcome to Neo4j.rbâ€ ™s documentation! – Neo4j.rb 10.0.1 documentation. *Welcome to Neo4j.rbâ€ ™s documentation! – Neo4j.rb 10.0.1*

documentation. URL: https://neo4jrb.readthedocs.io/en/stable/ (дата звернення: 11.11.2024).

20. Amazon Neptune Documentation. *docs.aws.amazon*. URL: https://docs.aws.amazon.com/neptune/ (дата звернення: 11.11.2024).

21. What is ArangoDB?. *ArangoDB Documentation*. URL: https://docs.arangodb.com/3.10/about-arangodb/ (дата звернення: 11.11.2024).

22. Basic Usage – JanusGraph. *JanusGraph*. URL: https://docs.janusgraph.org/getting-started/basic-usage/ (дата звернення: 11.11.2024).

23. Methods of intellectual analysis of processes in medical information systems / V. O. Filatov та ін. *Information extraction and processing*. 2020. Т. 2020, № 48. С. 92–98. URL: https://doi.org/10.15407/vidbir2020.48.092 (дата звернення: 26.12.2024).

24. Avrunin O., Vlasov O., Filatov V. MODEL OF SEMANTIC INTEGRATION OF INFORMATION SYSTEMS PROPERTIES IN RELAY DATABASE REENGINEERING PROBLEMS. *Innovative Technologies and Scientific Solutions for Industries*. 2020. № 4 (14). С. 5–12. URL: https://doi.org/10.30837/itssi.2020.14.005 (дата звернення: 26.12.2024).

25. Filatov V., Semenets V., Zolotukhin O. DATA MINING IN RELATIONAL SYSTEMS. *Innovative Technologies and Scientific Solutions for Industries*. 2020. № 3 (13). С. 65–76. URL: https://doi.org/10.30837/itssi.2020.13.065 (дата звернення: 26.12.2024).

26. Neo-Fuzzy Radial-Basis Function Neural Network and Its Combined Learning / Y. Bodyanskiy та ін. *Studies in Computational Intelligence*. 2023. С. 323–340. URL: https://doi.org/10.1007/978-3-031-37450-0_19 (дата звернення: 26.12.2024).

27. Filatov V. O., Yerokhin M. A. IMPROVED MULTI-OBJECTIVE OPTIMIZATION IN BUSINESS PROCESS MANAGEMENT USING R-NSGA-II. *Radio Electronics, Computer Science, Control*. 2023. № 3. С. 187.

URL: https://doi.org/10.15588/1607-3274-2023-3-18 (дата звернення: 26.12.2024).