БАКАЛАВРСЬКА РОБОТА

БР.ІП – 51.00.00.000 ПЗ

Група ІП-22-2

Федорчук Станіслав

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Федорчук Станіслав Олегович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Методи моделювання даних у програмних системах
(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____С.О. Федорчук_____
(підпис, ініціали та прізвище здобувача)

Науковий керівник ____Михайлюк Ірина Романівна, к.п.н., доцент_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Федорчук Станіславу Олеговичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Методи моделювання даних у програмних системах"

керівник проекту (роботи) Михайлюк Ірина Романівна, к.п.н., доцент

затверджені наказом вищого навчального закладу від “ 21_ ” _квітня 2026 р. № __179/7____

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз вимог до програмного забезпечення
2. Аналіз вимог і постановка задачі
3. Проектування системи
4. Реалізація проекту
5. Тестування та впровадження

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1 Ієрархічна схема бази даних (рис. 1.1, ст. 14)

2 Приклад кубічної моделі для системи управління навчальним процесом (рис. .2.3, ст. 40)

3 Приклад застосування алгоритму MapReduce в системі управління навчальним процесом. (рис. 2.8, ст. 50)

4 Внутрішня архітектура програмного забезпечення SDAM (рис. 3.4, ст. 57)

5 Розбиття області за допомогою квадродерева (рис. 3.10, ст. 64)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання 2026 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____ _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	17.01.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.02.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.03.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.04.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	02.05.2026	виконано

Студент _____ Федорчук С.О
(підпис).

Керівник роботи _____Михайлюк І.Р.
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 75 сторінок, 26 рисунків, 1 таблиця, список використаних джерел із 40 найменувань.

Метою роботи є дослідження методів моделювання даних у програмних системах та аналіз підходів до побудови ефективних, масштабованих і узгоджених моделей даних для сучасних інформаційних систем.

Об'єкт дослідження: процеси моделювання даних у програмних системах.

Предмет дослідження: методи, моделі та інструменти моделювання даних, зокрема реляційні, об'єктно-орієнтовані та сучасні підходи до обробки великих даних.

Результати дослідження: проведено аналіз підходів до моделювання даних, досліджено реляційну модель даних, а також сучасні технології роботи з великими даними та їх вплив на структуру програмних систем.

У першому розділі розглянуто теоретичні основи моделювання даних, визначено ключові поняття, терміни та особливості реляційної моделі даних.

У другому розділі досліджено методи та інструменти моделювання даних, включаючи підходи до аналізу даних і використання сучасних технологій обробки великих даних.

У третьому розділі розглянуто реалізацію моделі даних у програмній системі, особливості інтеграції з архітектурою програмного забезпечення, а також виконано оцінку ефективності та виявлено основні проблеми моделювання даних.

Висновок: використання сучасних методів моделювання даних дозволяє підвищити ефективність обробки інформації, а також покращити масштабованість і продуктивність програмних систем.

КЛЮЧОВІ СЛОВА: МОДЕЛЮВАННЯ ДАНИХ, РЕЛЯЦІЙНА МОДЕЛЬ, ER-ДІАГРАМИ, UML, БАЗИ ДАНИХ, BIG DATA, ПРОЄКТУВАННЯ СИСТЕМ.

ANNOTATION

The bachelor's thesis contains 75 pages, 26 figures, 1 tables, and a reference list containing 40 sources.

The aim of the work is to study data modeling methods in software systems and to analyze approaches for building efficient, scalable, and consistent data models for modern information systems.

Research object: data modeling processes in software systems.

Research subject: methods, models, and tools for data modeling, including relational, object-oriented, and modern approaches to big data processing.

Research results: the main approaches to data modeling were analyzed, the relational data model was studied, design methods such as ER diagrams and UML were reviewed, and modern big data technologies and their impact on software system structure were examined.

The first section describes the theoretical foundations of data modeling, defines key concepts and terms, and analyzes the features of the relational data model.

The second section examines methods and tools for data modeling, including approaches to data analysis and modern big data technologies.

The third section covers the implementation of a data model within a software system, integration with software architecture, and evaluation of efficiency, as well as identification of key data modeling challenges.

Conclusion: the use of modern data modeling methods improves data consistency, enhances processing efficiency, and ensures scalability and performance of software systems.

KEY WORDS: DATA MODELING, RELATIONAL MODEL, ER DIAGRAMS, UML, DATABASES, BIG DATA, SYSTEM DESIGN.

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій дані стають ключовим ресурсом для функціонування програмних систем. Обсяги інформації постійно зростають, а вимоги до її обробки, зберігання та аналізу ускладнюються. Це зумовлює необхідність використання ефективних методів моделювання даних, які дозволяють забезпечити структурованість, цілісність і масштабованість програмних рішень. Від якості побудови моделей даних значною мірою залежить продуктивність системи, зручність її підтримки та можливість подальшого розвитку.

Актуальність теми зумовлена необхідністю розробки ефективних підходів до моделювання даних у програмних системах, що дозволяють оптимізувати процеси зберігання, обробки та передачі інформації. Існуючі підходи, зокрема реляційні, об’єктно-орієнтовані та NoSQL-моделі, мають свої переваги та обмеження, які необхідно враховувати при проектуванні сучасних інформаційних систем. У зв’язку з поширенням великих даних (Big Data) та розподілених систем особливого значення набуває вибір адекватної моделі даних і технологій її реалізації.

Метою роботи є дослідження методів моделювання даних у програмних системах, аналіз їх особливостей та розробка підходів до ефективного використання моделей даних у процесі створення програмного забезпечення.

Завданнями дослідження є аналіз предметної області та існуючих підходів до моделювання даних, визначення основних понять і термінів у сфері баз даних, дослідження реляційної моделі даних, вивчення сучасних методів моделювання та аналізу даних, розгляд технологій обробки великих даних, проектування моделі даних для програмної системи, аналіз архітектурних рішень та оцінка ефективності реалізованих підходів, а також виявлення основних проблем і викликів у процесі моделювання даних.

Об’єктом дослідження є процеси моделювання даних у програмних системах, що включають проектування, реалізацію та оптимізацію структур даних для забезпечення ефективної роботи програмного забезпечення.

Предметом дослідження є методи, моделі та інструменти моделювання даних, зокрема реляційні моделі, підходи до аналізу даних, технології обробки великих даних, а також архітектурні рішення, що забезпечують ефективну інтеграцію даних у програмних системах.

Методи дослідження включають аналіз наукових джерел для вивчення теоретичних основ моделювання даних, порівняльний аналіз різних моделей і технологій, моделювання структур даних, експериментальні методи для оцінки ефективності розроблених рішень, а також узагальнення отриманих результатів.

У роботі передбачається розробка моделі даних для програмної системи з урахуванням сучасних вимог до продуктивності, масштабованості та надійності. Особлива увага приділяється вибору архітектури програмного забезпечення, що забезпечує ефективну взаємодію компонентів системи та оптимальне використання ресурсів. Очікується, що результати дослідження дозволять підвищити ефективність обробки даних, спростити процес розробки та підтримки програмних систем, а також забезпечити їх адаптивність до змінних умов.

Бакалаврська робота містить 75 сторінок, 26 рисунків, 1 таблиця, 3 розділи, список використаних джерел із 40 найменувань.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ МОДЕЛЮВАННЯ ДАНИХ

1.1 Основні поняття та сутність моделювання даних

Збір, інтеграція та зберігання великих обсягів інформації швидко стає необхідністю серед інженерів-програмістів у промисловості, а також для науковців у дослідженнях.

В останні роки сучасна спільнота data mining розробила безліч алгоритмів та методів, що використовуються для різних завдань пошуку знань у великих базах даних. Однак лише деякі з них є загальнодоступними, і дослідник, який бажає порівняти новий алгоритм з існуючими алгоритмами або проаналізувати реальні дані, вважає це завдання складним. Крім того, оскільки алгоритми стають складнішими, а також пропонуються гібридні алгоритми, що поєднують кілька підходів, завдання впровадження таких алгоритмів з нуля стає дедалі трудомісткішим.

Ми спроектували та розробили цю інфраструктуру ітеративним способом, спочатку впровадивши її за допомогою стандартної трирівневої веб-архітектури [1]. Потім ми розширили систему, щоб відкрити рівні сервісів, які можуть використовувати інші дослідницькі групи, а також наша власна команда, щоб ізолювати складнощі взаємодії з нашою моделлю даних та дозволити повторне використання наших інструментів збору даних. Ми також відкрили рівень веб-сервісів, щоб забезпечити географічно розподіленим мобільним клієнтам кращий доступ до наших сервісів та даних. Хоча це сучасні практики розробки програмного забезпечення, зрештою ми зіткнулися з проблемою масштабування нашого рівня збереження даних через величезні обсяги даних, що генеруються навіть однією масовою надзвичайною подією [1], що призвело до необхідності використання додаткових методів та технологій.

Тут ми повідомляємо про поточний стан та майбутній напрямок інфраструктури збору даних, яку ми розробляємо для підтримки досліджень у кризовій інформатиці. Програмна інженерія почала відігравати ще важливішу

роль у цій галузі, ніж ми могли спочатку уявити. Дані, зібрані та збережені нашою системою, є життєво важливими для досліджень наших колег у таких галузях, як обробка природної мови (NLP), системи та безпека, інтернет-політика та людиноцентричні обчислення (HCC) [2]. Щоб ці групи могли витягувати репрезентативні вибірки та робити статистично значущі дослідницькі заяви, вони майже повністю покладаються на точний та своєчасний збір даних соціальних мереж нашою системою.

Ця залежність ставить обмеження масштабованості та стійкості на перший план у проектуванні системи, часто отримуючи пріоритет над простими запитами на функції, що стосуються наших інструментів аналізу. Ці обмеження створюють великий тиск на нашу дослідницьку команду з програмної інженерії, оскільки робота, яку вони виконують з проектування та розробки інфраструктури, часто приховується від решти дослідницької групи. Наявність точних та повних наборів даних часто вважається само собою зрозумілою колегами-дослідниками, які звикли працювати з загальнодоступними наборами даних, такими як «Коричневий корпус» [6]. Наша мета - задовольнити це очікування та забезпечити їм доступ майже в режимі реального часу до наборів даних аналогічної якості. Розробляючи систему для підтримки цих високих потреб, ми маємо завдання забезпечити систему, яка є відмовостійкою в низці областей, що стосуються - високорозподілених та масштабованих систем - характеристик, які зазвичай не потрібні під час розробки прототипів програмного забезпечення для досліджень у галузі програмної інженерії.

Також відомо, що не існує універсально найкращого алгоритму інтелектуального аналізу даних для всіх областей застосування [3]. Для підвищення стійкості систем інтелектуального аналізу даних можна використовувати інтегровану архітектуру інтелектуального аналізу даних для застосування різних видів алгоритмів та/або гібридних методів до заданого набору даних. Найпоширенішими є архітектури інструментальних наборів, де кілька алгоритмів зібрані в пакет, з якого якимось чином вибирається найбільш підходящий алгоритм для цільової проблеми. Прикладом є Бібліотека машинного

навчання (MLC++) [1], яка розроблена для надання дослідникам широкого спектру інструментів, що можуть прискорити розробку алгоритмів, підвищити надійність програмного забезпечення, забезпечити порівняння та візуально відобразити інформацію. Це досягається за допомогою бібліотеки класів та функцій C++, що реалізують найпоширеніші алгоритми. Крім того, масив інструментів, що надаються MLC++, дає користувачеві гарну відправну основу для впровадження нових алгоритмів. Інший приклад, Clementine [2], - це інтегрований інструмент, який реалізує алгоритми інтелектуального аналізу даних двох парадигм виявлення знань, а саме індукції правил та нейронних мереж. Він розроблений, щоб дозволити неспеціалістам витягувати цінну інформацію зі своїх історичних даних. Що стосується індукції правил, Clementine включає два алгоритми побудови дерева рішень. Один з них базується на ID3 [3], розширеному для прогнозування неперервних цільових атрибутів, завдяки чому алгоритм може виконувати як регресійні, так і класифікаційні завдання. Інший – це добре відомий алгоритм C4.5 [4]. Що стосується нейронних мереж, Clementine включає алгоритм зворотного поширення [5], розширений методом «обрізки», головним чином для завдань класифікації, та мережу Кохонена [6] для завдань кластеризації.

Досягнення в просторових базах даних дозволили збирати величезні обсяги даних у різних ГІС-додатках, починаючи від систем дистанційного зондування та супутникової телеметрії до комп'ютерної картографії та екологічного планування [4]. Підгалузь інтелектуального аналізу даних, яка займається вилученням неявних знань та просторових зв'язків, що явно не зберігаються в просторових базах даних, називається просторовим інтелектуальним аналізом даних. Однак, схоже, що наразі немає ГІС-систем зі значною функціональністю просторового інтелектуального аналізу даних. Було розроблено певне програмне забезпечення для просторового інтелектуального аналізу даних, але більшість систем в основному базуються на незначних модифікаціях попередніх непросторових систем інтелектуального аналізу даних. Система GeoMiner [7] є просторовим розширенням реляційної системи інтелектуального аналізу даних DBMiner [8], яка була розроблена для

інтерактивного інтелектуального аналізу багаторівневих знань у великих реляційних базах даних та сховищах даних. Система DBMiner реалізує широкий спектр функцій інтелектуального аналізу даних, включаючи характеристику, порівняння, асоціацію, класифікацію, прогнозування та кластеризацію. Завдяки включенню кількох цікавих методів інтелектуального аналізу даних, включаючи OLAP (онлайнову аналітичну обробку) та атрибутно-орієнтовану індукцію, система забезпечує зручне для користувача інтерактивне середовище інтелектуального аналізу даних з хорошою продуктивністю [5]. GeoMiner використовує архітектуру SAND (просторові та непросторові дані) для моделювання просторових баз даних та включає модуль побудови куба просторових даних, модуль просторової онлайн-аналітичної обробки (OLAP) та модулі просторового аналізу даних. Ще одним кроком у розробці програмного забезпечення для просторового аналізу даних є інтерфейс S-PLUS для ArcView GIS [9]. Цей програмний пакет надає інструменти для аналізу певних класів просторових даних (наприклад, геостатистичних даних, даних решітки, просторових точкових шаблонів). Однак, оскільки S-PLUS є інтерпретованою мовою програмування, функції, написані в цьому пакеті, здаються набагато повільнішими, ніж їхні еквіваленти, реалізовані на C та C++, що обмежує практичне застосування досить невеликими базами даних.

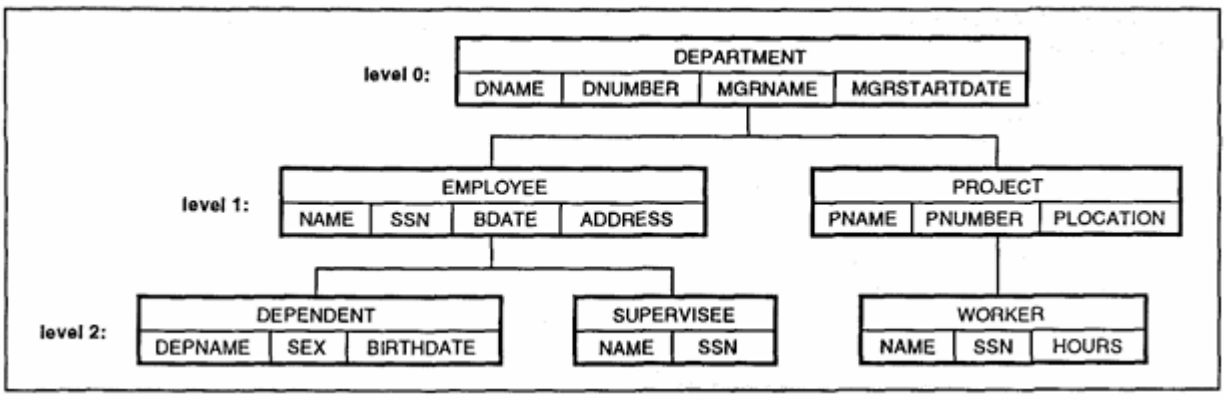


Рисунок 1.1 - Ієрархічна схема бази даних

у різних середовищах програмування реалізовано різні алгоритми

інтелектуального аналізу просторових даних. Щоб кінцеві користувачі могли скористатися перевагами кількох підходів до інтелектуального аналізу просторових даних, існує потреба в розробці програмної системи, яка інтегруватиме всі реалізовані методи в єдиному середовищі та таким чином зменшить зусилля користувача щодо планування своїх управлінських дій.

Точне землеробство є одним із застосувань, яке може процвітати завдяки новим методам просторового аналізу даних [10]. Сільськогосподарські виробники збирають великі обсяги просторових даних за допомогою систем глобального позиціонування для географічного визначення показань датчиків та місць відбору проб. Сподіваємося, що ці дані призведуть до покращення управління в межах поля та призведуть до більшої економічної віддачі та охорони навколишнього середовища [6]. Однак, як відомо, стандартних методів аналізу даних недостатньо для точного землеробства через просторовий вимір даних. Тому для точного землеробства та інших згаданих вище застосувань необхідні методи просторового аналізу даних, щоб успішно виконувати аналіз та моделювання даних.

Крім того, дані точного землеробства за своєю суттю розподілені на кількох фермах і не можуть бути локалізовані на одній машині з різних практичних причин, включаючи фізично розподілені набори даних по багатьох різних географічних місцях, служби безпеки та конкурентні причини. Зі зростанням мереж це часто спостерігається в інших областях. У таких ситуаціях вигідно мати розподілену систему інтелектуального аналізу даних, яка може навчатися з великих баз даних, розташованих на кількох сайтах обробки даних. Система JAM [11], призначена для навчання з таких баз даних, являє собою розподілений, масштабований та портативний пакет програмного забезпечення для інтелектуального аналізу даних на основі агентів, який використовує загальний підхід до масштабування програм інтелектуального аналізу даних. JAM надає набір навчальних програм, які обчислюють моделі з даних, що зберігаються локально на сайті, та набір методів для об'єднання кількох моделей, вивчених на різних сайтах. Однак програмна система JAM не надає жодних інструментів для просторового аналізу даних.

Таким чином, наша програмна система намагається підтримувати гнучкий просторовий аналіз даних у централізованих або розподілених сценаріях. Окрім надання інтегрованого інструменту для більш систематичного експериментування фахівцями з аналізу даних, наш проект має на меті запропонувати просту у використанні програмну систему для аналізу даних для людей без технічних знань, зазвичай експертів у своїх галузях, але з невеликими знаннями аналізу даних та інтелектуальних методів аналізу даних.

Нашою головною метою було створити тестове середовище як для стандартних, так і для просторових алгоритмів інтелектуального аналізу даних, яке могло б швидко генерувати статистику продуктивності (наприклад, точність прогнозування), порівнювати різні алгоритми на кількох наборах даних, реалізовувати гібридні алгоритми (наприклад, бустінг, нелінійні регресійні дерева) та графічно відображати проміжні результати, вивчену структуру та кінцеві результати прогнозування. Для ефективного досягнення цієї мети ми розробили програмну систему SDAM (просторовий аналіз та моделювання даних), яка виконує програми, розроблені в різних середовищах (C, C++, MATLAB), за допомогою єдиного керування та простого графічного інтерфейсу користувача (GUI) [7].

1.2 Основні терміни та концепції у сфері баз даних

У цьому розділі ми визначимо деякі основні терміни для області баз даних, що стосуються моделювання даних. Модель даних – це набір концепцій, які можна використовувати для опису

Структура бази даних та операції з нею. Під *структурою бази даних* ми маємо на увазі типи даних, зв'язки та обмеження, які визначають «шаблон» цієї бази даних. Модель даних повинна забезпечувати операції з базою даних, які дозволяють пошук та оновлення, включаючи вставку, видалення та модифікацію [8]. Зауважте, що ми використовуватимемо термін «модель даних» для позначення дисципліни моделювання даних певним чином - таким, що забезпечує

будівельні блоки або моделі**,** за допомогою яких можна описати структуру бази даних. Ми використовуватимемо термін « модель застосування» для позначення опису даних для конкретної бази даних. Наприклад, реляційна модель - це модель даних. Визначення конкретної бази даних для програми персоналу в компанії X називатиметься моделлю застосування цієї бази даних, яка використовує реляційну модель. Аналізатори застосувань часто називають останню моделлю даних, що викликає плутанину.

- якій програмі важливо розрізняти *опис* бази даних та *саму базу даних.* Цей опис називається схемою бази даних**.** Схема бази даних розробляється для заданого набору програм та користувачів шляхом аналізу вимог. Вона описується за допомогою певної моделі даних, яка надає моделі у вигляді синтаксису мови або схем. У деяких інструментах або СУБД модель даних явно не визначена, але неявно присутня з точки зору наявних функцій. Схеми зазвичай залишаються відносно стабільними протягом усього терміну служби бази даних для більшості програм [9]. Для динамічних середовищ, таких як системи автоматизованого проектування (CAD) або автоматизована розробка програмного забезпечення (CASE), схема продукту, що розробляється, може сама змінюватися. Це називається еволюцією схеми**.** Більшість СУБД обробляють дуже обмежену кількість еволюції схеми внутрішньо.

Під час процесу проектування бази даних схема може зазнавати трансформації з однієї моделі в іншу. Наприклад, схема бази даних персоналу може бути спочатку описана за допомогою моделі даних «сутність-зв'язок» у вигляді ER-діаграми. Потім вона може бути відображена в реляційну модель даних, яка використовує структуровану мову запитів (SQL) - новий стандарт, для визначення схеми [10]. Вся діяльність, починаючи з вимог і закінчуючи визначенням кінцевої реалізованої схеми в СУБД, називається проектуванням схеми. СУБД використовується для зберігання бази даних, що відповідає цій схемі; вона дозволяє обробляти транзакції бази даних , в яких транзакція є одиницею дії з базою даних, що включає операції пошуку та оновлення з бази

даних. Транзакція повинна бути виконана повністю, залишаючи «постійну» зміну в базі даних, або може бути перервана, взагалі не виконуючи жодних змін.

Фактична база даних відображає стан реального світу, що стосується програми, або «міні-світу». Вона повинна залишатися узгодженою з міні-світом, відображаючи фактичні зміни, що відбуваються. Дані в базі даних у певний момент часу називаються «екземпляром бази даних» або «станом бази даних». Фактичні випадки або екземпляри даних змінюються дуже часто, на відміну від схеми, яка залишається статичною.

Модель даних у термінології баз даних пов'язана з різноманітними мовами: мовою визначення даних, мовою запитів, мовою маніпулювання даними, якщо назвати найважливіші. Мова визначення даних (DDL) дозволяє адміністратору бази даних або розробнику баз даних визначати схему бази даних [11]. СУБД має компілятор для обробки визначення схеми в DDL та перетворення його у форму, придатну для машинної обробки. Таким чином, створюється централізоване визначення моделі програми, на основі якого можна визначити низку програм. Мова маніпулювання даними (DML) – це мова, що використовується для визначення пошуку, вставки, видалення та модифікації даних. DML можна умовно розділити на дві категорії: декларативні та процедурні. Перші дозволяють користувачеві вказати результат (запиту), який його цікавить, тоді як другі вимагають надання процедури для отримання результату запиту. Характер мови також залежить від моделі даних. Наприклад, хоча для такої моделі, як реляційна модель, можливо мати декларативну мову, мова для мережевої моделі даних є «навігаційною», яка вимагає від користувача вказати, як переміщатися по базі даних, і тому є за своєю суттю процедурною [12]. Будь-який тип мови може використовуватися окремо в інтерактивному режимі як мова запитів**.** Мови для моделей даних також можна розрізняти за тим, чи є вони записами за раз, чи наборами за раз. Обробка записів за раз вимагає складної структури керування, яка зазвичай надається основною мовою програмування**,** в яку вбудовані команди або дієслова DML. Обробка, орієнтована на множини, розглядає дані як набори елементів (наприклад, набори кортежів у реляційній моделі) та надає оператори,

які застосовуються до цих наборів, генеруючи нові набори. Зараз спостерігається рух до надання мов, які безшовно інтегрують можливості забезпечення обчислень загального призначення та спеціалізованого маніпулювання даними з базою даних однією мовою. Вони називаються мовами програмування баз даних (DBPL) [6].

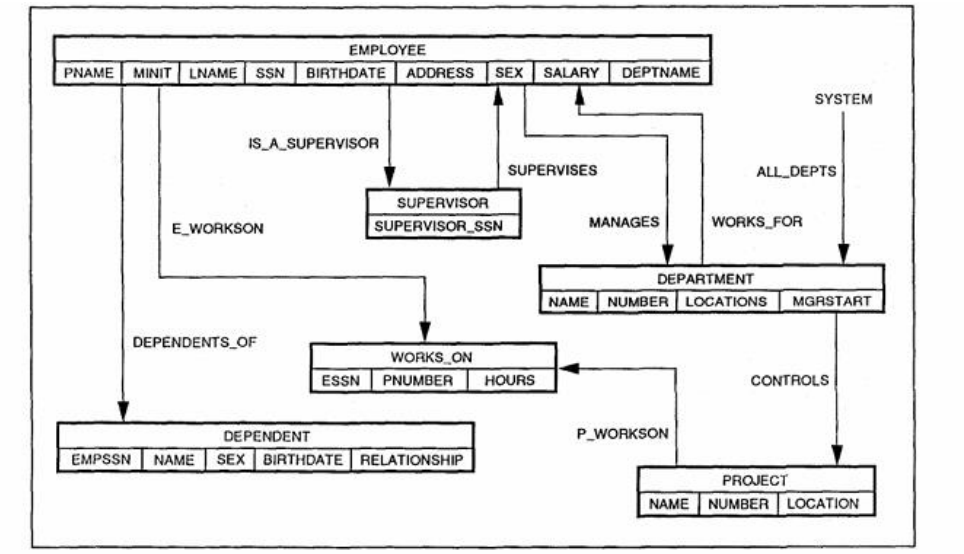


Рисунок 1.2 - Схема мережевої бази даних

У традиційному розумінні, моделі даних, що використовуються для проектування схем баз даних, мають обмежений обсяг [13]. Вони використовувалися для моделювання *статичних* властивостей даних, включаючи наступне:

Структура даних: Структура виражається через те, як атомарні типи даних агрегуються в типи вищого порядку. Крім того, моделі виражають зв'язки між цими агрегатами. Ранні моделі, як правило, були орієнтовані на записи, причому базова агрегатна структура була типом запису, що складався з типів елементів даних або типів полів. Схема бази даних складається з кількох типів записів, які пов'язані між собою різними способами. Ієрархічна модель даних організовує типи записів у деревоподібну структуру, тоді як мережева модель даних організовує схему бази даних з типами записів як вузлами графа. Обмеження записного підходу до моделювання даних обговорюються. Реляційна модель запровадила орієнтований на множини підхід до моделювання даних [16],

і наразі об'єктно-орієнтований підхід, який структурує базу даних з точки зору об'єктів та міжоб'єктних взаємодій, набирає популярності [3]. Ми обговоримо ці підходи більш детально пізніше в цій роботі.

Обмеження: Обмеження – це додаткові обмеження на входження даних у базу даних, які повинні бути чинними *постійно.* Обмеження моделі даних служать двом основним цілям:

1. Цілісність: Обмеження цілісності – це правила, що обмежують допустимі стани бази даних. Вони виникають або як властивості даних, або як визначені користувачем правила, що відображають значення даних.

2. Безпека та захист: це стосується обмежень та авторизаційних - обмежень, що застосовуються до бази даних для захисту її від неправильного та несанкціонованого використання.

Обмеження можна візуалізувати на різних рівнях:

- Внутрішні обмеження стосуються обмежень, вбудованих у правила самої моделі даних. Наприклад, у моделі «сутність-зв'язок» зв'язок повинен мати щонайменше два типи сутностей-учасників (залежно від варіанта використовуваної моделі, один і той самий тип сутності може використовуватися двічі).

- Неявні обмеження – це обмеження, які можна визначити за допомогою DDL моделі даних для опису додаткових властивостей. Очікується, що вони будуть автоматично застосовані. Прикладом є обов'язкове обмеження участі в моделі «сутність-зв'язок», яке стверджує, що певна сутність повинна брати участь у певному зв'язку.

- с) явні обмеження – це обмеження, що залежать від програми, які визначають семантичні обмеження, пов’язані з програмою. Це найзагальніші та найскладніші для детального опису обмеження. Існує загальна тенденція фіксувати якомога більше інформації про «поведінку програми» в базі даних, щоб централізовано її зберігати. Рух 4GL спрямований на фіксацію цієї семантики програми на високому рівні. Однак сучасні СУБД не оснащені для легкої обробки таких обмежень.

Ще один вимір моделювання обмежень полягає у фіксації переходу станів, а не лише статичної інформації про стан. Це призводить до появи динамічних обмежень , які виражаються в тому, які типи змін є дійсними в базі даних. Наприклад: «зарплата працівника може лише зростати». Як статичні, так і динамічні обмеження дозволяють нам визначити, чи є база даних узгодженою. У той час як статичні обмеження можна оцінити на «знімку» бази даних, щоб визначити її дійсність, динамічні обмеження набагато складніше забезпечити, оскільки вони передбачають блокування/запобігання переходу станів «під час виконання».

Інші параметри моделей даних: *А*

Модель даних для бази даних може також визначати деякі додаткові деталі, що стосуються використання даних. Однією з можливих функцій є «параметри розподілу». Вони стосуються фрагментації даних з точки зору того, як дані зберігаються як фрагменти. У реляційній моделі прийнято називати «горизонтальні» та «вертикальні» фрагменти [14]. Перші містять підмножини входжень даних у відношення, які відповідають певній умові предиката, тоді як другі посилаються на підмножину атрибутів даних для всього відношення. У відношенні під назвою ORDERS кожен горизонтальний фрагмент може містити замовлення, що відправляються з одного складу, тоді як ORDER може бути вертикально фрагментовано на інформацію про доставку та платіжну інформацію. Безпека – це ще одна функція, яка може бути вбудована в модель даних на різних рівнях деталізації. Ще однією особливістю є надмірність, яку важко моделювати явно; вона може бути зафіксована у формі специфікації явних копій або даних, що перекриваються. Модель повинна дозволяти специфікацію таких функцій, як ключі , які однозначно ідентифікують дані; він також може мати спосіб вказати фізичні параметри, такі як кластерне кільце або індексування (наприклад, індекс дерева B+ для певного поля) , як частину специфікації моделі застосунку.

Представлення: У моделюванні баз даних представлення – це сприйнята модель програми, визначена користувачем або групою програм. Представлення – це ще один механізм, за допомогою якого програма може записувати свої

конкретні вимоги, на додаток до явних обмежень, згаданих раніше. Більшість моделей даних надають мову для визначення представлень: вона може збігатися з мовою запитів, за допомогою якої результат запиту визначається як представлення [15]. Як правило, представлення створюється, коли є запит на отримання з нього даних, а не «матеріалізується» представлення, оскільки останнє створює надлишкові дані.

До спільного моделювання даних та застосунків

Щоб забезпечити належну підтримку моделювання динамічних середовищ додатків, інша школа мислення поєднує функціональний аналіз, який зазвичай є фокусом програмної інженерії під час проектування інформаційних систем, зі звичайним моделюванням даних [11, 26], що призводить до спільного аналізу та моделювання вимог до додатків та даних. Ми також відстоювали цю точку зору під час концептуального моделювання баз даних [8]. Щодо поточного випуску журналу *«Комунікації»,* попередній підхід видається найбільш актуальним та значущим. Запропоновано методологію структурованого проектування «зверху вниз» для моделювання даних та процесів з використанням розширених діаграм ER та діаграм потоків даних відповідно. Вона пропонує побудову «міні-ER-схем процесів» на основі найбільш уточнених діаграм процесів з використанням деяких простих правил. Ці міні-схеми потім інтегруються в загальну модель за допомогою методів інтеграції представлень [7], щоб створити глобальну модель даних. Пропонуємо, що проектування глобальної схеми для набору застосунків має відбуватися як «скелелазіння », чергуючи послідовне розширення моделі даних та моделі процесу, а також перехресну перевірку на кожному кроці, щоб переконатися, що модель даних враховує вимоги обробки, а модель процесу посилається на дані, визначені на попередньому кроці. Перехресна перевірка полегшується завдяки підтримці словника даних [16]. Як методологія спільного проектування описані методології «зверху вниз», «знизу вгору» та змішані або «збоку назовні». Було показано, що модель ER, яка зазвичай використовується для моделювання даних, може сама використовуватися з деякими додатковими конструкціями для таких завдань, як упорядкування та секвенування, щоб

адаптувати її до моделювання процесів.

Галузь моделювання транзакцій як частини моделювання даних є відносно новою. Мова специфікації транзакцій та інструмент, що використовує об'єктну модель [17]. Ми визначаємо мову для специфікації транзакції відносно «нейтральної» моделі даних на концептуальному рівні та показуємо її відображення в SQL, припускаючи, що транзакція виконується відносно базової реляційної бази даних.

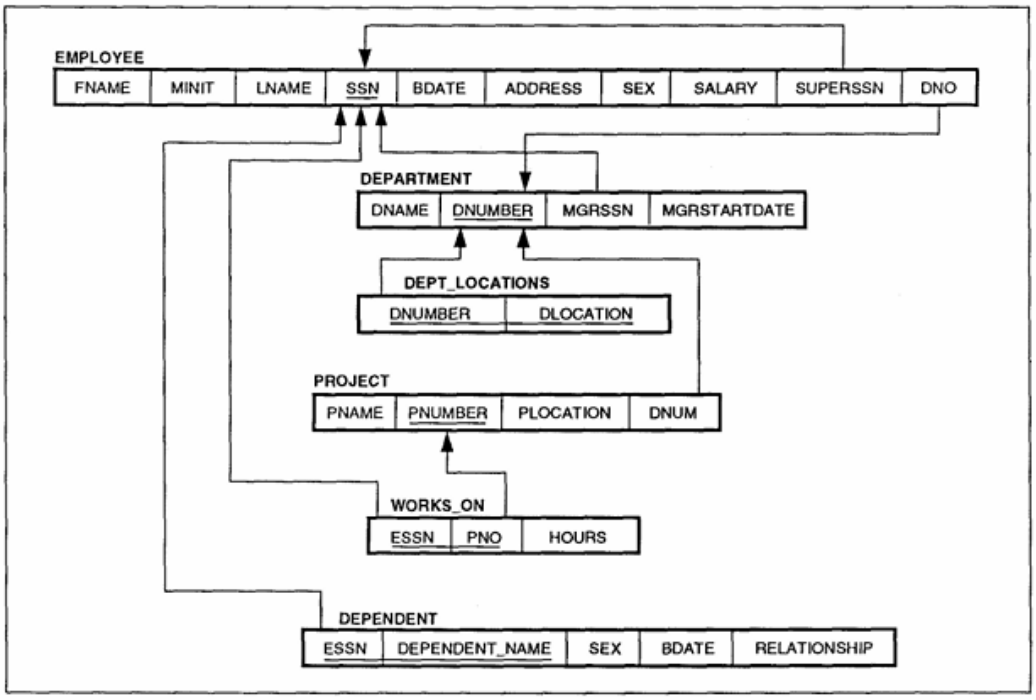


Рисунок 1.3 - Реляційна схема з референційним обмеженням

Класифікація моделі даних

Моделі даних для управління базами даних можна класифікувати за двома основними вимірами [18]. Перший вимір стосується етапів *загальної діяльності з проектування бази даних,* до якої застосовується модель. Процес проектування бази даних складається з послідовного зіставлення вимог до даних та програм через наступні кроки.

- Концептуальне проектування - тут модель забезпечує спосіб фіксації сприйняття даних користувачами. Концептуальна модель даних надає концепції,

необхідні для підтримки середовища застосунку на дуже високому, несистемному рівні.

- Логічне проектування - тут модель представляє організацію даних для деякої реалізованої моделі даних, яка називається логічною моделлю даних або моделлю реалізації. Ця модель має моделі, які легко зрозуміти користувачам, уникає фізичних деталей реалізації, але зазвичай призводить до прямої комп'ютерної реалізації в деяких СУБД.
- Фізичне проектування - зазвичай на цьому етапі моделювання даних не використовується для опису даних. Фізичне проектування складається з різноманітних варіантів зберігання даних з точки зору кластеризації, розділення, індексації, забезпечення додаткового доступу або структур каталогів тощо. Було проведено певну роботу з розробки концепцій або опису фізичних реалізацій за зразком моделі даних.

У трирівневій архітектурі комітету ANSI/SPARC з управління базами даних [4] чітко визначені схеми, що відповідають трьом рівням архітектури СУБД. Вони називаються концептуальною, зовнішньою та внутрішньою схемами. Зв'язок термінології ANSI/SPARC з нашими кроками проектування бази даних показано в таблиці 2.1.

Зверніть увагу, що ми навмисно додали крок проектування під назвою проектування перегляду, щоб врахувати поняття зовнішньої схеми з номенклатури ANSI/SPARC. Зазвичай він об'єднаний з кроком логічного проектування.

В іншому вимірі, моделі даних можна класифікувати на моделі на основі записів, семантичні моделі даних та об'єктно -орієнтовані моделі за шкалою «гнучкісті» або «експресивності». Гнучкість стосується легкості, з якою модель може справлятися зі складними ситуаціями застосування [19]. Експресивність аналогічно стосується здатності виявляти різні абстракції та зв'язки у відповідному застосуванні.

Моделі на основі записів були моделями, запозиченими з файлів, що призвело до появи ієрархічних та мережевих моделей даних. Вони були навряд чи

гнучкими та виразними в обмеженій мірі [23], але відіграли важливу роль як моделі реалізації для всього розвитку СУБД наприкінці 1960-х та 1970-х років. Реляційна модель, що є відгалуженням від попередніх моделей, забезпечувала більшу незалежність від даних, «піднімаючи» модель вище, віддаляючи її від деталей фізичної реалізації, а також забезпечувала більшу потужність з точки зору операцій з моделлю по черзі. Далі з'явилися семантичні моделі даних, які були більш придатними для концептуального проектування. Модель «сутність-зв'язок» [14] є прикладом семантичного режиму даних і була попередником багатьох подальших розробок. Семантичні мережі мають велику схожість із семантичними моделями даних, за винятком того, що семантичні зв'язки в останніх мають обмежений обсяг порівняно з першими, а дані, змодельовані семантичною моделлю даних, зазвичай є вторинним сховищем. Об'єктно-орієнтовані моделі усувають відмінність між сутностями та зв'язками, яка є спільною для моделі «сутність-зв'язок» та її розширень. На цю область вплинули об'єктно-орієнтовані мови програмування, такі як Smalltalk і стала життєздатним підходом до моделювання даних, особливо з огляду на нові застосування систем баз даних. Нещодавно група відомих дослідників навіть опублікувала маніфест [3], щоб визначити, що таке справжня «об'єктно-орієнтована» (ОО) система управління базами даних. Моделі даних 0-0 застосовуються як до концептуальної, так і до логічної фаз проектування. У решті цієї роботи ми висвітлимо концепції, що лежать в основі класів моделей, описаних раніше, та їхній внесок у загальну галузь моделювання даних.

1.3 Реляційна модель даних та її особливості

Реляційна модель даних, стала знаковою розробкою, оскільки вона забезпечила математичну основу дисципліни моделювання даних на основі понять множин та відношень. Реляційна модель даних організовує дані у вигляді відношень (таблиць) [20]. Ці таблиці складаються з кортежів (рядків) інформації, визначеної за набором атрибутів (стовпців). Атрибути, у свою чергу, визначені за

набором атомарних доменів значень. Завдяки простоті моделювання вона здобула широку популярність серед розробників бізнес-додатків. Сьогодні на ринку представлено низку відомих продуктів СУБД.

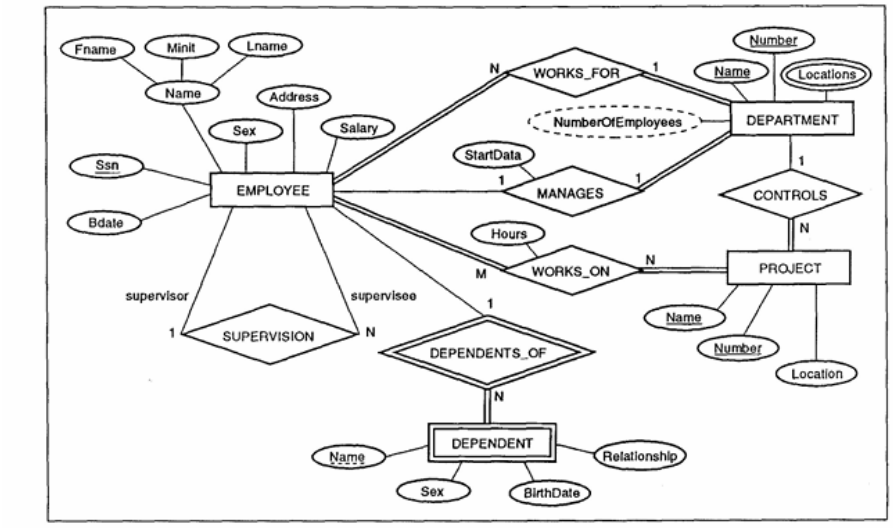


Рисунок 1.4 - Концептуальна схема бази даних у форматі «сутності-відношення»

(DB2, RDb, INGRES, Oracle, INFORMIX , SyBase, і це лише деякі з них) надають доступ до реляційної моделі широкому колу користувачів. Модель має пов'язану алгебру, яка включає операції вибору, проекції, з'єднання, а також операції об'єднання, перетину, різниці, декартового добутку тощо. Характер цих операцій, що базується на принципі "множина за раз", робить їх дуже потужними, оскільки цілі таблиці стають аргументами операторів [21]. Структурована мова запитів SQL, заснована на реляційному численні, яке є формою числення предикатів першого порядку, стає *фактичним* стандартом для галузі обробки даних. Ця декларативна мова, пов'язана з реляційною моделлю, надає системі багато можливостей для прийняття "інтелектуальних рішень" щодо обробки цих операцій з множинами. Що стосується обмежень, то існують два типи обмежень, які підпадають під категорію неявних обмежень для реляційної моделі даних. Перше, яке називається обмеженням "цілісності сутності", гарантує, що жодні два кортежі, що належать до одного відношення, не посилаються на одну й ту саму сутність реального світу. Іншими словами, це гарантує унікальність ключів. Друге обмеження, яке називається «обмеженням посилальної цілісності», гарантує, що

щоразу, коли стовпець в одній таблиці отримує значення з ключа іншої таблиці, ці значення повинні бути узгодженими. На жаль , усі існуючі комерційні реляційні СУБД не дозволяють належної специфікації та автоматичного застосування цих обмежень. Але постачальники розробляють різноманітні способи їх вирішення. На рисунку 1.3 показано набір відношень, з'єднаних стрілками посилальних обмежень, які еквівалентно представляють ті ж дані, що й схеми на рисунках 1.4 та 1.5.

Інші особливості реляційної моделі даних, які зробили її популярним вибором для додатків баз даних сьогодні, належать до наступного :

- Основи реляційної алгебри та проста й точна семантика роблять можливим значний обсяг оптимізації запитів у системі, тим самим полегшуючи навантаження програмістів. Це не так у мережевих та ієрархічних моделях, в яких програміст вручну оптимізує навігацію моделі.
- Модель дозволяє чітко розділити логічні параметри від фізичних, що полегшує звичайним користувачам розуміння та роботу з базою даних. На фізичному рівні типові СУБД дозволяють використовувати різноманітні індекси та функції підвищення продуктивності.
- Теорія, що лежить в основі паралельної обробки транзакцій, а також відновлення, була добре розроблена для цієї моделі, щоб забезпечити адекватну продуктивність для більшості комерційних застосувань.
- Для подальшого розвитку розподілених баз даних реляційна модель забезпечує надійну основу для вирішення проблем, пов'язаних з фрагментацією, надмірністю та розподіленою обробкою транзакцій у різних режимах загалом.

Головним аргументом проти реляційної моделі даних є її «плоскостність» структури, через яку вона втрачає цінну інформацію, що міститься у зв'язках або «зв'язках» між даними. Тому їй явно бракує характеристик виразності та семантичного багатства, завдяки яким семантичні моделі даних є кращими.

Семантичні моделі даних

Як показано на рисунку 1.4, діяльність зі збору та аналізу вимог створює набір вимог до бази даних, які мають бути враховані моделлю даних на

наступному кроці, який називається концептуальним проектуванням. Цей наступний крок найкраще виконується за допомогою високорівневої моделі, яка називається концептуальною або семантичною моделлю даних. Важливо використовувати високорівневу модель на цьому кроці, оскільки на цьому етапі аналізу база даних уявляється потенційним набором користувачів дуже попередньо. Вона не прив'язана до конкретної моделі реалізації і, звичайно ж, до конкретної СУБД. Одним із недоліків діяльності з проектування баз даних в організаціях була неувага до концептуального проектування бази даних та передчасна зосередженість на певній цільовій СУБД. Проектувальники дедалі більше усвідомлюють важливість діяльності з концептуального проектування бази даних [21]. Моделі, які зазвичай використовуються для цієї діяльності, називаються семантичними моделями даних. Семантична модель даних, що використовується для цілей концептуального проектування, повинна мати такі характеристики [8]:

- Виразність: Модель повинна бути достатньо виразною, щоб виявити відмінності між різними типами даних, зв'язками та обмеженнями.
- Простота: Модель має бути достатньо простою для використання та розуміння кінцевим користувачем. Отже, вона завжди повинна супроводжуватися легкою діаграмою.
- Мінімальність: Модель повинна складатися з невеликої кількості основних понять, які є різними та ортогональними за своїм значенням.
- Формальність: Концепції моделі повинні бути формально визначені. Повинна бути можливість сформулювати критерії валідності схеми в моделі.
- Унікальна інтерпретація: В ідеалі має існувати єдина семантична інтерпретація заданої схеми. У свою чергу, це означає, що для кожної моделі має бути визначена повна та однозначна семантика.

Раннім кандидатом на семантичну модель даних була модель «сутність-зв'язок» [14], яку зазвичай називають ER-моделлю. З точки зору попередніх - критеріїв, вона досить проста у використанні, має лише три основні конструкції, які є досить, але не повністю, ортогональними, була формалізована та має досить

унікальну інтерпретацію з легкою діаграмною нотацією. Ми не будемо зупинятися на описі моделі тут. На рисунку 1.4 показано концептуальну модель бази даних, представленої в ER-моделі для того ж застосування, що й на рисунках 1.2 та 1.3 для ієрархічної та мережевої схеми робіт адаптовані [19].

Базовими конструкціями в моделі ER є типи сутностей, типи зв'язків та типи атрибутів. На діаграмі вони представлені прямокутниками, ромбами та овалами відповідно. Різниця між сутностями та зв'язками, як правило, нечітка; аналогічно, те, що можна назвати атрибутом (наприклад, Відділ є атрибутом типу сутності Співробітник), насправді може бути гідно називатися типом сутності (наприклад, Відділ як тип сутності з атрибутами Місцезнаходження, Ім'я_менеджера тощо) [22]. Через недостатню точність моделювання, модель не стала основою для формальних робочих областей, таких як розподіл та обробка транзакцій, як це сталося з реляційною моделлю даних. Вона залишається улюбленою серед розробників інструментів в інструментах, які створюють та редагують діаграми ER як засіб для легкої комунікації на ранніх етапах проектування бази даних, а потім напівавтоматично відображають схеми ER на логічні та фізичні схеми цільових СУБД.

Модель ER отримала багато розширень. Ранні роботи над розширеннями, визначили інваріантні властивості моделі ER. Модель була додатково вдосконалена концепціями класів та підкласів , а також ієрархій успадкування на основі узагальнення та спеціалізації. Нещодавно до моделі було додано правила [36]. Детальне обговорення вдосконаленої моделі ER можна знайти в розділі роботи [19].

Фундаментальна ідея полягає в тому, щоб збагатити модель для підтримки всіх абстракцій даних. Якщо усунути різницю між сутністю та зв'язком, це призведе до загальної моделі даних з класами або типами об'єктів. У цьому сенсі, запропоновано модель семантичної ієрархії, засновану на поняттях агрегації та узагальнення. Дуже універсальний підхід, який забезпечує багатий репертуар можливостей для групування, зв'язування та обмеження визначень класів [23]. Фундаментальні абстракції, які повинна підтримувати семантична модель даних,

наведені в наступному списку. Ці абстракції даних однаково застосовні до об'єктно-орієнтованих моделей, які будуть обговорюватися.

- Агрегація: це концепція абстракції для побудови агрегованих об'єктів з об'єктів-компонентів. Зв'язок між об'єктом нижчого порядку та об'єктом вищого порядку описується як зв'язок «IS_PART_OF». На іншому рівні ця абстракція використовується для агрегації атрибутів в об'єкт. У моделі ER агрегація використовується, коли два або більше типів сутностей пов'язані для визначення типу зв'язку.
- Ідентифікація: це процес, за допомогою якого абстрактні поняття, а також конкретні об'єкти робляться унікальними за допомогою певного ідентифікатора.
- Класифікація та створення екземплярів: Класифікація передбачає класифікацію подібних об'єктів у класи об'єктів. Зв'язок між об'єктом та класом - це зв'язок "IS_MEMBER_OF". Протилежністю класифікації є "створення екземплярів". Деякі моделі дозволяють використовувати цю абстракцію серед класів, де метаклас загалом означає низку класів, які є його членами .
- Концепція підкласу та суперкласу: Екземпляри одного типу сутності можуть бути підмножиною екземплярів іншого типу сутності. Наприклад, тип сутності STUDENT є підкласом типу сутності PERSON. Зв'язок між ними називається зв'язком «IS_A».
- Успадкування атрибутів: Підклас автоматично успадковує атрибути свого суперкласу.
- Ієрархії узагальнення: це ієрархії класів, де суперклас пов'язаний з низкою своїх підкласів на основі певної відмінної ознаки. Наприклад, суперклас PERSON може мати підкласи STUDENT та EMPLOYEE. Підклас STUDENT далі поділяється на GRAD_STUDENT та UNDER GRAD-STUDENT, тоді як EMPLOYEE поділяється на FULL_TIME та PARTIAL. Вся ієрархія класів являє собою абстракцію узагальнення, що рухається вгору по ієрархії, та спеціалізації, що рухається вниз по ієрархії. Обмеження, що керують суперкласом та його

підкласами, можна змоделювати як повну чи часткову участь та обов'язкову чи факультативну участь [7].

Бінарна модель даних, наприклад, NIAM [37], також досить популярна. Порівняно з підходом ER, бінарний підхід можна розглядати як висхідний, оскільки він будується з атомарних об'єктів у термінах бінарних зв'язків. Для охоплення семантики вищого рівня вони вдаються до різноманітних обмежень .

Іншим класом семантичних моделей даних є функціональні моделі даних [24]. Вони використовують об'єкти та функції над об'єктами як основні будівельні блоки. Будь-який запит інформації можна візуалізувати з точки зору функціонального виклику з аргументами. Функції на примітивному рівні мають об'єкти як аргументи; але можливо будувати функції над функціями. Модель DAPLEX [34] була однією з перших відомих моделей у цій категорії . Нещодавно була розроблена модель даних IRIS [20] та повністю реалізована на основі концепцій об'єктів та функцій. Відкрита система ODB компанії Hewlett-Packard, заснована на реалізації IRIS, незабаром буде комерційно доступна. Головною перевагою підходу функціонального моделювання є його формальна математична основа функцій, яку можна довільно комбінувати для створення функцій вищого порядку. У поєднанні з попередніми абстракціями та принципами об'єктно-орієнтованого підходу це створює потужну модель даних.

Об'єктно-орієнтовані моделі даних

У цьому розділі ми розглянемо низку «інших» моделей даних. Однією з основних областей є об'єктно-орієнтовані (ОО) моделі даних. Вони подібні до семантичних моделей з точки зору наступного:

- Структурна абстракція: Обидва про див. структурні абстракції, які були окреслені. Хоча успадкування є обов'язковим у всіх об'єктно-орієнтованих моделях, це не так у всіх семантичних моделях даних.
- Композиція значень та типів: Обидва забезпечують конструктори типів та значень, які дозволяють розробникам додатків створювати вищі форми значень або «масивні типи», такі як списки, набори, пакети та кортежі.

- Області, в яких моделі даних 0-0 відрізняються від семантичних моделей даних:
- Обробка ідентифікаторів: Семантичні моделі створюють ідентифікатори або ключі на основі внутрішніх атрибутів або внутрішніх властивостей. Об'єктно-орієнтовані моделі використовують ідентифікатори, зовнішні по відношенню до об'єкта, завдяки чому ідентифікатори залишаються незмінними, тоді як об'єкти можуть зазнавати змін. Ці ідентифікатори називаються «сурогатами» або системними ідентифікаторами.
- Обробка успадкування: Семантичні моделі реалізують лише успадкування атрибутів, і воно обмежене між підтипами та надтипами. Моделі 0-0, з іншого боку, забезпечують як структурне, так і поведінкове успадкування [24]. Це стосується успадкування структурних властивостей та процедур або методів. Успадкування серед типів, які не пов'язані зв'язком тип-підтип, також може бути дозволено, але це не рекомендується.
- Приховування інформації та інкапсуляція: методи, інкапсульовані в об'єктному типі, дозволяють доступ до його змінних екземпляра. Немає способу отримати доступ до цих змінних екземпляра (або атрибутів).

Крім того, очікується, що моделі даних 0-0 обов'язково повинні забезпечувати такі характеристики [3]:

- Успадкування властивостей та методів
- Підтримка складних об'єктів
- Ідентичність об'єкта
- Перевантаження операторів: це стосується використання операцій з однаковими іменами, що мають різне значення, в контексті різних типів об'єктів.
- Розділення публічної та приватної частин об'єктів.

Як згадувалося раніше в цій роботі, ми не будемо детально обговорювати об'єктно-орієнтоване моделювання та аналіз, оскільки весь цей випуск присвячений цій темі. У літературі було запропоновано кілька моделей даних 0-0, які були реалізовані в системах, подібних до GemStone [17], IRIS [20] та Оригіналу. Також було представлено низку інших комерційних 0-0 СУБД,

включаючи Object Design, ObjectStore та Versant . Сподіваємося, що ці моделі - стануть на допомогу користувачам, які отримають такі переваги, які зазвичай приписуються 0-0 моделям: краще моделювання поведінки, легше моделювання складних об'єктів, розширюваність типів та локалізація впливу змін у визначенні даних. Об'єктні моделі можна розглядати головним чином як моделі реалізації, такі як ієрархічні та мережеві моделі. З іншого боку, їх також можна розглядати як семантичні моделі даних. Нещодавній набір посилань в області 0-0 СУБД.

Динамічні та активні моделі баз даних

Основний аргумент, який можна висунути проти традиційного моделювання даних у галузі баз даних, полягає в тому, що воно неадекватно враховувало моделювання поведінки даних, а отже, і «активні» або динамічні аспекти бази даних. Парадигма активної бази даних була запропонована і пізніше була включена в моделі даних таких систем, як POSTGRES в Каліфорнійському університеті в Берклі та Starburst [38] в IBM Almaden Center. Активна СУБД – це система з додатковою можливістю, на додаток до стандартної СУБД, моніторити стан бази даних і виконувати деякі заздалегідь визначені дії, коли відбуваються відповідні заздалегідь визначені події. Динамічний аспект можна додатково поділити на дві частини:

a. Процесно-орієнтований погляд: це стосується постійно мінливого середовища програми. Весь набір програм, яким підлягає база даних, можна розглядати як середовище обробки з відомими процесами або діями . Цей погляд потім може бути змодельований та підтримуваний у СУБД, b. Подійно-орієнтований погляд: ми раніше вказували на природу обмежень цілісності, включаючи динамічні обмеження, у розділі «Область застосування моделей даних» [25]. Подійно-орієнтований погляд визначає різні типи подій, що відбуваються під час виконання під час обробки бази даних (тобто в результаті таких подій, як вставка або видалення), та розглядає дії, необхідні для забезпечення дотримання обмежень цілісності.

Правила були запропоновані як засіб фіксації динамічного аспекту даних

[13, 38] і можуть бути поєднані з існуючими стандартними моделями, такими як модель ER, щоб надати останній активну можливість [36]. Ми запропонували альтернативний спосіб, який називається парадигмою структури-функції , що дозволяє розглядати функції нарівні зі структурою в моделюванні даних [18, 29]. У цій роботі пропонується розглядати функції як об'єкти першого класу, що генерують дві окремі решітки класів для функції та структури. Це призводить до появи складних структурних об'єктів та складних функціональних об'єктів [30], які самі по собі пов'язані з третьою групою об'єктів, які називаються об'єктами взаємодії. Такий шар моделювання поверх традиційної моделі 0-0 надає моделюванню велику гнучкість для роботи зі складними фізичними системами, типовими для інженерних даних (наприклад, складна машина, атомна електростанція або біологічні області, включаючи людський організм) [26]. Ми розширили цей підхід **,** включивши керування як додатковий компонент для явного моделювання [12], що особливо корисно в моделюванні складних систем. Загалом, галузь моделювання даних не зробила багато для підтримки зберігання даних для керування контрольованими симуляціями та для збору даних, отриманих в результаті симуляцій. Включення цих ідей у моделі даних призводить до модифікації даних.

1.4 Висновок по розділу

У першому розділі було розглянуто теоретичні основи моделювання даних у програмних системах. Проаналізовано основні поняття та сутність моделювання даних як важливої складової проєктування інформаційних систем. Досліджено ключові терміни та концепції у сфері баз даних, що забезпечують організацію, збереження та обробку інформації. Також розглянуто реляційну модель даних та її особливості, які визначають принципи побудови сучасних баз даних. Отримані результати формують теоретичну основу для подальшого дослідження методів і технологій моделювання даних.

РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ МОДЕЛЮВАННЯ ДАНИХ

2.1 Основні методи моделювання даних

У цьому підрозділі наведено детальний огляд найпопулярніших моделей даних, що використовуються для визначення та підтримки операційних баз даних, сховищ даних та технологій великих даних.

Таблиця 2.1

Підходи та перспективи дослідження.

Підходи	Операційний (Operational)	Підтримка рішень (Decision Support)	Великі дані (Big Data)
Перспектива моделювання даних	ER та реляційні моделі RDBMS (Реляційні СУБД)	Схема «Зірка» та моделі OLAP-кубів DW (Сховища даних)	Ключ-значення, Документні, Широкостовпчикові та Графові моделіСистеми на базі Big Data
Перспектива аналітики даних	OLTP (Онлайн-обробка транзакцій)	OLAP (Онлайн-аналітична обробка)	Множинні класи (Пакетна обробка, потокова обробка, OLTP та інтерактивні ad-hoc запити)

Бази даних широко використовуються як для особистого, так і для корпоративного використання, зокрема завдяки їхнім надійним гарантіям ACID (атомарність, узгодженість, ізоляція та довговічність) та рівню зрілості систем керування базами даних (СКБД), які їх підтримують [15] .

Процес моделювання даних може включати визначення трьох моделей даних (або схем), визначених на різних рівнях абстракції, а саме: концептуальної, логічної та фізичної моделей даних [15] [16] . На рисунку 2.1 показано частину трьох моделей даних для тематичного дослідження AMS. Усі ці моделі визначають три сутності (Людина, Студент та Викладач) та їхні основні зв'язки (асоціації викладання та керівництва).

Концептуальна модель даних. Концептуальна модель даних використовується для визначення на дуже високому та незалежному від

платформи рівні абстракції сутностей або концепцій, які представляють дані проблемної області, та їхніх зв'язків . Вона залишає додаткові деталі про сутності (такі як їхні атрибути, типи або первинні ключі) для наступних кроків. Ця модель зазвичай використовується для дослідження концепцій предметної області із зацікавленими сторонами та може бути пропущена або використана замість логічної моделі даних.

Логічна модель даних [27]. Логічна модель даних є вдосконаленням попередньої концептуальної моделі. Вона детально описує сутності домену та їхні зв'язки, але також знаходиться на платформо-незалежному рівні. Вона відображає всі атрибути, що характеризують кожну сутність (можливо, також включаючи її унікальний ідентифікатор, первинний ключ), та всі зв'язки між сутностями (можливо, включаючи ключі, що ідентифікують ці зв'язки, зовнішні ключі). Незважаючи на те, що ця модель незалежність від будь-якої СУБД, її можна легко відобразити на фізичну модель даних завдяки деталізації, яку вона надає.

Фізична модель даних. Фізична модель даних візуально представляє структуру даних, реалізовану певним класом СУБД. Таким чином, сутності представлені у вигляді таблиць, атрибути представлені у вигляді стовпців таблиці та мають заданий тип даних, який може змінюватися залежно від обраної СУБД, а зв'язки між кожною таблицею визначаються за допомогою зовнішніх ключів. На відміну від попередніх моделей, ця модель, як правило, є платформоспецифічною, оскільки вона відображає схему бази даних і, як наслідок, деякі аспекти, специфічні для платформи (наприклад, типи даних, специфічні для бази даних, або розширення мови запитів).

Підсумовуючи, складність та деталізація зростають від концептуальної до фізичної моделі даних. По-перше, важливо сприймати на вищому рівні абстракції сутності даних та їхні зв'язки за допомогою концептуальної моделі даних. Потім, основна увага приділяється деталізації цих сутностей, не турбуючись про деталі реалізації, за допомогою логічної моделі даних. Нарешті, фізична модель даних дозволяє представити, як дані підтримуються даною

СУБД [15] [16] .

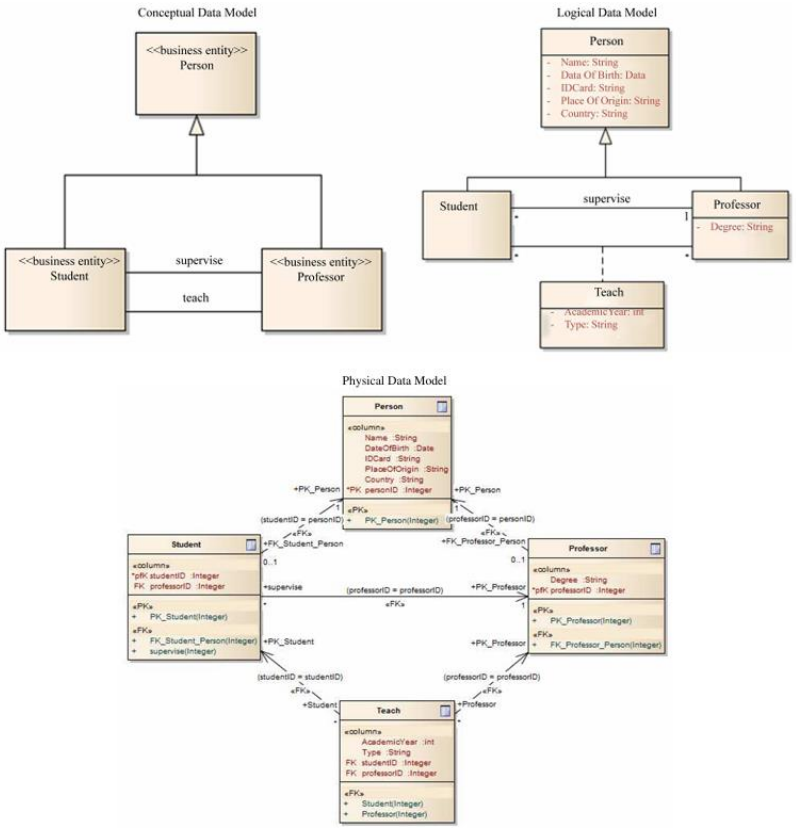


Рисунок 2.1 - Приклад трьох моделей даних (на різних рівнях абстракції) для системи управління навчальним процесом.

Бази даних отримали значний поштовх завдяки популярності реляційної моделі. Реляційна модель подолала проблеми попередніх моделей даних (а саме ієрархічної моделі та навігаційної моделі [18]). Реляційна модель спричинила появу реляційних систем керування базами даних (РСБД), які є найбільш використовуваними та популярними СУБД, а також визначення структурованої мови запитів (SQL) [19] як стандартної мови для визначення та маніпулювання даними в РСБД. РСБД широко використовуються для зберігання даних щоденних операцій. Щодо моделювання даних операційних баз даних, існує дві основні моделі: реляційна та сутність-зв'язок (ER) моделі.

Реляційна модель. Реляційна модель базується на математичній концепції відношення. Відношення визначається як множина (у математичній термінології) та представлена у вигляді таблиці, яка є матрицею стовпців та рядків, що містить інформацію про сутності домену та зв'язки між ними. Кожен стовпець таблиці

відповідає атрибуту сутності та визначає ім'я атрибута та його тип (відомий як домен).

Таблиця (відома як кортеж) відповідає одному елементу представленої сутності домену. У реляційній моделі кожен рядок унікальний, і тому таблиця має атрибут або набір атрибутів, відомий як первинний ключ , який використовується для однозначної ідентифікації цих рядків. Таблиці пов'язані одна з одною, використовуючи один або кілька спільних атрибутів. Ці атрибути відповідають первинному ключу в таблиці, на яку посилаються (батьківській), і відомі як зовнішні ключі в таблиці, що посилається (дочірній). У зв'язках «один до багатьох » таблиця, на яку посилаються, відповідає сутності сторони «один» зв'язку, а таблиця, що посилається, відповідає сутності сторони «багато». У зв'язках «багато до багатьох » використовується додаткова таблиця асоціацій, яка пов'язує залучені сутності через їхні відповідні первинні ключі. Реляційна модель також містить концепцію View (Вигляд), яка подібна до таблиці, рядки якої явно не зберігаються в базі даних, а обчислюються за потреби з визначення подання. Натомість, подання визначається як запит до однієї або кількох базових таблиць або інших представлень [17] .

Модель «сутність-зв'язок» (ER). Модель «сутність-зв'язок» (ER) [20] , запропонована Ченом у 1976 році, з'явилася як альтернатива реляційній моделі, щоб забезпечити більше виразності та семантики проектування бази даних з точки зору користувача. ER-модель – це семантична модель даних, *тобто* спрямована на представлення значення даних, що стосуються певної області. Ця модель спочатку визначалася трьома основними поняттями: сутності, зв'язки та атрибути [28]. Сутність відповідає об'єкту в реальному світі, який відрізняється від усіх інших об'єктів і характеризується набором атрибутів. Кожен атрибут має діапазон можливих значень, відомий як його область, і кожна сутність має своє власне значення для кожного атрибута. Подібно до реляційної моделі, набір атрибутів, що ідентифікують сутність, відомий як її первинний ключ.

Сутності можуть розглядатися як іменники та відповідати таблицям

реляційної моделі. У свою чергу, зв'язок – це асоціація, встановлена між двома або більше сутностями. Зв'язок можна розглядати як *дієслово* , і він включає ролі кожної сутності-учасника з обмеженнями кратності та їхньої потужності. Наприклад, зв'язок може бути один-до-одного (1:1), один-до-багатьох (1:M) або багато-до-багатьох (M:N). На ER-діаграмі сутності зазвичай представлені у вигляді прямокутників, атрибути – у вигляді кіл, з'єднаних із сутностями або зв'язками лінією, а зв'язки – у вигляді ромбів, з'єднаних з проміжними сутностями лінією.

Розширена модель ER [21] запропонувала додаткові концепції для представлення складніших вимог, таких як узагальнення, спеціалізація, агрегація та композиція. Інші популярні варіанти нотацій ER-діаграм - це діаграма Воронині ноги, Бахмана, Баркера, IDEF1X та UML-профіль для моделювання даних [22] .

Еволюція реляційних баз даних до баз даних підтримки рішень, які далі нечітко називаються « сховищами даних » (СД) , відбулася з необхідністю зберігання операційних, а також історичних даних, а також аналізу цих даних у складних інформаційних панелях та звітах. Хоча СД здається реляційною базою даних, вона відрізняється тим, що СД більше підходять для підтримки операцій запитів та аналізу (швидке читання) замість операцій обробки транзакцій (швидке читання та запис). СД містять історичні дані, що надходять з транзакційних даних, але вони також можуть включати інші джерела даних [29]. СД в основному використовуються для операцій OLAP (онлайн-аналітичної обробки) . OLAP – це підхід до надання даних звітів із СД через багатовимірні запити, і він вимагає створення багатовимірної бази даних [24] .

Зазвичай, сховища даних містять фреймворк, який дозволяє витягувати дані з кількох джерел даних та перетворювати їх перед завантаженням у репозиторій, відомий як фреймворк ETL (Extract Transform Load) [23] .

Моделювання даних у DW полягає у визначенні таблиць фактів з кількома таблицями вимірів, що пропонує моделі даних у вигляді схеми «зірка» або «сніжинка » [23] . Зіркова схема має центральну таблицю фактів, пов'язану з

таблицями вимірів . Зазвичай таблиця фактів має велику кількість атрибутів (у багатьох випадках денормалізованим способом), з багатьма зовнішніми ключами, які є первинними ключами до таблиць вимірів. Таблиці вимірів представляють характеристики, що описують таблицю фактів [30]. Коли зіркові схеми стають занадто складними для ефективного запиту, вони перетворюються на багатовимірні масиви даних, які називаються кубами OLAP (для отримання додаткової інформації про те, як виконується це перетворення, читач може звернутися до наступних посилань [24] [25]).

Зіркова схема перетворюється на куб шляхом розміщення таблиці фактів на передній грані, до якої ми звернені, а вимірів – на інших гранях куба [24] . З цієї причини куби можуть бути еквівалентними зірковим схемам за вмістом, але доступ до них здійснюється за допомогою більш платформоспецифічних мов, ніж SQL, які мають більше аналітичних можливостей (наприклад, MDX або XMLA). Куб з трьома вимірами концептуально легше візуалізувати та розуміти, але модель куба OLAP підтримує більше трьох вимірів і називається гіперкубом.

На рисунку 2.2 показано два приклади зіркових схем, що стосуються тематичного дослідження AMS. Зіркова схема ліворуч представляє модель даних для факту Студента, тоді як модель даних праворуч представляє факт Професора [31]. Обидві схеми мають центральну таблицю фактів, яка містить специфічні атрибути сутності в аналізі, а також зовнішні ключі до таблиць вимірів. Наприклад, Студент має місце походження (DIM_PLACEOFORIGIN), яке описується містом і пов'язане з країною (DIM_COUNTRY), яка має назву та код ISO. З іншого боку, на рисунку 2.3 показано кубічну модель з трьома вимірами для Студента. Ці виміри представлені сторонами куба (Студент, Країна та Дата). Цей куб корисний для виконання запитів, таких як: студенти за країною, зараховані вперше в певному році.

Проблема, з якою стикаються бази даних (DW), полягає у зростанні обсягу даних, оскільки це впливає на кількість вимірів та рівнів як у зірковій схемі, так і в кубічній ієрархії. Збільшення кількості вимірів з часом часто робить управління такими системами непрактичним; ця проблема стає ще серйознішою при роботі зі

сценаріями великих даних (Big Data), де дані генеруються безперервно [29] .

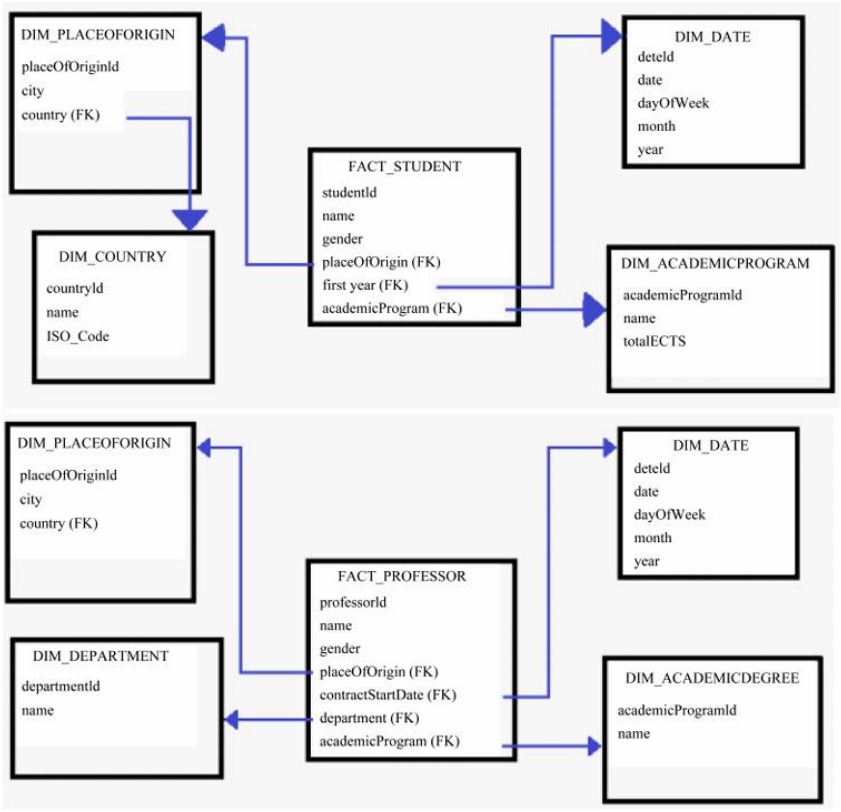


Рисунок 2.2 - Приклад двох моделей зіркової схеми для системи управління навчальним процесом.

Обсяг даних за останні роки зростає в геометричній прогресії, зокрема через одночасне зростання кількості джерел (наприклад, користувачів, систем або датчиків), які безперервно виробляють дані. Ці джерела даних створюють величезні обсяги даних зі змінними представленнями, що робить їх керування традиційними СУБД та базами даних часто непрактичним. Тому існує потреба в розробці нових моделей даних та технологій , які можуть обробляти такі великі дані.

NoSQL (не тільки SQL) [26] є одним із найпопулярніших підходів до вирішення цієї проблеми. Він складається з групи нереляційних СУБД, які, відповідно, не представляють бази даних за допомогою таблиць і зазвичай не використовують SQL для маніпулювання даними. NoSQL-системи дозволяють керувати та зберігати великомасштабні денормалізовані набори даних і

розроблені для горизонтального масштабування [32]. Вони досягають цього, поступаючись узгодженістю на користь доступності та толерантності до розділів, згідно з теоремою Брюера CAP [27] . Таким чином, NoSQL-системи є «зрештою узгодженими», *тобто* припускають, що записи в дані зрештою поширюються з часом, але існують обмежені гарантії того, що різні користувачі читатимуть одне й те саме значення одночасно. NoSQL надає гарантії BASE (базово доступний, м'який стан та зрештою узгоджений) замість традиційних гарантій ACID, щоб значно покращити продуктивність та масштабованість.

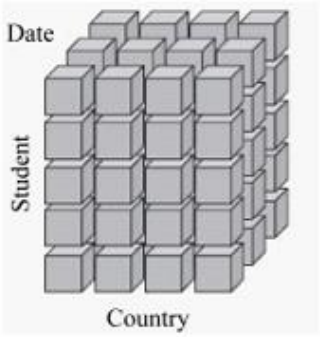
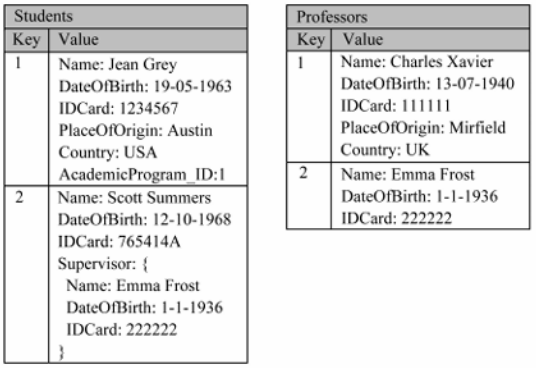


Рисунок 2.3 - Приклад кубічної моделі для системи управління навчальним процесом.

NoSQL бази даних можна класифікувати за чотирма категоріями [29] : сховища ключ-значення, (2) документи-орієнтовані бази даних, (3) сховища з широкими стовпцями та (4) графові бази даних.

Сховища пар ключ-значення. Сховище ключ-значення представляє дані як колекцію (відому як словник або карта) пар ключ-значення. Кожен *ключ* складається з унікального буквено-цифрового ідентифікатора, який працює як індекс, що використовується для доступу до відповідного значення. *Значеннями* можуть бути прості текстові рядки або складніші структури, такі як масиви. Модель ключ-значення може бути розширена до впорядкованої моделі, ключі якої зберігаються в лексикографічному порядку. Той факт, що це проста модель даних, робить сховища ключ-значення ідеально підходящими для дуже швидкого, доступного та масштабованого отримання інформації [33]. Наприклад, Amazon

широко використовує систему сховищ ключ-значення під назвою Dynamo для керування продуктами у своєму кошику для покупок [30] . Dynamo від Amazon та Voldemort, який використовується Linkedin, є двома прикладами систем, які успішно застосовують цю модель даних. Приклад сховища ключ-значення як для студентів, так і для викладачів Системи академічного управління показано на рисунку 2.4 .



Students	
Key	Value
1	Name: Jean Grey DateOfBirth: 19-05-1963 IDCard: 1234567 PlaceOfOrigin: Austin Country: USA AcademicProgram_ID:1

2	Name: Scott Summers
	DateOfBirth: 12-10-1968
	IDCard: 765414A
	Supervisor: {
	Name: Emma Frost
	DateOfBirth: 1-1-1936
	IDCard: 222222
	}

Professors	
Key	Value
1	Name: Charles Xavier DateOfBirth: 13-07-1940 IDCard: 111111 PlaceOfOrigin: Mirfield Country: UK
2	Name: Emma Frost DateOfBirth: 1-1-1936 IDCard: 222222

Рисунок 2.4 - Приклад сховища «ключ-значення» для системи управління навчальним процесом.

Документоорієнтовані бази даних. Документоорієнтовані бази даних (або сховища документів) спочатку створювалися для зберігання традиційних документів, таких як текстовий файл блокнота або документ Microsoft Word. Однак їхня концепція документа виходить за рамки цього, і документ може бути будь-яким об'єктом домену [26] . Документи містять закодовані дані у стандартному форматі, такому як XML, YAML, JSON або BSON (бінарний JSON), і однозначно ідентифікуються в базі даних унікальним ключем. Документи містять напівструктуровані дані, представлені у вигляді пар ім'я-значення, які можуть змінюватися залежно від рядка та можуть вкладатися в інші документи. На відміну від сховищ ключ-значення, ці системи підтримують вторинні індекси та дозволяють повністю здійснювати пошук за ключами або значеннями. Бази даних документів добре підходять для зберігання та управління величезними колекціями текстових документів (наприклад, текстових файлів або електронних повідомлень), а також напівструктурованих або денормалізованих даних, які

вимагали б широкого використання «нулів» у РСКБД [30]. MongoDB та CouchDB є двома найпопулярнішими документоорієнтованими системами баз даних. На рисунку 2.5 зображено дві колекції документів як для студентів, так і для викладачів Системи академічного управління.

Широкостовпцеві сховища. Широкостовпцеві сховища (також відомі як сховища сімейств стовпців, розширювані сховища записів або стовпцево-орієнтовані бази даних) представляють та керують даними як секціями стовпців, а не рядків (як у РСКБД) [34]. Кожна секція складається з пар ключ-значення, де ключі є рядками, а значення - наборами стовпців, відомими як сімейства стовпців. Кожен рядок ідентифікується первинним ключем і може мати сімейства стовпців, відмінні від інших рядків. Кожне сімейство стовпців також діє як первинний ключ набору стовпців, який воно містить. У свою чергу, кожен стовпець сімейства стовпців складається з пари ім'я-значення. Сімейства стовпців можна навіть групувати в суперсімейства стовпців [29] . Ця модель даних була дуже натхненна BigTable від Google [31]. Широкостовцеві сховища підходять для таких сценаріїв, як: (1) Розподілене зберігання даних; (2) Масштабна та пакетна обробка даних з використанням відомого методу MapReduce для таких завдань, як сортування, парсинг, запити або перетворення; та (3) Дослідницька та прогнозна аналітика. Cassandra та Hadoop HBase – це два популярні фреймворки таких систем управління даними [29] . На рисунку 2.5 зображено приклад сховища з широкими стовпцями для сутності «особа» Системи академічного управління.

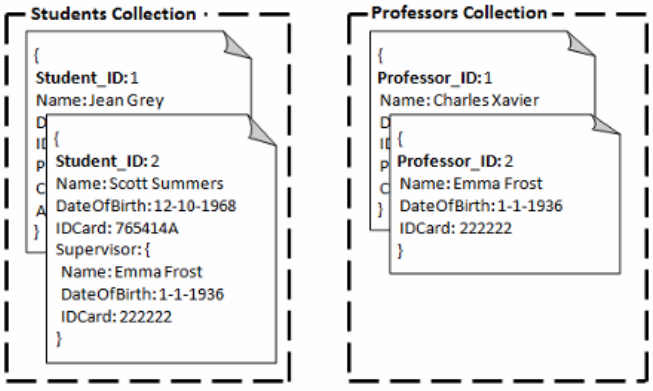


Рисунок 2.5 - Приклад документально-орієнтованої бази даних для системи управління навчальним процесом.

Графові бази даних. Графові бази даних представляють дані як мережу вузлів (що представляють сутності домену), з'єднаних ребрами (що представляють зв'язки між ними) та характеризуються властивостями, вираженими як пари ключ-значення. Графові бази даних досить корисні, коли основна увага приділяється дослідженню зв'язків між даними, наприклад, переміщенню через соціальні мережі, виявленню закономірностей або виведенню рекомендацій. Завдяки своєму візуальному представленню вони зручніші для користувача, ніж вищезгадані типи NoSQL баз даних. Neo4j та Allegro Graph - два приклади таких систем [36].

2.2 Методи аналізу та обробки даних у програмних системах

У цьому підрозділі представлено та обговорено типи операцій, які можна виконувати над моделями даних, описаними в попередньому розділі, а також проведено порівняння між ними.

Операційні бази даних

Системи, що використовують операційні бази даних, розроблені для обробки великої кількості транзакцій, які зазвичай вносять зміни до операційних даних, *тобто* даних, необхідних організації для забезпечення її щоденної нормальної роботи [35]. Ці системи називаються системами онлайн-обробки транзакцій (OLTP) , і саме вони є причиною того, що СУБД є такими важливими в наш час. СУБД все більше оптимізуються для ефективної роботи в OLTP-системах, а саме для забезпечення надійної та ефективної обробки даних [16] .

Набір операцій, що підтримуються системами управління базами даних (RDBMS), походить з реляційної алгебри та обчислення, що лежать в основі реляційної моделі [15] . Як згадувалося раніше, SQL є стандартною мовою для виконання цих операцій. SQL можна розділити на дві частини, що включають різні типи операцій: мову визначення даних (SQL-DDL) та мову маніпулювання даними (SQL-DML).

SQL-DDL дозволяє виконувати створення (CREATE), оновлення (UPDATE) та видалення (DROP) змінних-об'єкти бази даних. По-перше, це дозволяє керувати схемами, які є іменованими колекціями всіх об'єктів бази даних, пов'язаних один з одним. Потім, всередині схеми, можна керувати таблицями, вказуючи їхні стовпці та типи, первинні ключі, зовнішні ключі та обмеження. Також можна керувати представленнями, доменами та індексами. Індекс - це структура, яка пришвидшує процес доступу до одного або кількох стовпців заданої таблиці, можливо, покращуючи продуктивність запитів [15] [16].

Наприклад, розглядаючи систему управління академічним навчанням, системний менеджер може створити таблицю для зберігання інформації про студента, виконавши таку команду SQL-DDL:

Лістинг 2.1 - Таблиця для зберігання інформації про студента

```
CREATE TABLE Student (
    Student ID NOT NULL IDENTITY,
    Name VARCHAR(255) NOT NULL,
    Date of Birth DATE NOT NULL,
    ID Card VARCHAR(255) NOT NULL,
    Place of Origin VARCHAR(255),
    Country VARCHAR(255),
    PRIMARY KEY (Student ID))
```

З іншого боку, SQL-DML – це мова програмування, яка дозволяє маніпулювати об'єктами бази даних і, зокрема, витягувати з неї цінну інформацію. Найпоширенішою та найскладнішою операцією є операція SELECT, яка дозволяє користувачам запитувати дані з різних таблиць бази даних. Це потужна операція, оскільки вона здатна виконувати в одному запиті еквівалент операцій вибору, проекції та об'єднання в реляційній алгебрі. Операція SELECT повертає на вивід таблицю з результатами. За допомогою операції SELECT одночасно можна: визначити, які таблиці користувач хоче запитувати (через речення FROM), які рядки задовольняють певну умову (через речення WHERE), які стовпці повинні з'явитися в результаті (через речення SELECT), упорядкувати результат (у порядку зростання або спадання) за одним або кількома стовпцями (через речення

ORDER BY), згрупувати рядки з однаковими значеннями стовпців (через речення GROUP BY) та фільтрувати ці групи на основі певної умови (через речення HAVING). Операція SELECT також дозволяє використовувати функції агрегації, які виконують арифметичні обчислення або агрегацію даних (наприклад, підрахунок або підсумовування значень одного або кількох стовпців) [37].

Часто виникає потреба об'єднати стовпці кількох таблиць у результаті. Для цього користувач може використовувати операцію JOIN у запиті. Ця операція виконує підмножину декартового добутку між залученими таблицями, *тобто* повертає пари рядків, де відповідні стовпці в кожній таблиці мають однакове значення. Найпоширеніші запити, що використовують об'єднання, стосуються таблиць, які мають зв'язки "один до багатьох". Якщо користувач хоче включити до результату рядки, які не задовольняють умову об'єднання, він може використовувати операції зовнішнього об'єднання (ліве, праве та повне зовнішнє об'єднання). Окрім завдання запитів, DML дозволяє змінювати дані, що зберігаються в базі даних. А саме, він дозволяє додавати нові рядки до таблиці (за допомогою оператора INSERT), змінювати вміст рядків заданої таблиці (за допомогою оператора UPDATE) та видаляти рядки з таблиці (за допомогою оператора DELETE) [16] .

SQL-DML також дозволяє об'єднувати результати двох або більше запитів в одну таблицю результатів, застосовуючи операції об'єднання, перетину та виключення, засновані на теорії множин [15] .

Наприклад, розглядаючи Систему академічного управління, системний менеджер може отримати список усіх студентів з країн G8, ввівши наступний запит SQL-DML:

Лістинг 2.2 – Список усіх студентів з країн G8

```
SELECT Name, Country
FROM Student
WHERE Country in (“Canada”, “France”, “Germany”, “Italy”, “Japan”, “Russia”, “UK”, “USA”)
ORDER BY Country
```

Бази даних для підтримки прийняття рішень

Найпоширенішою моделлю даних, що використовується в DW, є куб OLAP, який пропонує набір операцій для аналізу моделі куба [23] . Оскільки дані концептуалізуються як куб з ієрархічними вимірами, його операції мають знайомі назви під час маніпулювання кубом, такі як slice, dice, drill та pivot. На рисунку 2.7 зображено ці операції з урахуванням фактів Стьюдента з тематичного дослідження AMS [38].

зрізу починається з вибору одного з вимірів (або граней) куба. Цей вимір – це той, який ми хочемо переглянути, після чого куб «розрізається» на певну глибину, що нас цікавить. Операція зрізу залишає нам більш обмежений вибір куба, а саме потрібний нам вимір (передня грань) та шар цього виміру (зрізана секція). У прикладі на рисунку 2.7 (угорі ліворуч) куб було розрізано.

Кубик – це операція, яка дозволяє обмежити передню грань куба, зменшуючи його розмір до меншої цільової області. Це означає, що користувач створює меншу «передню грань», ніж та, що була на початку. На рисунку 2.7 (угорі праворуч) показано, що набір студентів зменшився після операції з кубиком.

Деталізація – це операція, яка дозволяє переміщатися, вказуючи різні рівні вимірів, починаючи від найдетальніших (деталізація вниз) і закінчуючи найбільш узагальненими (деталізація вгору). На рисунку 2.7 (внизу ліворуч) показано деталізацію, щоб користувач міг бачити міста, звідки походять студенти країни Португалія.

Операція повороту дозволяє змінити розмір, до якого звертається (змінити поточну передню грань) на суміжну, обертаючи куб. Роблячи це, користувач отримує іншу перспективу даних, що вимагає від запитів іншої структури, але може бути кориснішим для певних запитів. Наприклад, він може розрізати куб, щоб отримати потрібні результати, але іноді за допомогою повороту більшості цих операцій можна уникнути, сприймаючи загальну структуру для майбутніх запитів та правильно повертаючи куб [23] [24] . На рисунку 2.7 (внизу праворуч) показано операцію повороту, де роки розташовані вертикально, а країни –

горизонтально.

.

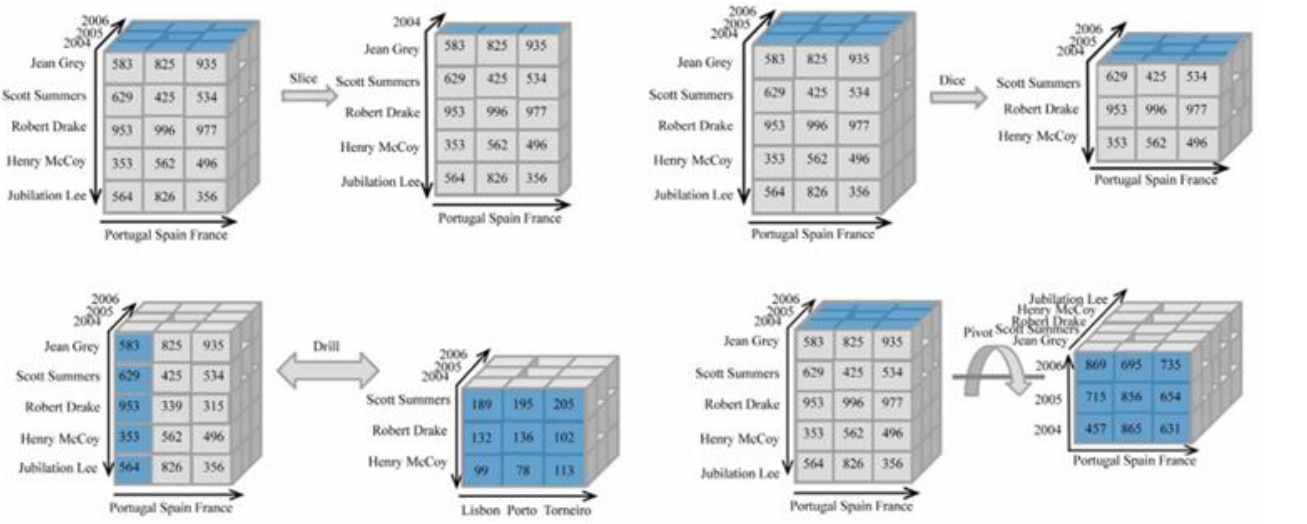


Рисунок 2.7 - Ілюстрація операцій з кубами в системі управління навчальним процесом: зріз (угорі ліворуч), розбиття на кубики (угорі праворуч), деталізація/узагальнення (внизу ліворуч) та поворот (внизу праворуч).

Звичайні операції, що виконуються над кубом OLAP, стосуються лише запитів до історичних подій, що зберігаються в ньому OLAP є MDX (багатовимірні вирази) [32] , мова запитів для баз даних OLAP, яка підтримує всі вищезгадані операції. MDX використовується виключно для аналізу та зчитування даних, оскільки він не був розроблений з урахуванням SQL-DML. Зіркова схема та куб OLAP розроблені *anpiopi* з певною метою та не можуть приймати запити, які значно відрізняються від тих, для відповіді на які вони були розроблені . Перевага цього полягає в тому, що запити набагато простіші та швидші, а за допомогою куба ще швидше виявляти закономірності, знаходити тенденції та орієнтуватися в даних, одночасно «розрізаючи та аналізуючи» [23] [25] .

Знову ж таки, розглядаючи приклад системи управління академічними науками, наступний запит представляє собою оператор вибору MDX. Речення SELECT встановлює осі запиту як ім'я та стать виміру Student та 2015 рік виміру Date. Речення FROM вказує на джерело даних, тут це куб Students, а речення WHERE визначає вісь зрізу як значення «Інформатика» виміру Academic Program

[39].

Лістинг 2.3 – Запит повертає студентів (за іменами та статтю), які зарахувалися на факультет інформатики у 2015 році.

```
SELECT
    { [Student].[Name],
      [Student].[Gender]} ON COLUMNS
    { [Date].[Academic Year] &[2015] } ON ROWS
FROM [Students Cube]
WHERE ([Academic Program].[Name] &[Computer Science])
```

2.3 Сучасні технології роботи з великими даними (Big Data)

Аналітика великих даних полягає в процесі виявлення та вилучення потенційно корисної інформації, прихованої у величезних обсягах даних (наприклад, виявлення невідомих закономірностей та кореляцій). Аналітику великих даних можна розділити на такі категорії: (1) Пакетна обробка; (2) Потокова обробка; (3) OLTP та (4) Інтерактивні спеціальні запити та аналіз.

Пакетно-орієнтована обробка – це парадигма, де великий обсяг даних спочатку зберігається, а потім аналізується, на відміну від потокової обробки. Ця парадигма дуже поширена для паралельного виконання великомасштабних повторюваних завдань, таких як розбір, сортування або підрахунок. Найпопулярнішою моделлю пакетно-орієнтованої обробки є MapReduce [5] , а точніше її реалізація з відкритим кодом у Hadoop· MapReduce базується на - парадигмі «розділяй і володарюй» (D&C), щоб розбити складні проблеми великих даних на невеликі підзадачі та обробляти їх паралельно. MapReduce, як випливає з назви, складається з двох основних функцій: Map та Reduce. Спочатку дані розділяються на невеликі фрагменти та розподіляються по мережі вузлів. Потім функція Map, яка виконує такі операції, як фільтрація або сортування, застосовується одночасно до кожного фрагмента даних, генеруючи проміжні результати. Після цього ці проміжні результати агрегуються за допомогою функції Reduce для обчислення кінцевого результату. Рисунок 2.8 ілюструє

приклад застосування MapReduce для розрахунку кількості студентів, зарахованих до певної академічної програми, за роками. Ця модель планує обчислювальні ресурси близько до місця розташування даних, що дозволяє уникнути накладних витрат на передачу даних [40]. Вона проста та широко застосовується в біоінформатиці, веб-майнінгу та машинному навчанні. Також пов'язані з середовищем Hadoop, Pig та Hive - це два фреймворки, що використовуються для вираження завдань аналізу великих наборів даних у програмах MapReduce. Pig підходить для завдань потоку даних і може створювати послідовності програм MapReduce, тоді як Hive більше підходить для підсумовування даних, запитів та аналізу. Обидві вони використовують власні SQL-подібні мови, Pig Latin та Hive QL відповідно [33] . Ці мови використовують як операції CRUD, так і ETL.

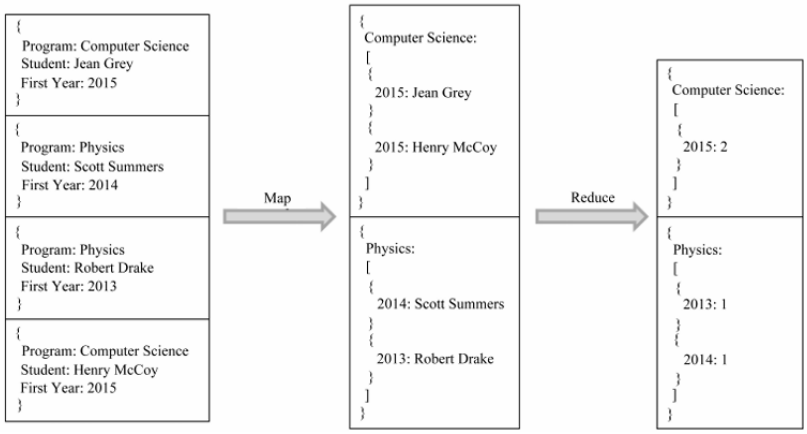


Рисунок 2.8 - Приклад застосування алгоритму MapReduce в системі управління навчальним процесом.

Потокова обробка – це парадигма, де дані безперервно надходять у потоці в режимі реального часу та аналізуються якомога швидше для отримання приблизних результатів. Вона базується на припущенні, що потенційна цінність даних залежить від їхньої актуальності. Через свій обсяг, лише частина потоку зберігається в пам'яті [33] . Парадигма потокової обробки використовується в онлайн-додатках, які потребують точності в режимі реального часу (наприклад, панелі інструментів виробничих ліній на заводі, розрахунок витрат залежно від

використання та доступних ресурсів). Вона підтримується системами управління потоками даних (DSMS), які дозволяють виконувати SQL-подібні запити (наприклад, вибір, об'єднання, групування, підрахунок) у заданому вікні даних. Це вікно встановлює період часу (на основі часу) або кількість подій (на основі тривалості) [34] . Storm та S4 – два приклади таких систем.

OLTP , як ми вже бачили раніше, в основному використовується в традиційних системах управління реляційними базами даних (RDBMS). Однак ці системи не можуть гарантувати прийнятну продуктивність, коли обсяг даних і запитів величезний, як у Facebook або Twitter. Тому було необхідно використовувати NoSQL бази даних, які дозволяють досягати дуже високої продуктивності в системах з такими великими навантаженнями. Такі системи, як Cassandra, HBase або MongoDB, є ефективними рішеннями, що використовуються в даний час. Всі вони пропонують власні мови запитів з еквівалентними операціями CRUD, ніж ті, що надаються SQL. Наприклад, у Cassandra можна створювати сімейства стовпців за допомогою CQL, у HBase можна видалити стовпець за допомогою Java, а в MongoDB вставляти документ у колекцію за допомогою JavaScript. Нижче наведено запит на JavaScript для бази даних MongoDB, еквівалентний запиту SQL-DML, представленому раніше.

Лістинг 2.3 - Запит на JavaScript для бази даних MongoDB

```
db.students.find({ country: ["Canada", "France", "Germany", "Italy", "Japan", "Russia", "UK", "USA"] }, { name: 1, country: 1 }).
sort({ country: 1 })
```

Зрештою, інтерактивні ad-hoc запити та аналіз полягають у парадигмі, яка дозволяє запитувати різні великомасштабні джерела даних та інтерфейси запитів з дуже низькою затримкою. Цей тип систем стверджує, що виконання запитів не повинно займати більше кількох секунд навіть у масштабі великих даних, щоб користувачі могли реагувати на зміни, якщо це необхідно. Найпопулярнішою з цих систем є Drill. Drill працює як рівень запитів, який перетворює запит, написаний у зрозумілому для людини синтаксисі (наприклад, SQL), на логічний план (запит, написаний платформо-незалежним способом). Потім логічний

план перетворюється на фізичний план (запит, написаний платформо-специфічним способом), який виконується в потрібних джерелах даних (наприклад, Cassandra, HBase або MongoDB).

2.4 Висновок по розділу

У другому розділі було досліджено методи та інструменти моделювання даних у програмних системах. Розглянуто основні методи моделювання даних, що використовуються для структурування та організації інформації. Проаналізовано методи аналізу та обробки даних, які забезпечують ефективне функціонування програмних систем. Окрему увагу приділено сучасним технологіям роботи з великими даними (Big Data), що дозволяють обробляти значні обсяги інформації в реальному часі. У результаті дослідження визначено сучасні підходи до побудови та управління моделями даних у програмних системах.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ МОДЕЛЕЙ ДАНИХ

3.1 Проєктування та реалізація моделі даних

Одна з ключових змін у дизайні, яку необхідно здійснити для успішного переходу від реляційної технології до використання методів і технологій NoSQL, – це трансформація існуючої моделі даних системи. Дійсно, це може бути складним завданням. Студентів-програмістів навчають моделювати світ як об'єкти, що взаємодіють один з одним за допомогою повідомлень, та думати про зв'язки між об'єктами з точки зору один-до-одного, один-до-багатьох, багато-до-одного та багато-до-багатьох. Окремий об'єкт рідко є цінним без чітко визначених зв'язків з іншими об'єктами. Цей стиль дизайну добре підходить для передачі об'єкта дозволяє розробнику програмного забезпечення моделювати об'єкти як таблиці бази даних, а зв'язки між об'єктами як первинні та зовнішні ключі, що пов'язують таблиці між собою. Ці зв'язки потім можна використовувати, надсилаючи запити через SQL. Сьогодні для розробників програмного забезпечення існує прямий, добре протоптаний шлях, щоб брати вимоги та розробляти складні моделі , які легко представляти в традиційних реляційних сховищах даних. Вимоги можна зчитувати та легко перевести в UML, який використовується для моделювання складних зв'язків та дій. Сучасні інструменти UML можуть навіть генерувати вихідний код для класів з діаграм UML. Цей код потім можна анотувати за допомогою фреймворків, таких як Hibernate, для автоматичної генерації необхідних таблиць бази даних для збереження об'єктів, не вимагаючи від розробника будь-яких знань про базову реляційну базу даних. Завдання, необхідні для перетворення довільної моделі даних з набору вимог на повноцінний рівень збереження, були чудово абстраговані. Це дуже цінні інструменти, доступні сучасним розробникам програмного забезпечення, оскільки вони дозволяють йому зосередитися на застосунку, що створюється, а не на складних деталях, необхідних для створення та управління базою даних. Зараз

стандартною практикою є використання фреймворків ORM для обробки всіх взаємодій з базою даних, що дозволяє клієнту взаємодіяти лише з об'єктами та їх зв'язками. Дійсно, рівні збереження та обслуговування Project EPIC, а також моделі, які вони використовують, базуються на цих самих передових практиках. Модель даних Project EPIC - це багатий набір звичайних старих об'єктів Java (POJO). POJO часто називають Java-бінами, що означає, що вони відповідають домовленості, згідно з якою кожен об'єкт надає набір властивостей, які матимуть аналогічні властивості.

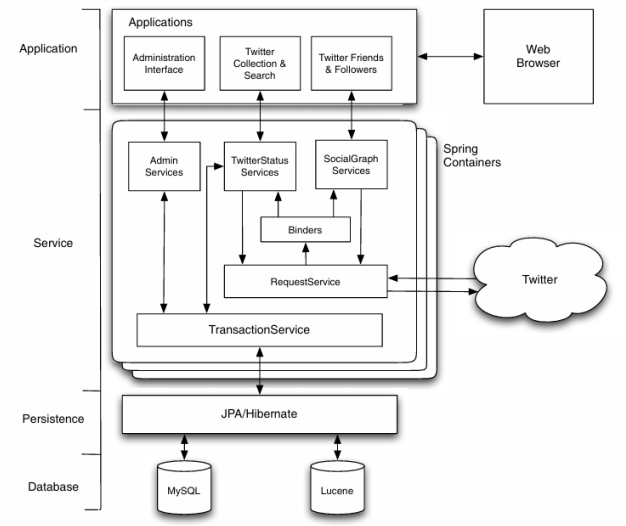


Рисунок 3.1 - Архітектура програмного забезпечення Project EPIC до впровадження технологій NoSQL

Також доступні іменовані методи отримання та встановлення. Наприклад, об'єкт Person з властивістю *name* , за домовленістю, надаватиме два методи: *getName()* та *setName()*. Це дозволяє іншим Java-фреймворкам робити припущення щодо взаємодії з цим об'єктом. Використання рефлексії під час виконання дозволяє легко отримати доступ або визначити значення будь-якої властивості об'єкта, використовуючи домовленість про Java-біни. Java Persistence API (JPA), який визначає специфікацію для автоматичного зберігання об'єктів Java, використовує ці POJO. Всі об'єкти домену Project EPIC позначені анотаціями Java 5.0, які дозволяють їх зберігати за допомогою JPA з мінімальними зусиллями з боку команди розробників. Ця технологія дозволила нашій команді перейти від

білої дошки до повноцінно функціонуючої системи зберігання за короткий проміжок часу, просто дотримуючись найкращих практик JPA. Однак - невдовзі після початкового розгортання - команді стало зрозуміло, що хоча ми змогли швидко розробити та розгорнути систему, ми не знали деталей, які спричиняють вузькі місця в продуктивності цих систем.

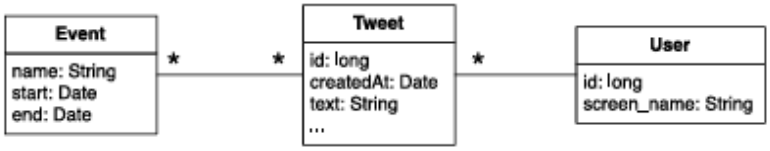


Рисунок 3.2 - Спрощена об’єктна модель проекту EPIC для збору даних із Twitter під час надзвичайної ситуації.

JPA надає розробнику простий набір анотацій для визначення способу збереження властивостей об'єктів та зв'язків між ними. Об'єкти часто автоматично виявляються за допомогою анотації *@Entity* , а таблиці бази даних, імена стовпців та типи стовпців часто створюються шляхом відображення властивостей об'єкта. Зв'язки між об'єктами можна зберігати за допомогою анотацій @OneToOne, *@OneToMany, @ManyToOne* та *@ManyToMany*. Ці анотації є надзвичайно потужними, оскільки вони дозволяють розробнику моделювати різноманітні складні зв'язки між об'єктами; неймовірно, але ці анотації здатні автоматично генерувати складні первинні ключі, зовнішні ключі та таблиці об'єднань, необхідні для моделювання цих зв'язків у базовій базі даних. Хоча ці анотації є важливим та необхідним активом, вони можуть створювати різноманітні проблеми з продуктивністю, змушуючи реляційну базу даних виконувати неефективні операції для підтримки довільної моделі, що використовується на окремому рівні програми.

Наприклад, на рис. 3.2 ми представляємо спрощену версію об'єктної моделі, яку ми використовуємо під час збору даних з Twitter під час стихійного лиха: *Користувачі генерують твіти; твіти можуть бути пов'язані з однією або кількома подіями.* Збір даних з Twitter під час масової надзвичайної події є

основним завданням інфраструктури збору даних Project EPIC. Збір має здійснюватися в режимі реального часу з мінімальними помилками або без них протягом визначеного вікна події. Твіти, зібрані під час події, залишаються пов'язаними з цією подією. Наприклад, протягом березня 2011 року Project EPIC відстежував кілька подій, включаючи землетрус у Японії; триваючі конфлікти «Арабської весни»; землетрус у Крайстчерчі, Нова Зеландія; та локальну лісову пожежу поблизу Боулдера, штат Колорадо. Через обмеження Twitter Streaming API наша інфраструктура повертала твіти, які відповідають будь-якому з 400 різних пошукових термінів. Інфраструктура повинна аналізувати кожен твіт, повернутий цим API, та пов'язувати його з відповідною (ими) подією(ями). Після завершення збору даних для події відповідні твіти можна експортувати в різні формати або перенести з робочого середовища на інші машини, щоб звільнити місце для зберігання нових подій.

Між подією (Event) та твітом (Tweet) існує зв'язок «багато-до-багатьох», а між користувачем (User) та твітом (Tweet) - «один-до-багатьох». Якщо анотувати об'єкти Java, що представляють подію (Event), твіт (Tweet) та користувача (User), анотаціями @ManyToMany та @OneToMany, об'єктно-реляційний менеджер, такий як Hibernate, автоматично створюватиме таблиці об'єднань (join) та зовнішні ключі в реляційному сховищі даних, що дозволить, наприклад, розробнику переходити від об'єкта Event до колекції, що містить усі об'єкти Tweet цієї події. Проблема полягає в тому, що Hibernate створить цю колекцію незалежно від того, чи містить вона 100 твітів, чи 75 мільйонів (розмір нашого набору даних Haiti). В останньому випадку клієнтська програма буде заблокована, оскільки Hibernate витягуватиме інформацію, необхідну для створення 75 мільйонів екземплярів класу Tweet, і зрештою аварійно завершить роботу, оскільки системі закінчиться пам'ять.

Існують й інші проблеми, які можуть виникнути під час використання технологій ORM у великих масштабах, але цей приклад ілюструє основну проблему: фреймворки ORM не можуть масштабуватися до справді великих наборів даних, оскільки зв'язки між об'єктами змусять фреймворк необов'язково

витягувати інформацію з пам'яті. Наприклад, простий акт збору нового твіту з Twitter та додавання його до існуючої події може призвести до того, що об'єктно-реляційний менеджер витягуватиме з пам'яті всі твіти, пов'язані з подією, якщо система не розроблена для захисту від такої автоматичної поведінки. Ця автоматична поведінка неймовірно корисна; на жаль, вона просто не масштабується до великих наборів даних.

Row Key 1	Column Name 1	•••	Column Name N
	Value	•••	Value
•••			
Row Key N	Column Name 1	•••	Column Name N
	Value	•••	Value

Рисунок 3.3 - Сімейство стовпців Casandra: кожен рядок відповідає потенційно різному набору стовпців.

Моделювання зв'язків є найскладнішим аспектом використання ORM-фреймворків. Як ми щойно бачили, логічні абстракції на рівні програми часто не перетворюються на ефективні представлення сховища. Це дуже ускладнює створення високодоступної та масштабованої програми з використанням цих технологій, особливо з урахуванням того, що широко визнано як хороші принципи програмної інженерії, такі як об'єктно-орієнтована евристика проектування та нормалізація даних. Ми виявили, що для забезпечення масштабованості багато з цих найкращих практик програмної інженерії необхідно застосовувати *поза рівнем персистентності*. У світі високої масштабованості дані часто реплікуються та розподіляються між сотнями або тисячами машин. *Нормалізовані, реляційні дані тут не мають місця* , і це, в свою чергу, призводить до вибору дизайну, який враховує складнощі, пов'язані з цим.

3.2 Архітектура програмної системи та інтеграція моделей дани

Через складний характер аналізу та моделювання просторових даних, реалізовані алгоритми поділяються на шість етапів процесу: генерація та

маніпулювання даними, перевірка даних, попередня обробка даних, розділення даних, моделювання та інтеграція моделі (рисунок 3.3). Оскільки не всі етапи аналізу просторових даних є необхідними в процесі просторового аналізу даних, стрілки потоку даних на рисунку 3.3 показують, які етапи попередньої обробки можна пропустити. На Рисунку також показано, як модулі пов'язані між собою, як вони використовують вихідні дані та як вони маніпулюють даними з проміжними результатами та побудованими моделями. Важливим питанням у нашому проектуванні програмного забезпечення є внутрішня організація файлів для документування результатів процесів SDAM. Документуються два типи процесів: попереднє моделювання та побудова моделі. Для збереження інформації про попереднє моделювання зберігаються такі файли:

1. Файл історії операцій, який містить інформацію про файл даних, з якого побудовано результуючий файл, виконану операцію, назву з її параметрами, а також кінцеві результуючі параметри виконаної операції.

2. Результуючий файл, що містить дані, згенеровані в результаті виконаної операції

3. Для документування процесу побудови моделі для кожної моделі зберігаються два файли:

4. Файл параметрів моделі з достатньою інформацією для збереження або перетворення моделі на інший сайт.

5. Файл інформації про модель містить всю інформацію, необхідну для опису цієї моделі користувачеві.

Функціональність програмного забезпечення включає численні функції, корисні для непросторових даних, Система призначена переважно для просторового аналізу даних. Програмна система SDAM рисунок 3.4 розроблена для підтримки, тому в цьому розділі ми зосереджуємося на просторових аспектах процесу виявлення знань у цілому. Хоча SDAM також є лізисом та моделюванням

.

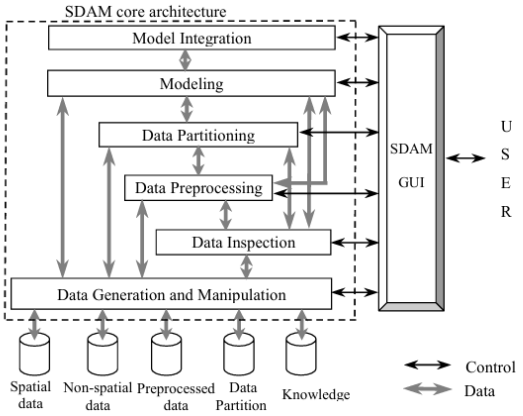


Рисунок 3.4 - Внутрішня архітектура програмного забезпечення SDAM

Генерація та обробка даних

Завантаження даних з існуючих реальних просторових баз даних у програмне забезпечення SDAM вимагає вказівки бази даних та списку атрибутів. Крім того, SDAM включає нещодавно розроблений симулятор просторових даних, який генерує шари керуючих атрибутів із просторовими властивостями, визначеними користувачем, та цільовими шарами відповідно до функцій, визначених користувачем [16]. Генератор даних може легко імітувати просторові характеристики реальних просторових наборів даних, як показано на рисунку 3.4, де було згенеровано атрибут із азотоподібною статистикою з пшеничного поля та змодельовано його вплив на мінливість врожайності. Таким чином, користувач може оцінювати та експериментувати з SDAM, використовуючи змодельовані набори даних бажаної складності та розміру.

Модуль маніпулювання даними може випадковим чином розділяти доступні дані на просторово непересічні підмножини для навчання, перевірки та тестування замість випадкових (отже, просторово випадкових) розділень, поширених у непросторових задачах інтелектуального аналізу даних (рисунок 3.5). Це дозволяє уникнути переоцінки справжніх узагальнюючих властивостей предиктора в середовищі, де атрибути часто мають високу просторову кореляцію [17].

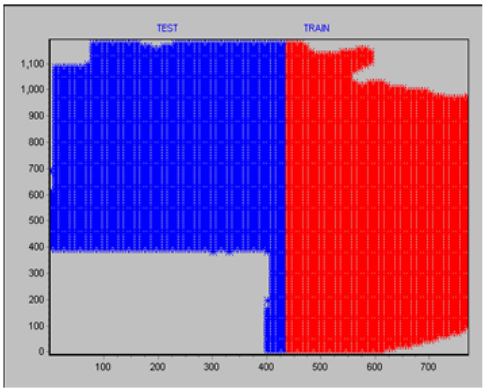


Рисунок 3.5 - Розподіл даних на навчальну та тестову підгрупи

Після завантаження даних користувач може застосовувати доступні алгоритми відповідно до послідовності за замовчуванням, запропонованої на екрані графічного інтерфейсу, або в послідовності, контрольованій користувачем.

Перевірка даних

Цей модуль включає кілька методів для надання базової та просторової статистики щодо регіону та його атрибутів. Базова статистична інформація включає параметри першого порядку (середнє значення, варіацію тощо) та стандартні показники, такі як гістограми, діаграми розсіювання між двома атрибутами, QQ-діаграми (для порівняння розподілів вибірок з нормальним розподілом, а також для порівняння двох розподілів вибірок) та кореляцію між атрибутами. Усі реалізовані операції використовують графічний інтерфейс для відображення результатів у вигляді діаграм, графіків та таблиць. Просторова статистична інформація включає графік регіону та просторову автокореляцію між точками даних у просторі атрибутів, показану за допомогою 2D та 3D перспективних фігур, а також за допомогою різних типів варіограм та корелограм [18].

Оскільки наша основна проблема пов'язана з просторовими характеристиками регіону, ми надаємо 2D та 3D графіки, які візуально показують, як атрибути змінюються в просторі (рисунок 3.6). Тривимірні перспективні графіки, включаючи контурні лінії, можна обертати, панорамувати та масштабувати, щоб спостерігати всі відповідні характеристики поверхні регіону.

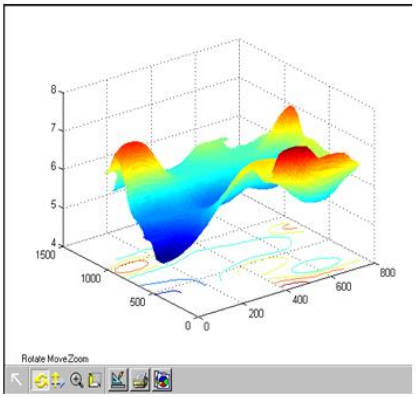


Рисунок 3.6 - 3D-перспектива шарів об’єктів

Варіограми та корелограми використовуються для характеристики просторового зв'язку між точками даних для заданого атрибута. У варіограмах визначається міра різниці між точками даних на відстані h одна від одної. Це повторюється для всіх пар точок даних, що знаходяться на відстані h одна від одної, і отримується середнє значення квадратичної різниці. Ця міра подібності називається g(h). Ці значення відображаються на графіку xy, де вісь x представляє відстань h, а вісь y - g(h) (рисунок 3.7). Програмна система спочатку будує оцінені варіограми, отримані з експериментальних даних, а потім підганяє теоретичні варіограми до оцінених (рисунок 3.7). Корелограми надають ту саму інформацію, що й варіограми, за винятком того, що в корелограмах враховується міра подібності між точками даних.

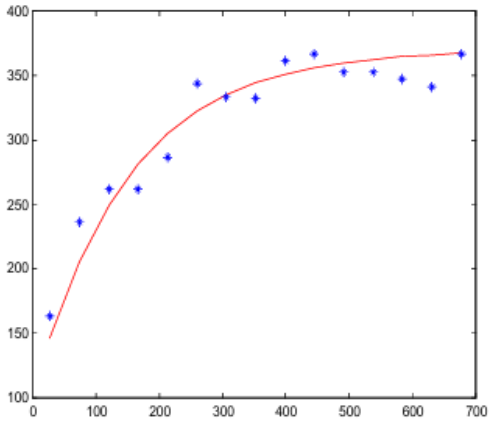


Рисунок 3.7 - Адаптація теоретичних варіограм до експериментальних даних

Попередня обробка даних

Просторові набори даних часто містять великі обсяги даних, розташованих у кількох шарах. Ці дані можуть містити помилки та можуть не бути зібрані в одному й тому ж наборі координат.

Таким чином, для підготовки даних до подальших кроків моделювання часто необхідні різні кроки попередньої обробки даних. Цей модуль містить функції:

Очищення та фільтрація даних. Через високу ймовірність шуму вимірювань, присутнього в зібраних наборах даних, існує потреба в очищенні даних. Очищення даних полягає у видаленні дублікатів точок даних, видаленні викидів значень, а також просторових викидів. Дані також можна фільтрувати або згладжувати, застосовуючи медіанний фільтр з розміром вікна, визначеним користувачем.

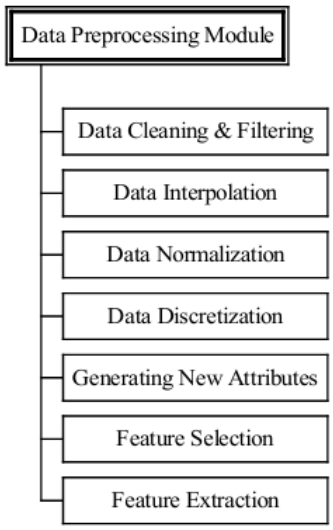


Рисунок 3.8 - Функції попередньої обробки даних

Інтерполяція даних. У багатьох реальних просторових застосуваннях роздільна здатність (кількість точок даних на область) відрізнятиметься між шарами даних, і дані не збиратимуться в одному наборі просторових місць. Тому необхідно застосовувати процедуру інтерполяції до даних, щоб змінити роздільну здатність даних та обчислити значення для одного набору місць. Можна

використовувати детерміновані методи інтерполяції, такі як обернена відстань [19] та тріангуляція [20], але вони не враховують модель просторового процесу або варіограми. Методи інтерполяції, що підходять для просторових даних, такі як кригінг [19] та інтерполяція з використанням методу мінімальної кривини [21], часто є кращими та надаються в програмній системі на додаток до звичайних методів інтерполяції, згаданих вище.

Нормалізація даних. Програмний комплекс SDAM підтримує два методи нормалізації: перетворення даних до нормального розподілу та масштабування даних до заданого діапазону.

Дискретизація даних. Цей крок необхідний у деяких методах моделювання (правила асоціації, навчання дерев рішень та всі задачі класифікації) та включає різні критерії розділення атрибутів та цілей.

Генерація нових атрибутів. Користувачі можуть генерувати нові атрибути, застосовуючи підтримувані оператори до набору існуючих атрибутів.

Вибір ознак . В областях з великою кількістю атрибутів цей крок часто корисний для зменшення простору атрибутів шляхом видалення нерелевантних атрибутів. Підтримується кілька методів відбору (прямий відбір, зворотне виключення, метод розгалужень та меж) та різні критерії (міжкласові та ймовірнісні критерії відбору) для визначення відповідної підмножини атрибутів.

Вилучення ознак. На відміну від вибору ознак, де рішення приймається на основі цілі, також підтримується зменшення розмірності на основі дисперсії шляхом вилучення ознак. Тут використовується лінійний аналіз головних компонент [22] та нелінійне зменшення розмірності за допомогою 4-шарових нейронних мереж прямого зв'язку (NN) [23]. Перетворені дані можна відобразити в d-вимірному просторі (d = 2, 3), а отримані графіки можна обертати, панорамувати та масштабувати для кращого перегляду можливих груп даних, як показано на рисунку 3.9.

Розділення даних

Розділення дозволяє користувачам розділяти набір даних на більш однорідні сегменти даних, забезпечуючи таким чином кращі результати

моделювання. У більшості задач просторового аналізу даних існують підрегіони, де точки даних мають більше схожих характеристик та більш однорідний розподіл, ніж у порівнянні з точками даних поза цими регіонами. Щоб знайти ці регіони, SDAM підтримує розділення даних відповідно до атрибутів ландшафту або цільового значення, а також використовує квадродерево для розділення просторової області вздовж її вимірів x та y на 4 підрегіони [24], як показано на рисунку 10. Він також підтримує кластеризацію на основі k-середніх та розподілу, розроблену для просторових баз даних [25], та використання ентропії та отримання інформації для розділення простору атрибутів за допомогою дерев регресії.

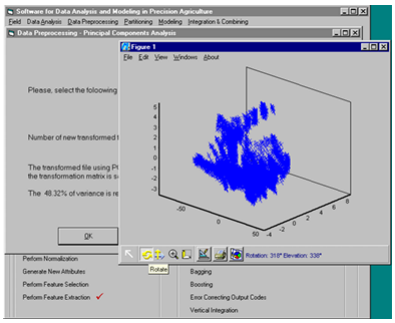


Рисунок 3.9 - Графічний інтерфейс користувача для операції попередньої обробки даних (PCA)

SDAM також підтримує наші нещодавно запропоновані методи розподілу даних для ідентифікації більш однорідних підполів на основі об'єднання кількох полів для ідентифікації набору просторових кластерів з використанням параметрів, які впливають на цільовий атрибут, але не на сам цільовий атрибут. Після цього проводиться підгонка цільових моделей прогнозування до кожного кластера в навчальній частині об'єднаних польових даних та ідентифікація на основі подібності найбільш підходящої регресійної моделі для кожної тестової точки [26]. Інший вдосконалений підхід до розподілу даних в SDAM базується на розробці послідовності локальних регресорів, кожен з яких добре відповідає певній підмножині навчальних даних, побудові моделей розподілу для ідентифікованих підмножин та їх використанні для визначення того, який

регресор є найбільш підходящим для кожної тестової точки даних [27]. Система також реалізує нашу ітеративну схему розподілу даних на основі аналізу просторово відфільтрованих помилок кількох локальних регресорів та використання статистичних тестів для визначення того, чи потрібен подальший розподіл для досягнення однорідних областей [28].

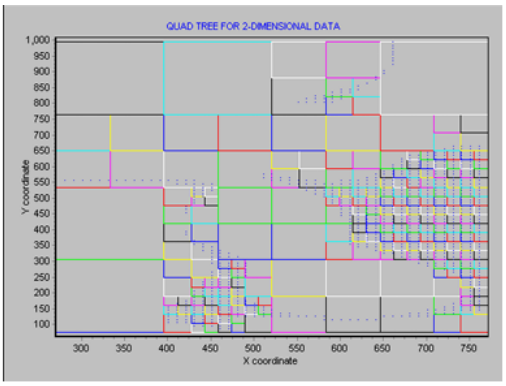


Рисунок 3.10 - Розбиття області за допомогою квадродерева

Моделювання

Цей модуль використовується для побудови моделей, що описують зв'язки між атрибутами та цільовими значеннями. Для початківців підтримується автоматична конфігурація параметрів, що використовуються в алгоритмах інтелектуального аналізу даних. Користувачеві пропонуються найбільш підходящі параметри відповідно до наборів даних та обраної моделі. Більш досвідчені експерти з інтелектуального аналізу даних можуть змінювати запропоновані параметри конфігурації та експериментувати з різними налаштуваннями алгоритму. Важливо підкреслити, що застосування методів прогнозування до просторових наборів даних вимагає інших схем розподілу, ніж простий випадковий вибір, як у стандартних методах інтелектуального аналізу даних. Тому алгоритми навчання використовують просторові набори блокової валідації під час процесу навчання [17]. Задачі моделювання поділяються на класифікаційні та регресійні, як показано на рисунку 3.11.

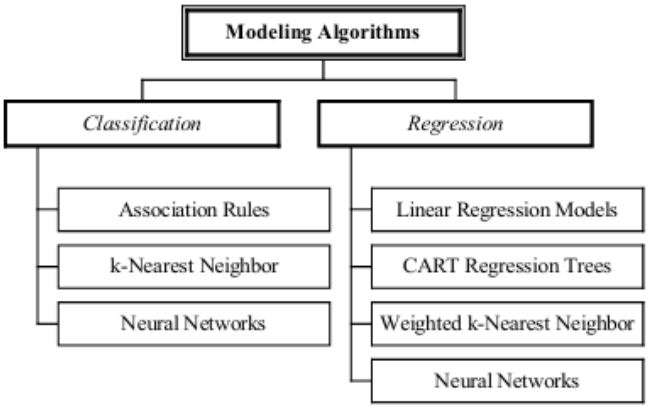


Рисунок 3.11 - Структура алгоритмів моделювання

Користувач може вибрати з кількох процедур класифікації та регресії (рис. 3.11). Для перевірки достовірності користувач може протестувати вивчені моделі прогнозування на невидимих (тестових) областях. Всі результати прогнозування відображаються графічно, як і процес навчання нейронної мережі (НМ) та вивчені структури НМ і дерев регресії. Приклад процесу навчання НМ та вивченої архітектури НМ показано на рисунку 3.12.

Інтеграція моделей

Враховуючи різні моделі прогнозування, реалізовано кілька методів підвищення точності їх прогнозування за допомогою різних схем інтегрування та об'єднання. Найпоширеніші методи інтегрування, включаючи як мажоритарний, так і зважений мажоритарний, доступні в програмній системі SDAM.

У цій роботі представлено розподілену програмну систему для аналізу та моделювання просторових даних, щоб надати інтегрований програмний пакет дослідникам та користувачам як у сфері інтелектуального аналізу даних, так і в просторових доменних застосуваннях. З точки зору фахівців з аналізу даних, численні алгоритми просторового інтелектуального аналізу даних та розширення непросторових алгоритмів підтримуються під єдиним керуванням та гнучким інтерфейсом користувача. Ми згадали кілька проблем, з якими дослідники в галузі інтелектуального аналізу даних наразі стикаються під час аналізу просторових даних, і вважаємо, що програмна система SDAM може допомогти їх вирішити. З іншого боку, методи SDAM для аналізу та моделювання просторових даних доступні експертам у предметній області для аналізу реальних просторових даних,

необхідного для розуміння впливу та важливості рушійних атрибутів і прогнозування відповідних управлінських дій.

Найважливішою перевагою програмної системи SDAM є те, що вона зберігає переваги простого в розробці та використанні графічного інтерфейсу користувача (GUI) на базі Windows, швидкого програмування в MATLAB та швидкого виконання скомпільованого коду на C та C++, що доречно для цілей інтелектуального аналізу даних. Підтримка дистанційного керування централізованою програмною системою SDAM через локальну мережу та Всесвітню мережу корисна, коли дані розташовані у віддаленому місці (наприклад, на фермі в точному землеробстві), тоді як розподілена SDAM дозволяє інтеграцію знань з даних, розташованих на кількох об'єктах.

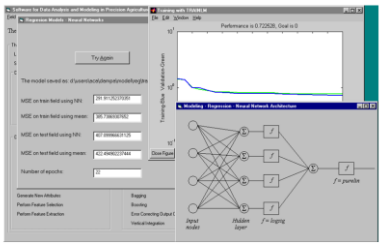


Рисунок 3.12 - Графічний інтерфейс користувача процесу SDAM для навчання нейронних мереж

Програмна система SDAM забезпечує відкритий інтерфейс, який легко розширюється для включення додаткових алгоритмів інтелектуального аналізу даних. Отже, відповідно до нашого плану досліджень та розробок, до системи поступово додаватимуться додаткові функції інтелектуального аналізу даних. Крім того, будуть розроблені більш просунуті розподілені аспекти програмної системи SDAM. Зокрема, одними з наших важливих майбутніх завдань є одночасне підключення кількох користувачів та обмін знаннями в режимі реального часу між моделями навчання в розподіленій системі.

Наша поточна експериментальна та дослідницька робота над системою SDAM спрямована на підвищення потужності та ефективності алгоритмів інтелектуального аналізу даних у дуже великих базах даних, відкриття більш

складних алгоритми моделювання просторових даних та розробка ефективних та результативних алгоритмів навчання для розподілених середовищ. Тим не менш, ми вважаємо, що попередня програмна система SDAM, описана в цій роботі, вже може бути корисною для користувачів, починаючи від фахівців з інтелектуального аналізу даних та експертів з просторових даних і закінчуючи студентами в обох галузях.

3.3 Проблеми та оптимізація моделювання даних у програмних системах

У попередньому розділі обговорювалися різні проблеми моделювання даних, з якими зіткнуться розробники програмного забезпечення, коли потреби масштабованості вимагатимуть переходу від реляційної технології до технології NoSQL. Однак цей перехід не полягає в повній заміні першої другою - звідси й розширення терміну NoSQL «не тільки SQL» - а радше в додаванні NoSQL-технологій до існуючої програмної інфраструктури, що забезпечує їй гібридну архітектуру персистентності.

Після додавання NoSQL-технологій до інфраструктури збору даних Project EPIC, його програмна архітектура тепер набуває вигляду, показаного на рис. 7. Існуючі компоненти, пов'язані зі збереженням даних - наш сервіс транзакцій, Hibernate, MySQL та Lucene - все ще присутні, і всі сервіси та програми, які раніше їх використовували, все ще функціонують з попереднім рівнем масштабованості та надійності [1]. Однак тепер на рівні збереження даних існує додатковий API під назвою Hector, який тепер доступний для будь-якого сервісу на нашому рівні. Hector - це обгортка Java для власного API Cassandra, який використовує проект Apache Thrift. Thrift обробляє взаємодію зі сервісами кластера Cassandra, а Hector забезпечує доступ до Thrift через Java API.

Архітектура програмного забезпечення Project EPIC добре підходила для переходу від реляційної персистентної архітектури до гібридної персистентної архітектури завдяки використанню фреймворку Spring для впровадження

залежностей. На рис. 1 всі сервіси, показані на рівні сервісів, мають абстрактні інтерфейси, які реалізуються певними конкретними класами. Усі взаємодії між сервісами відбуваються через абстрактні інтерфейси та покладаються на Spring для підключень конкретних реалізацій під час виконання для досягнення бажаної функціональності.

Отже, хоча це не було чітко зазначено на рис. 3.13, саме *TwitterStatusService* взаємодіє з TransactionService.

Event Name 1: Day W	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON
•••			
Event Name 1: Day X	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON
•••			
Event Name N: Day Y	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON
•••			
Event Name N: Day Z	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON

Рисунок 3.13 - Група стовпців «Події», з розбивкою за днями

MySQLTwitterStatusService, який взаємодіє з ProjectEPICTransactionService під час виконання. Ці два останні класи є конкретними реалізаціями двох попередніх абстрактних інтерфейсів. MySQLTwitterStatusService знає, як використовувати службу транзакцій та Hibernate для зберігання та доступу до твітів у MySQL. Тим часом програма Twitter Collection & Search, яка знаходиться на рівні програми, знає лише про абстрактний інтерфейс TwitterStatusService та нічого не знає про MySQL і не залежить від нього.

Завдяки цій ретельно розробленій архітектурі програмного забезпечення, перехід на Cassandra в межах інфраструктури легко здійснюється. Нам просто потрібно було створити конкретну реалізацію TwitterStatusService під назвою CassandraTwitterStatusService, яка інкапсулює знання про те, як створювати відповідні сімейства стовпців у Cassandra (через Hector) для зберігання подій, твітів та користувачів Twitter після того, як вони були отримані з Twitter за допомогою RequestService (також абстрактного інтерфейсу з кількома конкретними реалізаціями), як показано на рис. 3.13. Оскільки

CassandraTwitterStatusService прихований за абстрактним інтерфейсом TwitterStatusService, існуючий додаток Twitter Collection & Search працює *без змін* поверх цієї нової реалізації та тепер має можливість збирати та шукати значно більші набори даних, ніж раніше. Найскладнішою частиною цього переходу були проблеми моделювання даних, обговорені в попередньому розділі; фактичний перехід через абстрактну та гнучку природу нашої архітектури програмного забезпечення був простим.

Звичайно, при здійсненні цього переходу існує потреба створити та налаштувати кластер Cassandra, і це нетривіальне завдання. Однак розробники, які працюють у контексті отримання наборів даних достатнього розміру, щоб вимагати використання NoSQL-технологій, часто мають доступ до досвіду системного адміністрування та ресурсів, необхідних для придбання обладнання, налаштування кластера та встановлення відповідного програмного забезпечення. Хоча наша програмна інфраструктура може працювати на одній машині, переваги NoSQL-технологій не можуть бути по-справжньому реалізовані, доки вони не працюватимуть на значному кластері машин.

Наприклад, для популярних запитів Twitter може надсилати 50-60 твітів на секунду цілодобово через свій Streaming API. Це приблизно 5 мільйонів твітів на день. З нашою старою інфраструктурою така швидкість призводила б до заповнення наших черг на основі пам'яті тисячами необроблених твітів, оскільки Hibernate Search насилу встигав за цим. Тепер, у нашому кластері, здатність Cassandra надавати вставки за менш ніж мілісекундний час дозволяє нам обробляти 50-60 твітів на секунду без необхідності зберігати твіти в черзі в очікуванні оновлення записів нашим механізмом збереження. Тепер ми впевнені, що можемо впоратися зі швидкістю понад 100 твітів на секунду (приблизно 8,6 мільйона твітів на день), яку ми спостерігали під час збору даних.

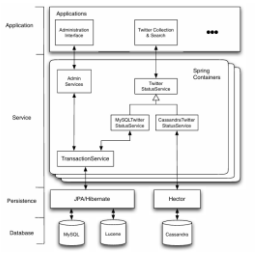


Рисунок 3.14 - Архітектура програмної інфраструктури проекту EPIC після додавання Cassandra - системи зберігання даних NoSQL.

На цій схемі опущено деталі, наведені на рис. 3.14, щоб зосередитися на двох важливих аспектах. По-перше, жоден із сервісів, які раніше залежали від нашого початкового рівня персистентності, не потребує змін. Вони можуть і надалі зберігати дані та отримувати до них доступ у

MySQL та Lucene. Другий полягає в тому, що ті сервіси, які потребують гарантій масштабованості та доступності NoSQL,

можуть додати додаткову реалізацію свого сервісного інтерфейсу, яка зберігає дані та отримує до них доступ у Cassandra. Завдяки використанню

Spring ми тепер можемо гнучко підключати реалізації сервісів, які відповідатимуть широкому діапазону вимог до масштабованості.

Дійсно, протягом кількох днів перед першим тижнем Лондонської Олімпіади 2012 року (14 днів включно) наша нова інфраструктура зібрала 40 мільйонів твітів (98,2 ГБ) на 4-вузловому кластері Cassandra, зібравши дані про 712 облікових записів користувачів та ключових слів. В якийсь момент під час цього збору наша система отримала сплеск твітів, який призвів до збільшення нашої черги в пам'яті до 40 623 твітів; після сплеску наша система очистила цю чергу менш ніж за дві хвилини, обробляючи твіти зі швидкістю 491 твіт за секунду. Ми дуже задоволені покращеннями, які забезпечує наша система на базі Cassandra під час збору справді масштабних подій «масової конвергенції».

Важливо зазначити, що завдяки дизайну нашої програмної архітектури та використанню Spring, ми маємо можливість розгортати програмну інфраструктуру Project EPIC у широкому спектрі конфігурацій: від одного дослідника , який зберігає дані Twitter у файлах JSON (у нас є сервіс, не показаний

на рис. 1 та рис. 7, який може зберігати твіти в одному файлі), до дослідницької групи, яка керує інфраструктурою на одному потужномусервері (як Project EPIC робив протягом перших двох років), та ще більшої дослідницької групи, яка використовує гібридну архітектуру збереження на великому кластері машин (як Project EPIC робить сьогодні). Використання Spring нашою програмною інфраструктурою дозволяє реалізувати кожну з цих конфігурацій простим способом шляхом редагування кількох гнучкості, оскільки її іноді не помічають. Характер роботи, яку виконує Project EPIC, є за своєю суттю багатодисциплінарним, що включає широкий спектр людей та технічних навичок. Забезпечення інфраструктури, здатної дозволити будь-якій особі, залученій до дослідницької діяльності, збирати та аналізувати відповідні дані, є значним досягненням.

Виконавши цю роботу з додавання Cassandra до рівня персистентності нашої програмної інфраструктури, ми отримуємо значні можливості для просування дослідницьких цілей проекту EPIC, особливо щодо аналізу в режимі реального часу. До цього моменту навіть надання простої статистики, такої як кількість точок даних, доступних у наших наборах даних, виявлялося трудомістким та проблематичним

Зокрема, перехід на Cassandra тепер дозволяє нам розробляти сервіси, які паралельно аналізуватимуть та індексуватимуть наші набори даних. Наприклад, існує реалізація Lucene, побудована на Cassandra. У міру збору додаткових наборів даних ми зможемо використовувати цей варіант Lucene, щоб забезпечити актуальність нашого індексу, навіть коли загальна кількість твітів сягатиме мільярдів. Lucene також дозволяє застосовувати різноманітні методи пошуку інформації до наших даних у великих масштабах. Використання Lucene для генерації векторних представлень наших даних виявилося цінним у застосуванні метрик подібності до твітів, що дозволяє легко ідентифікувати ретвіти та досліджувати простір пошуку .

Зрештою, для Cassandra розробляються аналітичні інструменти, які тепер доступні для використання у досягненні дослідницьких цілей Project EPIC.

Наприклад, DataStax розробила продукт, який поєднує Hive, Hadoop та Cassandra таким чином, що Hadoop працює безпосередньо з Cassandra, ефективно імітуючи рідну файлову систему Hadoop HDFS. Операції MapReduce Hadoop тепер можна застосовувати безпосередньо до даних, що зберігаються в сімействах стовпців Cassandra, що дозволяє використовувати інші фреймворки, сумісні з Hadoop, для масштабування.

Ми використовуватимемо проєкт Apache Mahout для застосування - алгоритмів розподіленого машинного навчання та інтелектуального аналізу даних до наших великих наборів даних, що забезпечить різноманітні можливості кластеризації та класифікації. Проєкт Apache Hive дозволяє використовувати мову, подібну до SQL, під назвою QL, яка може безпосередньо взаємодіяти з даними, що зберігаються в Hadoop. Завдяки використанню DataStax, Hadoop і, транзитивно, Hive можуть отримати доступ до даних, що зберігаються в Cassandra, що повертає деякі переваги роботи зі структурованою мовою запитів, втрачені під час переходу на NoSQL. Hive дозволяє нашій дослідницькій команді переглядати та зберігати наші дані новими та цікавими способами, навіть не вимагаючи знання нашого сервісного рівня, перетворюючи кожен запит на набір завдань MapReduce, які виконуються паралельно в кластері Cassandra. Це може значно покращити час виконання складних запитів, результати яких можна зберігати та надавати застосункам через наш сервісний рівень.

Ці функції дозволять нам забезпечити експериментальний інтерфейс перегляду даних, що зберігаються в нашому кластері Cassandra нашим сервісним рівнем. Набір сервісів, створених для підтримки цього інтерфейсу, потім може бути використаний дослідниками Project EPIC для розуміння типів інформації, яку можна витягти з наших наборів даних; це, у свою чергу, допоможе в розробці та впровадженні нових сервісів і програм, які будуть безпосередньо корисними для досліджень наших співробітників з NLP та HCC у Project EPIC.

Розробники програмного забезпечення все частіше стикаються з - ситуаціями розробки, в яких легко збирати великі обсяги даних. Хоча технології NoSQL забезпечують засоби масштабування, що виходять за межі можливостей

реляційних баз даних, вони створюють безліч проблем моделювання даних, які ускладнюють для розробників розуміння того, як найкраще перенести дані, що раніше зберігалися за допомогою реляційної схеми, до світу NoSQL без схеми. Крім того, платформи NoSQL не призначені для заміни реляційних баз даних, що створює тиск на розробників програмного забезпечення, щоб вони створювали програмні інфраструктури, які використовують гібридні архітектури персистенції, що містять обидва типи технологій. Без правильного підходу до архітектури програмного забезпечення ці гібридні архітектури важко досягти таким чином, щоб результуюча інфраструктура була придатною для обслуговування та легкою для розвитку. У цій роботі ми представляємо підхід, який Project EPIC застосував для вирішення значних проблем моделювання даних, що виникають під час переходу від реляційного підходу до підходу NoSQL, а також проблем архітектури програмного забезпечення, пов'язаних зі створенням гнучкої та розширюваної програмної інфраструктури. Наша інфраструктура збору даних тепер може легко масштабуватися для обробки дедалі більших наборів даних, які ми збираємо та аналізуємо під час кризи для підтримки нашої програми досліджень кризової інформатики.

3.4 Висновок по розділу

У третьому розділі було розглянуто реалізацію та оцінку ефективності моделей даних у програмних системах. Проведено проєктування та реалізацію моделі даних відповідно до вимог програмної системи. Досліджено архітектурні особливості інтеграції моделей даних у програмне середовище та їх взаємодію з іншими компонентами системи. Також проаналізовано проблеми, що виникають під час моделювання даних, і розглянуто методи їх оптимізації для підвищення продуктивності та ефективності роботи системи. Отримані результати підтверджують важливість якісного моделювання даних для забезпечення надійності та масштабованості сучасних програмних рішень.

ВИСНОВКИ

У ході виконання бакалаврської роботи було проведено комплексне дослідження методів моделювання даних у програмних системах, що є важливою складовою створення ефективного, масштабованого та надійного програмного забезпечення. У роботі проаналізовано існуючі підходи до моделювання даних, досліджено особливості реляційних, об'єктно-орієнтованих та нереляційних моделей, а також визначено їх переваги та обмеження в контексті сучасних інформаційних систем. Процес дослідження охопив аналіз предметної області, побудову моделей даних, проектування архітектури системи, оцінку ефективності рішень та формування практичних рекомендацій щодо їх використання.

На початковому етапі було виконано аналіз вимог, що дозволило визначити ключові функціональні та нефункціональні характеристики програмної системи. Особлива увага приділялася вимогам до цілісності даних, продуктивності, масштабованості та надійності. Визначення базових понять і термінів у сфері баз даних, а також дослідження реляційної моделі даних дозволили сформувати теоретичну основу для подальшого моделювання. Розробка концептуальних, логічних і фізичних моделей даних дала можливість структурувати інформаційні потоки та забезпечити узгодженість між компонентами системи.

У другому розділі було досліджено сучасні методи моделювання та аналізу даних, включаючи використання інструментів для побудови ER-діаграм, нормалізації даних і оптимізації запитів. Також було розглянуто технології обробки великих даних, що дозволяють працювати з розподіленими системами та значними обсягами інформації. Порівняльний аналіз різних підходів дав змогу визначити оптимальні рішення для конкретних умов застосування та обґрунтувати вибір відповідних технологій.

Реалізація моделі даних передбачала її інтеграцію в архітектуру програмного забезпечення, що забезпечило ефективну взаємодію між рівнями системи. Було враховано принципи модульності, масштабованості та

розширюваності, що дозволило створити гнучке та адаптивне рішення. У процесі реалізації було виявлено ряд проблем, пов’язаних із надлишковістю даних, складністю підтримки та забезпеченням узгодженості в розподілених середовищах, що вимагало застосування відповідних методів оптимізації та контролю.

Проведене оцінювання ефективності розроблених рішень показало, що використання сучасних методів моделювання даних дозволяє значно підвищити продуктивність системи, зменшити витрати на її підтримку та забезпечити високу якість обробки даних. Використання відповідних моделей даних залежно від типу задачі (транзакційні системи, аналітичні системи, системи великих даних) є ключовим фактором успішної реалізації програмного забезпечення.

Загалом, результати роботи підтверджують, що правильний вибір і застосування методів моделювання даних є критично важливими для створення ефективних програмних систем. Перевагами запропонованого підходу є гнучкість, можливість масштабування та адаптації до змінних умов. Водночас існують певні обмеження, зокрема складність проектування складних моделей, необхідність глибокого аналізу предметної області та потреба в оптимізації при роботі з великими обсягами даних. У подальшому можливим напрямом розвитку є впровадження інтелектуальних методів аналізу даних, використання машинного навчання для оптимізації моделей та інтеграція з сучасними хмарними платформами, що дозволить підвищити ефективність і функціональність програмних систем.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Chen, P. (1976). The Entity-Relationship Model. ACM Transactions on Database Systems. https://dl.acm.org/doi/10.1145/320434.320440
2. Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. Communications of the ACM. https://dl.acm.org/doi/10.1145/362384.362685
3. Date, C. J. (2019). An Introduction to Database Systems. pearson.com
4. Elmasri, R., & Navathe, S. (2016). Fundamentals of Database Systems. pearson.com
5. Silberschatz, A., Korth, H., & Sudarshan, S. (2020). Database System Concepts. mcgraw-hill.com
6. Fowler, M. (2002). Patterns of Enterprise Application Architecture. https://martinfowler.com/eaaCatalog/
7. Ambler,S. (2003). Agile Database Techniques. https://www.agiledata.org/
8. Kimball, R., & Ross, M. (2013). The Data Warehouse Toolkit. wiley.com
9. Inmon, W. (2005). Building the Data Warehouse. wiley.com
10. Kleppmann, M. (2017). Designing Data-Intensive Applications. oreilly.com
11. Stonebraker, M. (2018). The Case for Shared Nothing. Communications of the ACM.
12. MongoDB Documentation. (2025). https://www.mongodb.com/docs/
13. PostgreSQL Documentation. (2025). https://www.postgresql.org/docs/
14. MySQL Documentation. (2025). https://dev.mysql.com/doc/
15. Microsoft SQL Server Documentation. (2025). https://learn.microsoft.com/en-us/sql/
16. Oracle Database Documentation. (2025). https://docs.oracle.com/en/database/
17. Apache Hadoop Documentation. (2025). https://hadoop.apache.org/docs/
18. Apache Spark Documentation. (2025). https://spark.apache.org/docs/latest/
19. NoSQL Databases Explained. (2024). mongodb.com/nosql-explained

20. Amazon DynamoDB Documentation. (2025). https://docs.aws.amazon.com/dynamodb/

21. Google BigQuery Documentation. (2025). https://cloud.google.com/bigquery/docs

22. Azure Data Factory Documentation. (2025). https://learn.microsoft.com/en-us/azure/data-factory/

23. Data Modeling Guide. (2024). IBM Documentation. https://www.ibm.com/docs/en/data-modeling

24. UML Specification. (2023). omg.org/spec/UML

25. ER Diagram Tutorial. (2024). lucidchart.com/pages/er-diagrams

26. Data Normalization Guide. (2024). geeksforgeeks.org/dbms-normalization/

27. Database Indexing Techniques. (2023). tutorials point. https://www.tutorialspoint.com/dbms/dbms_indexing.htm

28. Big Data Analytics Overview. (2024). dataversity.net

29. Data Warehousing Concepts. (2024). snowflake.com/guides/data-warehouse

30. Graph Databases. (2025). neo4j.com/docs/

31. Neo4j Documentation. (2025). https://neo4j.com/docs/

32. Cassandra Documentation. (2025). https://cassandra.apache.org/doc/latest/

33. Redis Documentation. (2025). https://redis.io/docs/

34. Data Modeling Best Practices. (2024). microsoft.com/learn

35. Cloud Data Architecture. (2025). aws.amazon.com/architecture/

36. Event-Driven Architecture Guide. (2024). confluent.io/learn/event-driven-architecture/

37. Data Governance Framework. (2024). dataversity.net/data-governance/

38. OLAP vs OLTP Systems. (2023). oracle.com/database/olap-vs-oltp/

39. Data Integration Techniques. (2024). talend.com/resources/data-integration/

40. Machine Learning and Data Modeling. (2025). tensorflow.org/resources

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: "Методи моделювання даних у програмних системах "

Обсяг пояснювальної записки: 75 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____