Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів підтримки темпоральності в реляційних базах даних
(тема)

Виконав:
Випускник 2 курсу, групи ІПЗм-19-2
Михневич Т.К.
(прізвище,ініціали)

Спеціальність 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

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

Керівник доц. Мазурова О.О.
(посада,прізвище)

Допускається до захисту
Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2021р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

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

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
« 26 » березня 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Михневич Тетяни Костянтинівни
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів підтримки темпоральності в реляційних базах даних
затверджена наказом університету від 26.03.2021 № 385
2. Термін подання роботи до екзаменаційної комісії 09 травня 2021р.
3. Вихідні дані до роботи *електронні ресурси за обраною тематикою, порівняльний аналіз підтримки темпоральності у РСУБД Oracle, MS SQL Server, середовища розробки SQL Server Management Studio, SQL Developer, Visual Studio, мови PL/SQL, T-SQL, C#.*
4. Перелік питань, що потрібно опрацювати в роботі *аналіз предметної області і постановка задачі, огляд методів підтримки темпоральності у РСУБД, розробка математичної моделі для роботи з темпоральними даними у РСУБД, програмна реалізація методів підтримки темпоральності, проведення експерименту, формування рекомендацій.*
5. Перелік графічного матеріалу із зазначенням креслеників, схем, слайдів, ілюстрацій *актуальність області дослідження, мета, постановка задачі, аналіз проблемної області, моделі з різними вимірами часу, розширена темпоральна модель, етапи проектування темпоральної БД, аналіз методів підтримки темпоральності, вибір СУБД, планування експерименту, фізична модель БД, програмна реалізація, результати, рекомендації, висновки.*

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доц. Мазурова О.О.		07.05.21

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз проблемної області дослідження	25.01.21 – 11.02.21	виконано
2	Аналіз аналогів	7.02.21 – 11.02.21	виконано
3	Розробка постановки задачі	11.02.21 – 14.02.21	виконано
4	Дослідження існуючих моделей та методів підтримки темпоральності	14.02.21 – 20.02.21	виконано
5	Моделювання предметної області	15.02.21 – 21.02.21	виконано
6	Планування експериментальної частини дослідження	20.02.21 – 05.03.21	виконано
7	Розробка методів підтримки темпоральності	05.03.21 – 25.03.21	виконано
8	Програмна реалізація засобів для експерименту	25.03.21 – 10.04.21	виконано
9	Проведення експериментів	05.04.21 – 14.04.21	виконано
10	Оформлення статті	10.04.21 – 21.04.21	виконано
11	Підготовка пояснювальної записки	01.04.21 – 30.04.21	виконано
12	Підготовка презентації та доповіді	30.04.21 – 03.05.21	виконано
13	Нормоконтроль	02.05.21 – 10.05.21	виконано
14	Рецензування	02.05.21 – 10.05.21	виконано
15	Занесення диплома в електронний архів	07.05.21	виконано
16	Попередній захист	07.05.21	виконано
17	Допуск до захисту у зав. кафедри	11.05.21	виконано

Дата видачі завдання 25 січня 2021р.

Студент _____
(підпис)

Керівник роботи _____ доц. Мазурова О.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 93 с., 12 рис., 5 табл., 23 джер.

БІТЕМПОРАЛЬНА МОДЕЛЬ ДАНИХ, ДІЙСНИЙ ЧАС, РЕЛЯЦІЙНА СУБД, ТЕМПОРАЛЬНА МОДЕЛЬ ДАНИХ, ТЕМПОРАЛЬНІСТЬ, ТРАНЗАКЦІЙНИЙ ЧАС, ЧАСОВА МІТКА, SQL.

Об’єктом дослідження є темпоральні моделі даних в інформаційних системах. Предмет дослідження – методи зберігання та доступу до темпоральних даних з використанням реляційних СУБД.

Метою дослідження є аналіз існуючих рішень для організації зберігання та оперування темпоральними даними у реляційних СУБД, їх вдосконалення, створення нових методів.

Дослідження, що будуть проведені, базуватимуться на результатах моделювання процесів у інформаційних системах, булевій алгебрі, реляційному обчисленні, теорії множин.

Методи розробки базуються на наступних мовах та технологіях: SQL, T-SQL, PL / SQL, C#, MS SQL Server, Oracle.

У результаті роботи побудовано математичну модель темпоральних даних, описано методи зберігання та доступу до них, сформовані загальні рекомендації щодо організації та роботи з темпоральними моделями даних у реляційних базах даних.

BITEMPORAL DATA MODEL, CURRENT TIME, RELATIONAL DBMS, SQL, TEMPORAL DATA MODEL, TEMPORALITY, TRANSACTION TIME, TIMESTAMP.

The object of the research is temporal data models in information systems. The subject of research are methods of storage and access to temporal data using relational databases.

The purpose of the work is to analyze existing solutions for the organization of storage and operation of temporal data in relational databases, their improvement, the creation of new methods.

The research that will be conducted will be based on the results of process modeling in information systems, Boolean algebra, relational computing, set theory.

Development methods are based on the following languages and technologies: SQL, T-SQL, C #, MS SQL Server, Oracle.

As a result, mathematical models of temporal data are constructed, methods of storage and access to them are described, the general recommendations concerning the organization and work with temporal data models in relational databases are formed.

Я, Михневич Тетяна Костянтинівна, студентка гр. ІПЗм-19-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів підтримки в темпоральності в реляційних базах даних», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлена з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Кількість даних, які людство генерує щогодини, величезна. Стрімкий розвиток технологій, збільшення об’єму інформації сприяє створенню нових напрямів, таких, як робота з Big Data, але статичні дані вже не задовольняють потреб прикладних областей. «Звичайно, великі дані – це потужна лінза для погляду на наш світ. Незважаючи на свої обмеження та вимоги, аналіз великих даних може допомогти нам дізнатися багато про себе. Але якими б великими не були ці дані або які уявлення ми б не отримували з них, це все одно лише короткий знімок. Нам потрібно перестати зациклюватися лише на великих даних і почати думати про довгі дані – набори даних, що мають масштабний історичний аналіз», – так сказав вчений та письменник наших часів Самуель Арбесман.

По-перше, зараз дуже складно знайти додатки, що не зберігають та не маніпулюють даними, які змінюються з часом. Інформація про минуле може бути використана для знаходження трендів, оптимальних стратегій, зв’язків між певними подіями. Інформація про майбутнє використовується для планування та прийняття рішень. Дуже часто темпоральні моделі даних використовуються у галузі фінансових послуг – такі дані, як економічні показники, операції з банківськими рахунками, потоки кліків веб-сайтів, – усі вони є послідовністю, індексованою часом. Наприклад, системи, що існують у фінансових гігантів Goldman Sachs (SecDB), JP Morgan (Athena) та Bank of America (Quartz), були побудовані на основі бітемпоральної моделі даних. Також однією із сфер, де можуть бути застосовані темпоральні моделі даних, є системи, що призначені для рекомендацій, вони забезпечують підтримку вибору споживача, надаючи персоналізований перелік товарів та послуг. Такі системи дозволяють побудувати часову модель поведінки користувачів, що описує еволюцію вимог користувача до товарів, пропонованих системою [1, 2].

По-друге, не зважаючи на популярність NoSQL баз даних, більшість додатків все ж таки використовують реляційні бази даних [3].

По-третє, нажаль, в мові запитів SQL немає адекватної і ефективної підтримки роботи з темпоральними базами даних. Традиційна реляційна база даних зберігає інформацію лише про поточний стан, і можливості працювати з прив'язаними до певних дат або інтервалів часу даними, СУБД не надає. Тому майже у всіх базах даних підтримка роботи з темпоральними даними забезпечується зусиллями розробників програмного забезпечення, які використовують для вирішення завдань «незручні» конструкції мови. При цьому багато простих запитів, які легко сформулюються «поза часом», досить важко переписати для «саморобної» темпоральной системи, і вони виходять досить громіздкими, що загрожує появою помилок.

Отже, метою роботи є дослідження методів підтримки темпоральності в реляційних СУБД.

Об’єктом дослідження є темпоральні моделі даних в інформаційних системах. Предмет дослідження – існуючі методи зберігання та оперування темпоральними даними з використанням реляційних СУБД.

Результати проведеного дослідження може бути використано для розв’язання таких прикладних задач, як проектування темпоральних реляційних баз даних, обрання способу реалізації підтримки темпоральності (на рівні БД або додатку), додання підтримки темпоральності вже в існуючу базу даних.

За результатами кваліфікаційної роботи магістра було розроблено презентацію (див. додаток А).

Були опубліковані тези в рамках XVII науково-технічної конференції «Інноваційні технології – 2020». Тези доповіді наведено в додатку Д. Також була написана та подана до опублікування до науково-технічного журналу «Біоніка інтелекту» стаття «Дослідження методів підтримки темпоральності в реляційних базах даних» (див. додаток Д). Лістинг частини скриптів розроблюваної системи наведено в додатку Г.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області дослідження

Стандартна реляційна база даних зберігає інформацію у певній предметній галузі. СУБД дає змогу змінювати стан об’єктів, замінюючи їх старі значення новими. Запит до такої СУБД завжди дає актуальні дані, але попередні стани об’єктів до уваги не беруться. Такий підхід може бути доречним у невеликих системах, де попередній стан системи на має значення.

Тим не менш, існує велика кількість предметних галузей, де зміна кожного об’єкта є критичною, де можливість отримати стан системи на будь-який момент є не просто тривіальною задачею, а основним призначенням, де попередні значення використовуються для прогнозування майбутніх, де помилка без можливості знайти історію її появлення та відкату системи до попереднього стану може призвести до великих збитків. У таких випадках просто необхідно обробляти дані, що змінюються з часом, накопичувати історію їх зміни та видавати стан системи на будь-який момент.

Темпоральні (або часові) дані – це будь-які дані, що пов’язані з певними датами або проміжками часу. Темпоральні СУБД – це СУБД, які пристосовані для управління та зберігання таких даних [4].

Незважаючи на те, що підтримка темпоральності має свої переваги, актуальність, існування окремого напрямку досліджень і розробок, досі не існує єдиного стандарту реалізації темпоральних БД.

Майже всі додатки, що використовують реляційні бази даних, є темпоральними за своєю суттю. У базах даних зберігаються дані, що змінюються з плином часу. Але в SQL не має адекватної і ефективної підтримки роботи з темпоральними даними. Тому найчастіше за все підтримка роботи з темпоральними даними забезпечується зусиллями програмістів. Темпоральні принципи в цих системах використовуються неефективно [5].

Зазвичай побудова прототипу темпоральної СУБД заснована на обраній реляційній СУБД. Тобто часто темпоральна СУБД – це надбудова над реляційною [6, 7]. Цей спосіб є найбільш простим та доступним як для розробників, так і для користувачів, адже на користь реляційних СУБД свідчать такі фактори:

- успішно використовуються останні кілька десятиліть та зарекомендували себе надійними та випробуваними часом системами;
- мають гарну підтримку виробника, постійно вдосконалюються та розвиваються;
- є кращими для забезпечення цілісності даних;
- майже всі реляційні СУБД підтримують загальноприйняті стандарти;
- багато існуючих прикладних систем успішно та стабільно працюють з використанням реляційних СУБД.

Але розглядаючи проблеми використання реляційних СУБД з підтримкою темпоральності можна виділити наступні проблеми:

- відсутність єдиного стандарту для реалізації підтримки темпоральності. Кожна СУБД має свій набір функцій, рівень вбудованої підтримки темпоральності, особливості реалізації. Неможливо, зробивши підтримку темпоральності в одній СУБД, перенести її повністю в іншу;
- реляційна СУБД не має можливості повністю забезпечити контроль коректності значень часового поля;
- мови SQL не пристосовані для роботи з темпоральними даними [8].

Отже, враховуючи перелічені фактори, можна зробити висновок, що реалізація темпоральних СУБД на базі реляційних є найбільш ефективною на сьогоднішній момент, але існує низка проблем, для вирішення яких необхідно провести глибоку переробку моделі даних, треба мати фундаментальні знання темпоральної технології, а також особливостей обраних для реалізації СУБД. Саме тому було прийнято рішення дослідити різні варіанти підтримки темпоральності з використанням реляційних СУБД.

До 1986 року групи стандартів ANSI та ISO офіційно прийняли стандартне визначення мови «Мова бази даних SQL». Версія SQL, в якій було додана

підтримка темпоральності, це стандарт SQL:2011 [9]. Найважливіша нова функціональність SQL:2011 - це можливість створювати та керувати темпоральними таблицями.

Основні характеристики, що забезпечують підтримку темпоральності:

- специфікація періодів;
- таблиці з дійсним / транзакційним часом;
- темпоральна поведінка UPDATE/DELETE:

```
DELETE Emp FOR PORTION OF EPeriod
FROM DATE '2011' TO DATE '2013'
WHERE EName = 'Anton';
```

- темпоральні обмеження цілісності:

```
ALTER TABLE Emp
ADD PRIMARY KEY (EName, EPeriod WITHOUT OVERLAPS);
ALTER TABLE Emp
ADD FOREIGN KEY (EDept, PERIOD EPeriod)
REFERENCES (DName, PERIOD DPeriod);
```

- предикати та функції для періодів.

Але у стандарті є і певні обмеження, а саме:

- періоди - це лише відкриті-закриті інтервали [X, Y); Та це не новий тип даних, по факту це два стовпця для позначення початку та кінця періоду;
- у таблицях може бути щонайбільше один транзакційний і один дійсний період часу;
- немає понять «NOW», «infinity», «empty». Тільки 0001-01-01 та 9999-12-31 є «особливими» датами;
- багато відмінностей між таблицями періодів транзакційного та дійсного часу;
- обмежена підтримка формулювання запитів.

Та навіть цей стандарт не підтримують повністю реляційні СУБД, а лише частково у більшому або меншому ступені. Розглянемо деякі з них. Результати порівняння занесені у таблицю 1.1 [10-14].

Таблиця 1.1 – Підтримка темпоральності різними РСУБД

База даних	Періоди	Темпоральні ключові атрибути	Темпоральні UPDATE / DELETE	Темпоральні предикати	Темпоральні запити
MySQL	немає	немає	немає	немає	немає
MS SQL Server	для таблиць з транзакційним часом	немає	для таблиць з транзакційним часом	для типів діапазонів	немає
Postgre SQL	підтримуються	первинні підтримуються; зовнішні не підтримуються;	немає	для типів діапазонів	немає
Oracle	стовпці періодів неявні	підтримуються	виконуються з дійсним часом поточного сеансу	для типів діапазонів	немає
Teradata	підтримуються	обмеження первинного ключа перевіряються СУБД, але зовнішні обмеження не забезпечуються СУБД	підтримуються	для типів діапазонів	наступні операції не підтримуються для послідовних запитів: – outer joins; – set; – впорядковані аналітичні функції; – підзапити; – WITH, TOP n, GROUP BY, DISTINCT

Отже, класичні реляційні СУБД не орієнтовані на зберігання всіх змін об'єктів і не підтримують їх багатоверсійність. При цьому темпоральні принципи в цих системах використовуються неефективно. Це проявляється в наступних недоліках:

- неекономне використання дискового простору;
- низька ефективність взаємодії з темпоральними об'єктами;
- погана розширюваність БД;
- висока трудомісткість розробки БД;

- складність підтримки цілісності даних;
- складність організації доступу до даних;
- проблеми в забезпеченні конфіденційності даних.

Фактично, на практиці існує декілька принципових підходів до реалізації підтримки темпоральності, коли йде мова про використання реляційних СУБД:

- реалізація темпоральной підтримки на рівні бази даних за допомогою вбудованої підтримки (вибір СУБД з максимальною кількістю темпоральних функцій);
- реалізація підтримки на рівні бази даних шляхом додавання додаткових стовпців, тригерів, функцій, збережених процедур тощо (далі – підхід «на рівні БД»);
- реалізація підтримки темпоральності на рівні додатку (далі – підхід «на рівні додатку»).

Кожний з цих підходів має переваги та недоліки, передумови для використання, складності в реалізації. Тому необхідно дослідити описані підходи та сформувати рекомендації щодо їх вибору.

1.2 Постановка задачі

Метою роботи є дослідження методів підтримки темпоральності в реляційних базах даних.

Для проведення дослідження слід розробити схему СУБД з урахуванням темпоральних аспектів системи.

Також необхідно сформувати множину критеріїв для порівняння рівня темпоральності у програмних рішеннях. Виміряти такі показники, як швидкість роботи на різних об’ємах даних, швидкість та зручність реалізації, кількість зайнятого місця на дисковому просторі для збереження моделі.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- провести аналіз проблемної області побудови темпоральних інформаційних систем та методів підтримки темпоральності в реляційних БД;
- розробити нову модель та методи підтримки темпоральності в реляційних БД;
- спроектувати та реалізувати програмні рішення для проведення дослідження розроблених методів;
- спланувати та провести експериментальне дослідження розроблених методів, розробити відповідні рекомендації.

Результатом проведеного дослідження буде порівняння методів підтримки темпоральності різними підходами за множиною сформованих критеріїв, а також набір загальних рекомендацій щодо організації та роботи з темпоральними моделями даних.

2 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

2.1 Аналіз проблемної області побудови темпоральних інформаційних систем

2.1.1 Інтерпретація часу у темпоральній СУБД

Інтерпретація часу у темпоральній СУБД. Для того, щоб забезпечити роботу реляційної СУБД (РСУБД) з темпоральними даними, необхідно реалізувати три основні концепти:

- темпоральні типи даних;
- види часу;
- темпоральні твердження.

Для зберігання у СУБД неперервний час дискредитується та зберігається з певною точністю, характерною кожній СУБД. Розглянемо темпоральні типи даних, що зберігають час, їх три:

- момент часу – точка на часовій осі. Наприклад, CURRENT_DATE, 24 вересня 9 година ранку тощо. Цей тип даних існує і в звичайні моделі даних, майже кожна СУБД має оператори роботи з ним, для його зберігання достатньо одного стовпця (див. рис. 2.1);
- інтервал –довжина часу, тривалість відома, кінець і початок – ні;
- період – протяжність часу, тривалість якого відома, а початок і кінець визначаються двома моментами часу. Період може бути закритим (включені обидва моменти часу), відкрито-закритим (один момент часу виключений), відкритим (обидва моменти часу виключені). Період створює багато труднощів, необхідно підтримувати операції над множинами для часу, обмеження цілісності, адекватність моделі [15].

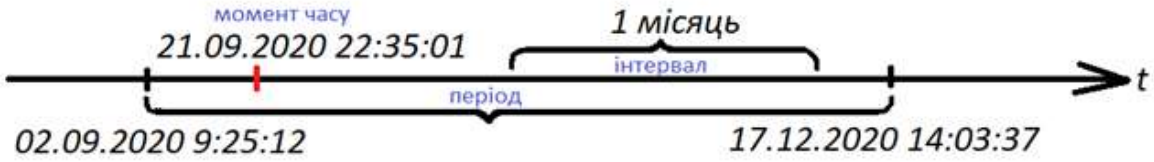


Рисунок 2.1 – Темпоральні типи даних

В ході аналізу [15] було виявлено три фундаментальних типи часу:

– час, визначений користувачем (неінтерпретований час);

– дійсний час (час, у який факт є правильним у реальності, що моделюється);

– транзакційний час (коли факт був записаний у базу даних).

В ході аналізу [15] було виявлено три базових твердження з часовим орієнтуванням:

– поточне: зараз;

– послідовне: у кожний момент часу;

– без послідовності: ігнорування часу.

Отже, три набори з трьох понять створюють основу зберігання та управління темпоральними даними. На жаль, ці поняття є чужими для мови запитів SQL. Більш того, виникають різні невідповідності та заміни, які кожен постачальник СУБД створив для рішення проблем темпоральності. Тому для реалізації темпоральних БД ці чисті, чіткі поняття повинні бути перетворені у громіздкі вирази та структури у SQL та схемах реляційних баз даних (РБД).

2.1.2 Таблиці з дійсним часом (Valid-time state tables)

Розглянемо таблиці, що містять темпоральні дані з дійсним часом. Кожен факт має час, коли він був/є/буде правильним у реальності, що моделюється. Цей факт може бути як істиною, так і ні. Навіть якщо дійсний час не зареєстрований у

базі даних, він все одно є у кожного факту. Більш того, дійсний час не залежить від того чи був він зареєстрований у базі даних и коли.

Наприклад, маємо таблицю зміни цін на товари з дійсним часом зі схемою:

– ProductID – унікальний ідентифікатор продукту;

– FromDate – дата початку дії ціни на товар;

– ToDate – дата закінчення дії ціни на товар;

– Price – ціна на товар.

Тоді дані в ній можуть виглядати так, як на рисунку 2.2:

	ProductID	FromDate	ToDate	Price
1	707	2011-05-31 00:00:00.000	2012-05-29 00:00:00.000	12.0278
2	707	2012-05-30 00:00:00.000	2013-05-29 00:00:00.000	13.8782
3	707	2013-05-30 00:00:00.000	NULL	13.0863
4	708	2011-05-31 00:00:00.000	2012-05-29 00:00:00.000	12.0278
5	708	2012-05-30 00:00:00.000	2013-05-29 00:00:00.000	13.8782
6	708	2013-05-30 00:00:00.000	NULL	13.0863
7	709	2011-05-31 00:00:00.000	2012-05-29 00:00:00.000	3.3963

Рисунок 2.2 – Таблиця з дійсним часом

Без колонок з відмітками часу можна було би сказати «ProductID є унікальним для кожного рядку даних». З відмітками часу можна сказати лише так: «У кожен момент часу ProductID є унікальним для кожного рядку даних». На жаль, конструкція «Первинний ключ» неадекватна для таблиць з дійсним часом. Це обмеження вимагає складних конструкцій для виконання вимог щодо обмежень цілісності у таблицях з дійсним часом.

2.1.3 Таблиці з транзакційним часом (Transaction-time state tables)

Для кожного факту є дата фіксації його у системі. Це і є транзакційним часом. Якщо дійсний час відображає зміни у реальності, транзакційний час

відображає зміни у стані бази даних. Важлива відміна дійсного часу від транзакційного є у внесенні змін для фактів, що зберігаються у базі даних. Якщо для дійсного часу операція UPDATE є стандартною, після її виконання кортеж зберігає оновлені дані, то для транзакційного часу операції зміни не існує. Для цього створюється новий кортеж, що містить нову інформацію (що є вірною з моменту виконання транзакції), а у старий кортеж заносить час транзакції у якості точки закінчення істинності. Тільки поточні модифікації дозволені в таблицях з транзакційним часом, тому що попередні стани неможливо змінити.

2.1.4 Бітемпоральні таблиці (Bitemporal tables)

Модель даних може не підтримувати жодного виміру (модель-знімок), підтримувати будь-який один чи обидва виміри. Бітемопральна модель даних підтримує обидва виміри (дійсний і транзакційний час) і є найбільш універсальною.

2.2 Аналіз методів підтримки темпоральності

2.2.1 Основні способи забезпечення підтримки темпоральних даних

Фактично, на практиці існує два принципових підходи до реалізації підтримки темпоральності:

– реалізація темпоральной підтримки «на рівні додатку»;

– розширення нетемпоральної моделі даних до темпоральної.

Метод реалізації темпоральності «на рівні додатку» передбачає розробку спеціальних власних інструментів підтримки темпоральності на рівні додатку. Але така реалізація може бути сильно прив’язана до певних технологій, платформ, на яких розроблюється додаток та не може бути легко перенесена на інші. Більш того, таку систему буде важко розширювати, адже темпоральна семантика відноситься до фундаментальних складових системи з підтримкою темпоральності.

Розширення нетемпоральної моделі даних до темпоральної моделі означає, що для специфікації темпоральних понять використовуються основні концепції, що підтримуються нетемпоральною моделлю даних. Для отримання можливості реалізовувати темпоральні операції з даними, алгебра і мова запитів розширюються додатковими операціями. На практиці цей підхід розширення схеми даних найбільш широко використовується для побудови систем з темпоральними моделями даних. Перевага методу розширення до темпоральної моделі полягає в тому, що змінюються лише окремі частини моделі, наприклад, мови запитів або обмежень цілісності. Метод доступу до інформації та структура даних залишаються незмінними.

В рамках даного підходу запропоновані різні моделі. Принципові відмінності цих моделей однієї від одної можна розділити за наступними критеріями:

– тип темпоральних даних (дискретне або інтервальне уявлення часу);
– забезпечення темпоральності на рівні окремих атрибутів або на рівні кортежу.

2.2.2 Обмеження цілісності

Для темпоральної бази даних з’являються додаткові обмеження цілісності, які теж потрібно враховувати.

Первинний ключ кожної таблиці з підтримкою дійсного часу повинен включати в себе позначку часу. Тобто у кожний момент часу складений первинний ключ має бути унікальним.

Цілісність зв’язків між відношеннями сильно залежить від того, чи є таблиця, яка посилається, та таблиця, на яку вона посилається, темпоральними. Існує чотири випадки:

– жодна з таблиць не є темпоральною;
– тільки таблиця, яка посилається, є темпоральною;
– тільки таблиця, на яку посилається, є темпоральною;
– обидві таблиці є темпоральними.

В залежності від описаних вище випадків та виду темпоральності (таблиці з дійсним часом, транзакційним, бітемпоральні) реалізуються обмеження цілісності, вони відрізняються за складністю та способами реалізації.

Більшість модифікацій на темпоральній базі даних потребує декілька SQL запитів. Наприклад, одне логічне видалення потребує один або два UPDATE запита, DELETE запит, а іноді ще й INSERT. База даних може бути узгодженою в кінці цієї серії тверджень, швидше за все, вона не буде узгодженою в кінці проміжного оператора SQL. Тому кожне обмеження цілісності повинно бути перевіреним після кожної модифікації, а не тільки перед їх початком або в кінці. Перевірка може бути реалізована за допомогою ASSERT операторів або тригерів в залежності від СУБД.

2.2.3 Журнал відстеження

Одним із варіантів підтримки темпоральності у таблиці є журнал відстеження. Він визначає послідовність модифікацій до однієї таблиці, моніторингової таблиці. Журнал відстеження фіксує факт застосування

модифікації, а також дані, що беруть участь у модифікації. Зазвичай він містить додаткову інформацію про зміни, наприклад, коли відбулася модифікація, або хто її виконав, або яке завдання або транзакція здійснила зміну. Журнал відстеження дозволяє реконструктувати вміст відстежуваної таблиці на будь-який час в минулому.

Журнал відстеження відрізняється від таблиць дійсного часу, обговорених у попередньому розділі. Ці таблиці моделювали стан підприємства з часом. На відміну від моделювання стану предметної галузі, журнал відстеження фіксує стан самої модифікованої таблиці з часом.

Перевага окремої таблиці полягає в тому, що моніторингова таблиця (або таблиця історії змін) залишається такою, як раніше, і весь код, який використовує цю таблицю, все ще працює як раніше. Реалізується заповнення журналу відстеження за допомогою тригерів на INSERT/UPDATE/DELETE в моніторингову таблицю.

2.2.4 Вакуумування

Таблиця з транзакційним часом може бути представлена двома таблицями, поточним та архівним сховищем. Архівне сховище містить рядки, які були виправлені, поточна таблиця містить актуальні дані. Для таблиці, що дуже часто змінюються, архівне сховище може стати досить великим. З одного боку, це бажано, оскільки це сховище фіксує різну в часі поведінку таблиці, що підлягає моніторингу, це дозволяє пізніше проаналізувати, що було б важко або неможливо без архівного сховища. Однак врешті-решт простір, необхідний сховищу, може стати надмірним. Транзакції у відстежуваній таблиці стають поступово повільнішими, оскільки вставки в архівне сховище займають більше часу; запити у віртуальну таблицю з транзакційним часом також уповільнюються через величезний розмір архіву.

У якийсь момент стає необхідним зменшити розмір архівного сховища, щоб видалити менш бажані рядки, тим самим збільшуючи простір і ефективність роботи з таблицею, з недоліком зменшення можливості запитів до попередніх станів.

Існує кілька видів вакуумних операцій. Два основних класи – це вакуумування сутностей та темпоральне вакуумування, що відрізняються тим, що саме видаляється. У першому випадку видаляються сутності, які вважаються менш цікавими, із архівної таблиці. В останньому – інформація видаляється на основі того, що вона була логічно видалена в певний момент у минулому. Як приклад, ми можемо очистити всі записи, які на сьогодні вже не відповідають дійсності.

2.2.5 Вибір методів підтримки темпоральності та СУБД

Для вирішення проблеми дослідження темпоральності можуть бути використані наступні три підходи.

По-перше, обрати одну СУБД з підтримкою темпоральних даних на основі проаналізованих літературних джерел. Порівняти за множиною сформованих критеріїв вбудовану підтримку темпоральності у СУБД з використанням фреймворку / створених нових методів для підтримки темпоральності «на рівні додатку» / «на рівні БД».

Перевагою такого підходу є те, що одна СУБД буде повністю проаналізована, будуть створені нові методи оптимізації роботи з темпоральними даними для обраної СУБД.

Недоліком цього підходу є те, що СУБД буде обрана лише на основі проаналізованих літературних джерел, та не обов’язково матиме найкращі показники за сформованими критеріями, тобто їх ні з чим буде порівняти.

Другий варіант, обрати дві СУБД з підтримкою темпоральних даних на основі проаналізованих літературних джерел. Реалізувати підтримку темпоральності «на рівні додатку», «на рівні БД». Порівняти за множиною критеріїв три способи: вбудована підтримка, реалізована «на рівні БД» та реалізована «на рівні додатку».

Перевагою такого підходу є те, що будуть проаналізовані вже дві СУБД, можна буде порівняти не тільки створені методи для однієї СУБД, але й СУБД між собою; будуть створені нові методи оптимізації роботи з темпоральними даними для обраних СУБД.

Недоліком цього підходу є те, що СУБД будуть обрані лише на основі проаналізованих літературних джерел, та не обов’язково матимуть найкращі показники за сформованими критеріями.

І третій варіант, провести дослідження декількох СУБД з підтримкою темпоральних даних на одній схемі БД. Порівняти їх за множиною сформованих критеріїв. Обрати одну з оптимальними показниками.

Перевагою такого підходу є те, що будуть проаналізовані декілька СУБД, та найкраща за сформованими критеріями буде обрана емпірично, а не лише на основі проаналізованих літературних джерел.

Недоліком цього підходу є те, що буде проаналізована лише вбудована підтримка та ніякі інші способи.

Суть методів можна відобразити у таблиці 2.1.

Таблиця 2.1 – Методи проведення дослідження

	Перший підхід	Другий підхід	Третій підхід
Аналіз вбудованої підтримки СУБД	+	+	+
Створення нових методів / способів	+	+	—
Аналіз декількох СУБД	—	+	+

Як можна побачити з таблиці 2.1, найбільш ефективним є другий підхід, тому що він має перевагу по усім пунктам.

У якості альтернатив для визначення, на яких саме двох СУБД буде проводитися аналіз, обираємо СУБД, що були розглянуті в аналізі проблемної області:

– Oracle [16];
– MS SQL Server [17];
– MySQL [18];
– PostgreSQL [19];
– Teradata [20].

У якості критеріїв обираємо наступні:

– міра вбудованої підтримки темпоральності (ставимо оцінку від 0 до 10 на основі теоретичних відомостей). До цієї групи занесемо декілька критеріїв, що були розглянуті в аналізі проблемної області: підтримка періодів, темпоральних ключових атрибутів, предикатів, запитів. Ці критерії важливі, тому що вони показують потенційну можливість СУБД до розширення у напрямку роботи з темпоральними даними, також це можливості для дослідження;
– популярність СУБД (ставимо оцінку від 0 до 10 на основі офіційного рейтингу СУБД [21]). Цей критерій є важливим, тому що дуже важливо, щоб результати дослідження були впроваджені та приносили користь розробникам та клієнтам, тому треба орієнтуватися на СУБД, якими користується більшість;
– кількість існуючих досліджень щодо темпоральності (ставимо оцінку від 0 до 10 на власного досвіду пошуку статей та інформації щодо конкретної СУБД). Цей критерій є важливим, тому що він показує потенційну можливість СУБД до розширення у напрямку роботи з темпоральними СУБД, зацікавленість та попит на вирішення цієї проблеми;

– чи є ця СУБД Open Source (ставимо 0 – не є Open Source, або 1 – є Open Source). Цей критерій є важливим, тому що ми хочемо, щоб результати дослідження були впроваджені та приносили користь розробникам, а легше їх запропонувати в Open Source проекти ніж у продукти великих корпорацій.

Показники заносимо у таблицю 2.2.

Для обрання найкращої стратегії використаємо лінійну адитивну згортку з ваговими коефіцієнтами, тому що всі критерії мають неоднозначний вплив. Ця згортка є досить популярною, застосовується у різних областях, наприклад, при реалізації штучного інтелекту в комп'ютерних іграх [22].

Таблиця 2.2 – Аналіз СУБД за обраними критеріями

	Періоди	Темпоральні ключові атрибути	Темпоральні UPDATE / DELETE	Темпоральні предикати	Темпоральні запити	Популярність СУБД	Кількість існуючих досліджень щодо темпоральності	Чи є Open Source
Oracle	8	10	8	10	0	10	10	0
MS SQL Server	10	0	6	10	0	8	8	0
MySQL	0	0	0	0	1	9	8	1
Postgre SQL	10	5	0	10	0	6	6	1
Teradata	10	8	10	10	4	2	10	0

Проведемо підрахунки для альтернатив (див. табл. 2.3):

Таблиця 2.3 – Підрахунки для альтернативних СУБД

	C1	C2	C3	C4	C5	C6	C7	C8	Результат згортки
A1	8	10	8	10	0	10	10	0	0,242
A2	10	0	6	10	0	8	8	0	0,172
A3	0	0	0	0	1	9	8	1	0,147
A4	10	5	0	10	0	6	6	1	0,169
A5	10	8	10	10	4	2	10	0	0,27
Коефіцієнти	2/20	2/20	2/20	2/20	2/20	5/20	4/20	1/20	

Таким чином, для дослідження слід обрати дві СУБД. По результатам згортки це повинні бути Teradata та Oracle. Також, якщо треба буде розглянути третю СУБД, це буде MS SQL Server. Але Teradata спеціально спочатку створювалася як темпоральна СУБД, на відміну від Oracle та MS SQL Server. Тому було б не дуже правильно порівнювати її з Oracle. Враховуючи всі фактори, зупиняємо вибір на Oracle та MS SQL Server. Після проведення експериментів їх результати порівняємо між собою.

Отже, на основі проведеного аналізу будуть вирішуватися наступні задачі:

- дослідження існуючої вбудованої підтримки роботи з темпоральними моделями даних у реляційних СУБД MS SQL Server та Oracle та підтримка ними стандарту SQL:2011;
- дослідження існуючих способів організації роботи з темпоральними даними: на рівні вбудованої підтримки в обраних СУБД, «на рівні додатка», «на рівні БД» за допомогою написання тригерів, додавання стовпців; Порівняння ефективності способів.

2.3 Планування експериментальної частини дослідження

Проведення дослідження буде складатися з наступних етапів:

а) вивчення інформаційних джерел. Вивчення проблем, які є актуальними у програмній інженерії при роботі з базами даних. Вибір об’єкта та предмета дослідження;

б) вибір методів дослідження. Для дослідження будуть використані наступні методи:

   1) первинні – проводяться з метою збору інформації, вивчення джерел про існуючі СУБД, які підтримують темпоральні моделі даних та роботу з ними, вивчення теоретичної інформації про обрані СУБД на предмет способів та ефективності підтримки темпоральності;
   2) вторинні – проводяться з метою обробки та аналізу отриманих даних після проведення дослідження на декількох СУБД, систематизація та шкалювання результатів після експерименту.

в) розробка математичної моделі для формалізації вирішуваної проблеми;

г) проектування бази даних для проведення дослідження. Буде проводитись з декількох етапів: концептуальне, логічне, фізичне;

д) формування множини критеріїв для порівняння результатів дослідження;

е) реалізація підтримки темпоральності за допомогою вбудованої підтримки та «на рівні БД» за допомогою додаткових стовпців, тригерів у MS SQL Server;

ж) заповнення БД у MS SQL Server даними різного об’єму, проведення замірів для різних типів запитів (SELECT, INSERT, UPDATE, DELETE) на різних об’ємах даних;

з) реалізація підтримки темпоральності за допомогою вбудованої підтримки та «на рівні БД» за допомогою додаткових стовпців, тригерів у Oracle;

и) заповнення БД у Oracle даними різного об’єму, проведення для різних типів запитів (SELECT, INSERT, UPDATE, DELETE) на різних об’ємах даних;

к) реалізація додатку для дослідження підтримки темпоральності «на рівні додатку»;

л) підключення СУБД без підтримки темпоральності до додатку, заповнення даними різного об’єму, проведення замірів для різних типів запитів (SELECT, INSERT, UPDATE, DELETE) на різних об’ємах даних;

м) накопичення результатів дослідження;

н) аналіз та узагальнення отриманої інформації, оцінювання результатів дослідження, наукової новизни та практичного значення;

п) формулювання висновків і рекомендацій.

2.4 Розробка математичної моделі представлення темпоральних даних

Як вже було сказано, час є багатовимірним: існує дійсний, транзакційний та деякі інші види часу, на яких ми не будемо зупинятись. Тому й моделі даних можуть підтримувати жодного, один, два або більше вимірів:

- модель-знімок: не підтримує жодного з вимірів;
- модель з дійсним часом: підтримує лише дійсний час;
- модель з транзакційним часом: підтримує лише транзакційний час;
- бітемпоральна модель: підтримує дійсний та транзакційний час.

Часові мітки дійсного часу зберігають інформацію про зміну деяких параметрів світу, що моделюється, а мітки транзакційного часу надають інформацію про час зміни даних або виправлення помилок. Отже, дійсний і транзакційний час є ортогональними один одному (див. рис. 2.3).

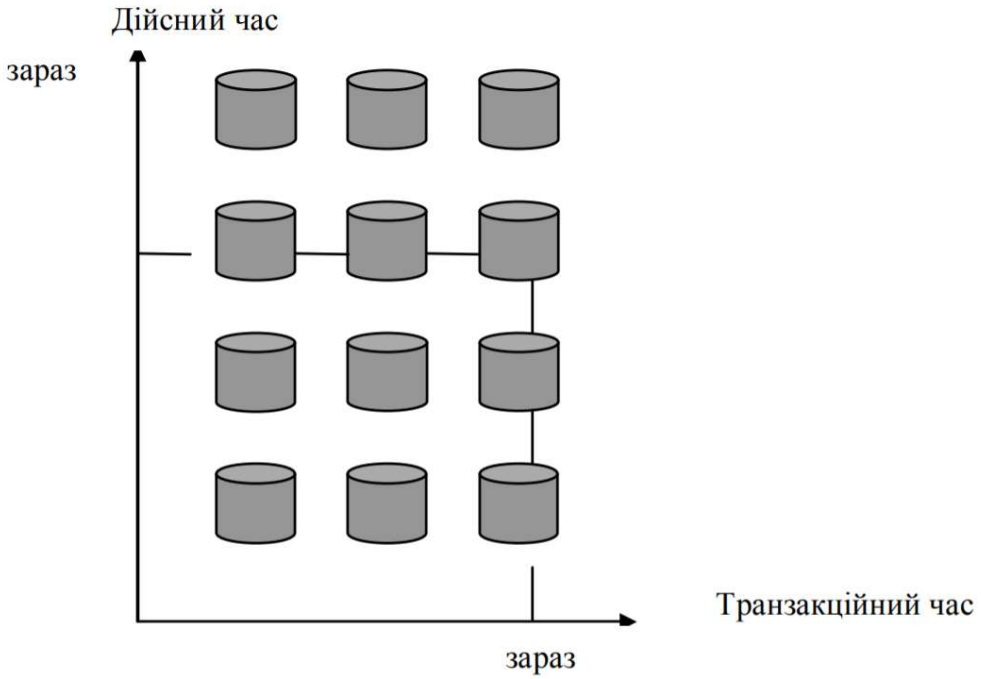


Рисунок 2.3 – Взаємовідношення між дійсним та транзакційним часом

Перейдемо до формального опису моделі темпоральних даних. На базі традиційного уявлення про модель даних [23], що складається з дозволеної організації даних, обмежень цілісності та множини операцій на об’єктами, розширена темпоральна модель (РТМ) може бути представлена, як

$$MT = (DT, OT, CT),$$

де DT – дані, OT – операції, CT – обмеження цілісності; але всі компоненти залежать від часу.

Для додання такої темпоральності була використана відкрита модель з абстрактним ідентифікатором об’єкта (Object Abstract Identify, AOID) [23], яка дозволяє будь-який об’єкт уявити у вигляді реляційного відношення. Життєвий відрізок об'єкта будемо описувати через життєві відрізки всіх його властивостей, визначених у різних відношення, що мають часові атрибути Tstart і Tend, та визначають відповідно час початку і закінчення життєвого відрізка.

Отже, об'єкт $O \in DT$ РТМ буде характеризуватися унікальним абстрактним ідентифікатором OID і набором властивостей A і може бути представлений у вигляді:

$$O = (OID, A).$$

У свою чергу набір властивостей A можна розділити на дві підмножини: множина статичних властивостей A^s, що не змінюється з часом, та множина динамічних властивостей A^d, що змінюються з часом, при цьому:

$$A = A^s \cup A^d, A^s \cap A^d = \emptyset.$$

Загальний вигляд РТМ представлено у вигляді ER-діаграми (див. рис. 2.4).

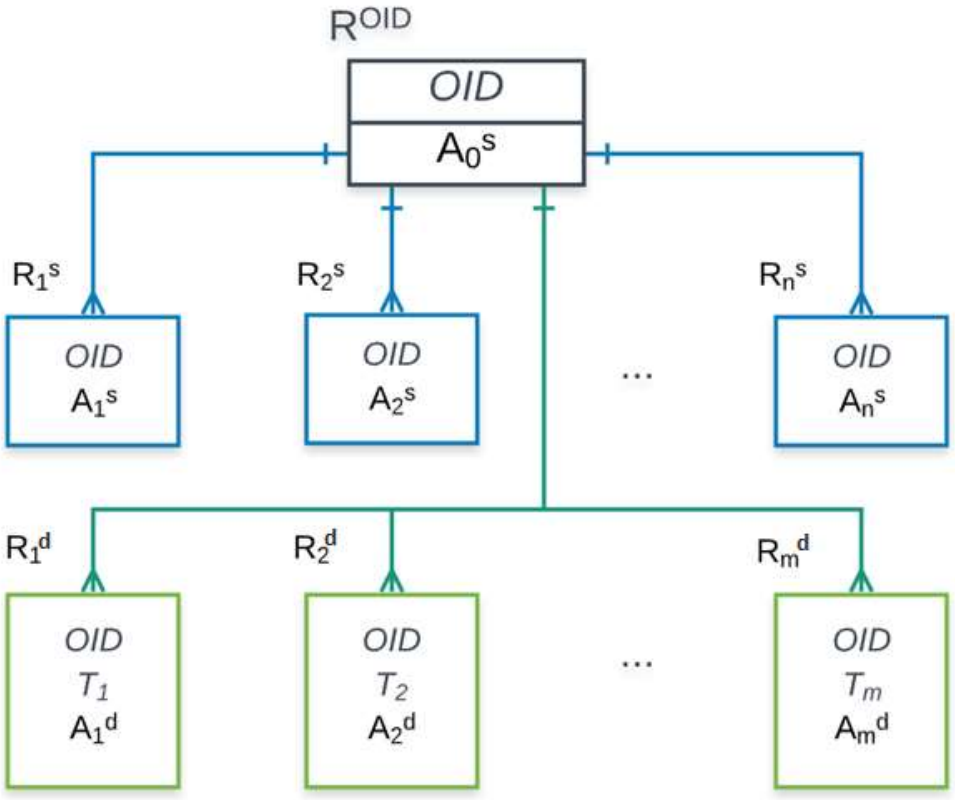


Рисунок 2.4 – ER-діаграма РТМ даних

Оскільки значення атрибутів у РБД повинні бути атомарними, то для представлення статичних та динамічних атрибутів лише реляційного відношення недостатньо. Представимо об’єкт О у вигляді сукупності взаємопов’язаних реляційних відношень:

$$O = \left(R^{OID}, R_1^s, R_2^s, \dots R_n^s, R_1^d, R_2^d, \dots R_m^d\right),$$

де $R^{OID} = R^{OID}(OID, A_0^s)$ – батьківське відношення, що описує абстрактний ідентифікатор об’єкта OID та включає множину статичних атомарних атрибутів об’єкта A_0^s; отже, OID – ключ відношення R^{OID};

$R_1^s, R_2^s, \dots R_n^s$ – дочірні відношення, що описують статичні властивості A^s об’єкта;

$R_1^d, R_2^d, \dots R_m^d$ – дочірні відношення, що описують дискретно змінні у часі динамічні атрибути A^d об’єкта, та мають схеми виду:

$$R_i^d = R_i^d\left(OID, A_i^d, T_i\right), i = 1,2, \dots, m,$$

де Ti – вектор атрибутів часу, що є в залежності від темпоральної форми або дійсним (VT, Valid Time), або транзакційним (TT, Transaction Time), або обома вимірами відразу, $T_i \subseteq T$, $T = \{t_{start}^{VT}, t_{start}^{VT}, t_{start}^{TT}, t_{end}^{TT}\}$ – множина атрибутів часових вимірів.

В подальшому підтримка темпоральності у моделі потребує визначення таких складових, як темпоральний ідентифікатор об’єкта OID, темпоральна алгебра OT та темпоральні обмеження цілісності CT.

Темпоральний ідентифікатор об’єкта OID може використовуватися в якості як первинного, так і зовнішнього ключів. В якості первинного ключа цей ідентифікатор однозначно може визначити стан будь-якого об’єкту у будь-який момент часу. У темпоральних моделях даних первинний та зовнішній ключі залежать від часу, адже час входить до їх складу. Додаткові обмеження цілісності CT, що накладає темпоральна модель даних, можуть бути реалізовані у РСУБД за

допомогою вбудованих можливостей, таких як тригери, функції, збережені процедури.

Операції з темпоральними даними OT можуть бути реалізовані за допомогою реляційних операцій та забезпечуються у тій чи інший мірі вбудованою підтримкою РСУБД операцій з темпоральними даними (темпоральні предикати). Множина операцій має більше розбіжностей з точки зору реалізації у конкретній РСУБД, ніж, наприклад, структура даних.

2.5 Етапи проектування темпоральної бази даних

За останні два десятиліття досвід та дослідження зблизилися на послідовності з трьох основних етапів для розробки та реалізації додатків: (1) концептуальне проектування з використанням моделі ER, (2) логічне проектування з використанням реляційної моделі та, нарешті, (3) фізичне проектування для забезпечення адекватних показників.

На жаль, оскільки і ER, і реляційна модель самі по собі не підтримують адекватно інформацію, що змінюється з часом, поточна практика використання цих моделей фактично непродуктивна. Зазвичай часові атрибути розглядаються дуже рано в концептуальному проекті, коли насправді це повинно бути одним з останніх міркувань у фізичному проектуванні. Надмірно складні схеми ER та реляційні схеми виникли в результаті врахування часу раніше, а не пізніше.

Часові аспекти повинні спочатку ігноруватися при розробці концептуальної схеми. Основна задача – фіксація поточної реальності, тому тимчасово ігноруємо будь-яку історію змін. Тільки після завершення повного проектування ми збільшуємо схему ER за допомогою змінної в часі семантики програми. Ми розглядаємо кожен компонент. Схема ER, у свою чергу, анотує цей компонент його тимчасовою семантикою. Типи відносин, атрибути та ключі розглядаються

кожен окремо. Ці анотації виражаються окремо текстом, щоб вони не захаращували схему ER.

Так само логічне проектування протікає у два етапи. По-перше, нечасова схема ER відображається у нечасову реляційну схему, набір таблиць. Ось ще раз ми ігноруємо часові аспекти програми і, отже, можемо застосувати існуючі стратегії, необтяжені, обмірковуючи, як фіксувати історію. На другому етапі логічного проектування кожна анотація застосовується до логічної схеми, модифікуючи таблиці або обмеження цілісності, щоб врахувати часовий аспект. Ми працюємо дисципліновано, розглядаючи кожну анотацію по черзі.

Далі ми почнемо з нечасової схеми ER і перейдемо до повністю розробленої схеми SQL з урахуванням різного в часі характеру програми.

Будемо дотримуватися поетапної методології при розробці додатків, що працюють з темпоральними даними:

– виконати концептуальний проект, ігноруючи час, отримавши схему ER;
– додати окремо опис тимчасових анотацій текстом;
– відобразити звичайну схему ER в логічну схему SQL;
– застосувати часові анотації, модифікуючи логічну схему;
– завершити фізичне проектування, включаючи часові аспекти.

Цей підхід переносе врахування часових аспектів з самого початку проектування майже до самого кінця. Методологія включає систематичну оцінку часових аспектів кожної конструкції моделювання. Ігнорування часу на початку призводить до менш заплутаних ER-схем, менше помилок виникають під час логічного проектування, і приходить глибше розуміння предметної галузі [15].

2.6 Аналіз предметної області та концептуальне проектування

Протягом концептуального проектування слід ігнорувати усі аспекти, що пов’язані з часом. Повинна бути побудована схема ER, яка не має часових

показників. В цей час слід провести всі міркування щодо проектування, включаючи ключі, обмеження цілісності, складені атрибути, багатозначні атрибути, відношення (один-до-одного, один-до-багатьох, багато-до-багатьох, необов'язкове або обов'язкове).

Як тільки ретельно розроблена схема ER буде побудована, лише тоді слід враховувати часові аспекти.

Для усіх етапів дослідження від концептуального моделювання до проведення експериментів, будемо використовувати одну предметну галузь – медицину, а саме діагностику захворювань. Клінічні дані, що орієнтовані на час, мають критичне значення, вони є основою побудови медичних записів. Дані з довгою історією є важливими для аналізу результатів та багатьох автоматизованих програм підтримки прийняття рішень. Дані про симптоми пацієнтів неможливо переоцінити для точної діагностики захворювань. Ретельний аналіз історії скарг пацієнтів часто дозволяє сформулювати конкретний діагноз без потреби проведення додаткового тестування. Дані про симптоми використовуються для аналізу рішень та необхідні для визначення оптимальні стратегії лікування. Автоматизовані записи про догляд є основними джерелами заявок на підтримку прийняття медичних рішень. Тому існує велика потреба в моделюванні часового аспекту клінічних даних, що зустрічаються в комп'ютеризованій медичній картці.

Введена модель може бути корисною для складних медичних послуг. Отже, коли пацієнт прибуває до лікарні, лікар шукає ознаки та симптоми, щоб ідентифікувати у хворого первинні та вторинні захворювання.

Після підтвердження діагнозу пацієнт повинен розпочати лікування, яке пропонує лікар. Щоразу, коли пацієнт приходить на прийом, лікар спостерігає за ознаками та симптомами пацієнта та приймає рішення, яке може бути: продовжити те саме лікування, змінити ліки або замовити якусь процедуру або тест. Всі ці рішення зберігаються в базі даних для подальшої експлуатації. Минулі епізоди пацієнта можна отримати, якщо вони потрібні. В тому випадку, якщо лікування або набір процедур не покращують стан пацієнта, у лікаря можуть

виникнути сумніви щодо того, як діяти. Так як база даних збирає інформацію про лікування, проведене усіма лікарями в лікарні, будь-який практикуючий лікар може скористатися досвідом або різними підходами до вирішення задач іншими колегами.

Побудуємо ER-діаграму предметної області (див. рис. 2.5). Діаграма на рисунку 2.5 та подальші діаграми етапів проектування БД побудовані за допомогою Microsoft Visio Professional.

Отже, маємо наступні сутності:

- пацієнт з наступними атрибутами: унікальний ідентифікатор, ім’я, прізвище, дата народження та стать;
- відвідування, яке відбувається, коли пацієнт приходить до лікарні на прийом. Має атрибути: унікальний ідентифікатор та опис;
- стан, у якому перебуває пацієнт; Має атрибути: унікальний ідентифікатор та назву; Пацієнт може мати один або більше станів, які будуть змінюватися з ходом лікування;
- симптом, який відноситься до певного стану, має унікальний ідентифікатор та значення; Стан може мати один або більше симптомів;
- діагностика, яка проводиться пацієнтові; Пацієнтові може бути проведена одна або більше діагностик;
- діагноз, який може бути поставлений у результаті діагностики; Під час діагностики може бути поставлений діагноз, а може бути ні. Діагноз має наступні атрибути: унікальний ідентифікатор, назву, опис;
- ліки, які можуть бути призначені для лікування при поставленому діагнозі, мають унікальний ідентифікатор та назву;
- рецепт на ліки, має унікальний ідентифікатор, частоту та дозу прийому; Пацієнтові може бути виписано жодного чи багато рецептів;
- процедура, яка може бути призначена для лікування при поставленому діагнозі, має унікальний ідентифікатор та опис;

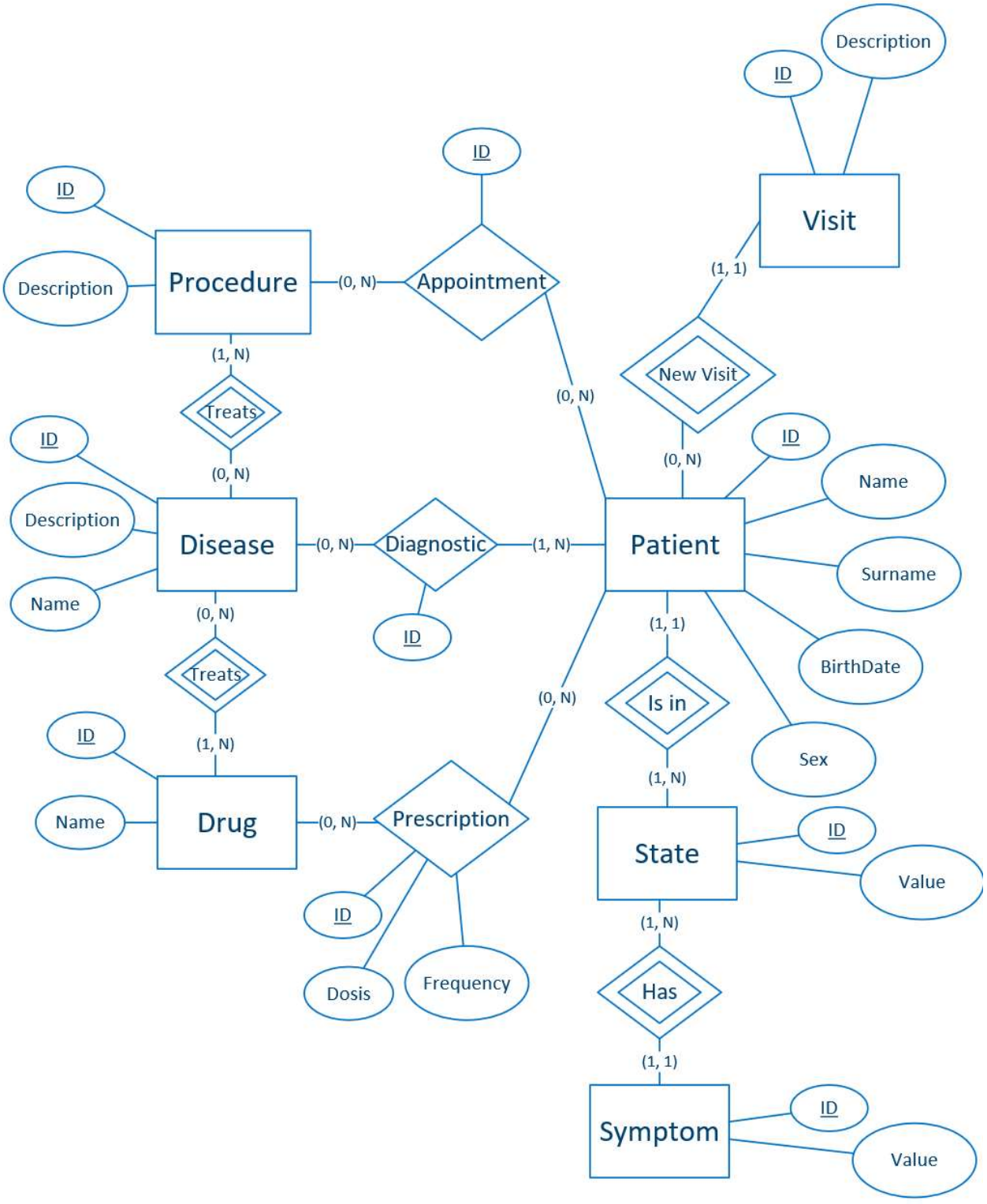


Рисунок 2.5 – Нетемпоральна ER-діаграма, результат концептуального проектування

– призначення на процедуру, має унікальний ідентифікатор. Пацієнтові може бути призначено жодної чи багато процедур.

2.7 Логічне проектування

На етапі концептуального проектування були визначені сутності, їх атрибути та відношення. На етапі логічного проектування розіб’ємо сутності на таблиці та визначимо відношення між ними. Також розглянемо темпоральні аспекти, які були повністю виключені на попередньому етапі (див. рис. 2.6).

Таблиця Patient (Пацієнт) – темпоральні аспекти у цій таблиці відсутні. Має відношення один-до-багатьох з таблицями Visit, Prescription, Appointment, Diagnostic, State;

Таблиця Visit (Відвідування): пацієнт відвідує клініку, кожного разу записується час початку та кінця візиту (це може бути як стаціонар, так і візити на огляд або процедури. Ця таблиця є з підтримкою транзакційного часу та часу, визначеного користувачем (дата початку та кінця візиту). Темпоральні обмеження:

- дата початку відвідування повинна бути раніше, ніж дата кінця для дійсного часу;
- дата початку та дата кінця не можуть приймати значення у майбутньому.

Таблиця State (Стан): пацієнт повинен знаходитися у певному стані: стабільний, тяжкий, задовільний тощо. Має відношення один-до-багатьох з таблицею State. Також є бітемпоральною з наступними темпоральними обмеженнями:

- дата початку визначення стану повинна бути раніше, ніж дата кінця для дійсного часу;
- дата початку визначення стану не може бути раніше, ніж дата першого візиту (розглядаємо стан пацієнта та історію хвороби лише для конкретного медичного закладу);
- періоди знаходження пацієнтів у різних станах не повинні перетинатися;

– не повинно бути проміжків між визначенням стану пацієнта. Після закінчення знаходження в поточному стані відразу йде визначення наступного;
– дата початку не може приймати значення у майбутньому.

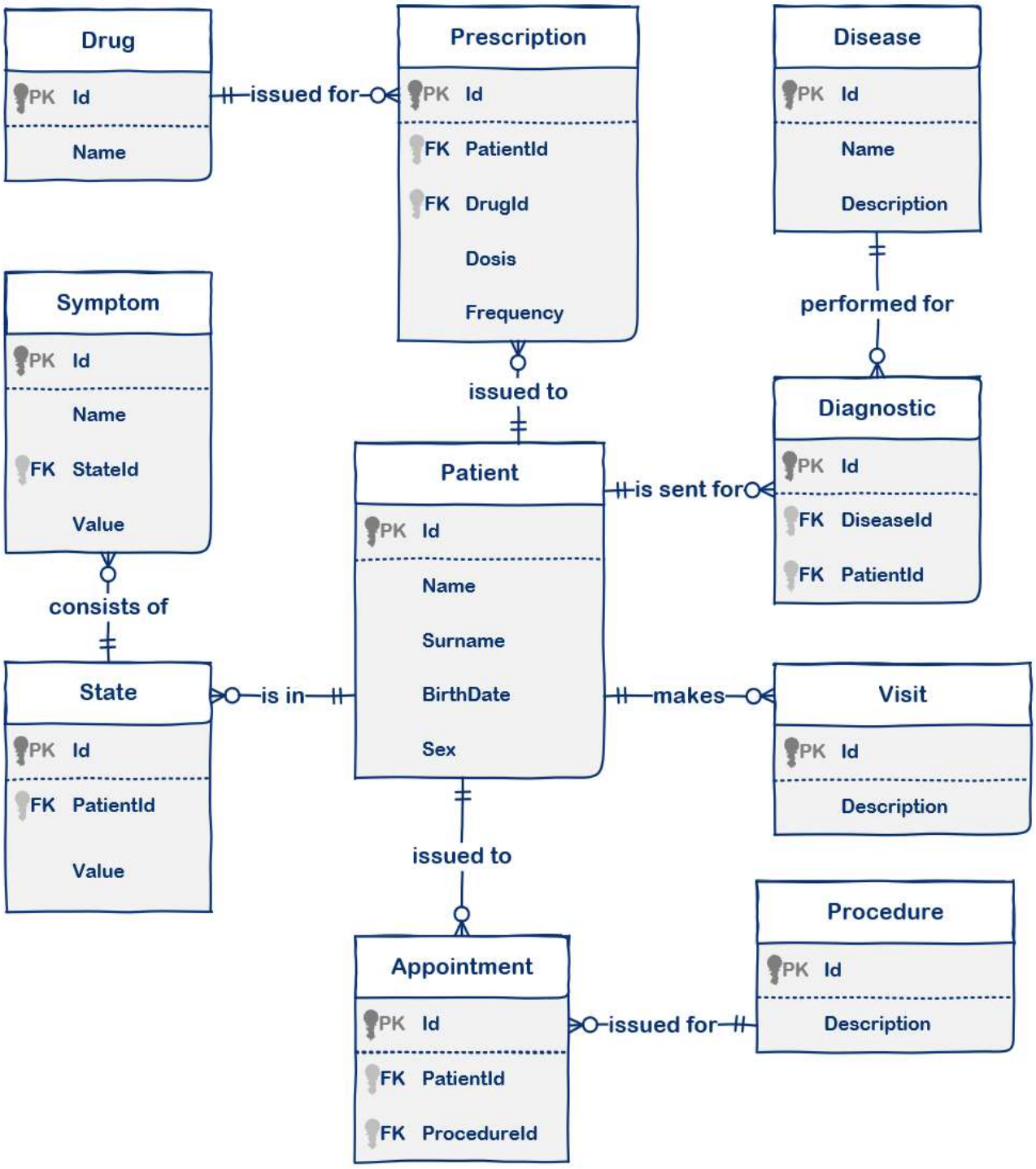


Рисунок 2.6 – Нетемпоральна ER-діаграма у Crow's Foot-нотації, результат логічного проектування

Таблиця Symptom (Симптом). Під час медичних оглядів у пацієнта виявляють симптоми, такі як температура тіла, дихання, біль, показники проведених аналізів тощо. Кожний симптом має дату його фіксації. Ця таблиця є таблицею тільки з підтримкою дійсного часу, а саме миттєвий час. Темпоральні обмеження:

- час фіксації симптому не може приймати значення у майбутньому;
- час фіксації симптому має входити в інтервал дійсного часу одного зі станів пацієнта.

Diagnostic (Діагностика): під час відвідування або стаціонарного лікування пацієнтові проводиться діагностика. Таблиця є з підтримкою транзакційного часу та часу, визначеного користувачем (дата початку та кінця діагностики). Обмеження:

- дата початку діагностики повинна бути раніше, ніж дата кінця для дійсного часу;
- дата початку діагностики не може бути раніше, ніж дата першого візиту;
- дата початку та дата кінця не можуть приймати значення у майбутньому;
- діагноз не може бути поставлений, якщо діагностика не має дати кінця;
- періоди проведення діагностики не повинні перетинатися.

Таблиця Disease (Діагноз): під час діагностики може бути поставлений діагноз. Має відношення один-до-багатьох з таблицею Diagnostic. Ця таблиця не є темпоральною;

Таблиця Drug (Ліки) – нетемпоральна таблиця, має відношення один-до-багатьох з таблицею Prescription;

Таблиця Prescription (Рецепт): на кожні ліки виписується рецепт. Рецепт має строки дії (період дійсного часу). Таблиця є бітемпоральною з обмеженнями:

- дата початку дії рецепту повинна бути раніше, ніж дата кінця для дійсного часу;
- дата початку дії рецепту не може бути раніше, ніж дата першої діагностики, в якій був поставлений діагноз (не пустий зовнішній ключ на таблицю Disease);

- дата початку та дата кінця не можуть приймати значення у майбутньому;
- періоди дії рецепту на одні й ті ж самі ліки не повинні перетинатися.

Таблиця Procedure (Процедура) – нетемпоральна таблиця, має відношення один-до-багатьох з таблицею Appointment;

Таблиця Appointment (Призначення): на кожну процедуру є призначення. Призначення має строки дії (період дійсного часу). Таблиця є бітемпоральною з обмеженнями:

- дата початку відвідування процедури повинна бути раніше, ніж дата кінця для дійсного часу;
- дата початку відвідування процедури не може бути раніше, ніж дата першої діагностики, в якій був поставлений діагноз (не пустий зовнішній ключ на таблицю Disease);
- дата початку та дата кінця не можуть приймати значення у майбутньому;
- періоди відвідування процедур не повинні перетинатися.

На основі розробленої схеми БД далі спроектована фізична модель БД з урахуванням зазначених обмежень та відношень, проведено експериментальне дослідження.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Вибір засобів програмної реалізації

Програмна реалізація буде складатися з декількох етапів:

- реалізація бази даних з підтримкою темпоральної моделі даних у MS SQL Server. На першому етапі буде використовуватися СУБД MS SQL Server, у якості клієнта для написання скриптів використовуватиметься SQL Server Management Studio. Запити будуть написані на мові Transact-SQL (T-SQL) - процедурне розширення мови SQL, створене компанією Microsoft (для Microsoft SQL Server);
- реалізація бази даних з підтримкою темпоральної моделі даних у Oracle. Для другого етапу використовується СУБД Oracle, у якості клієнта обрана SQL Developer Studio. Запити написані на мові PL / SQL - процедурне розширення мови SQL, створене компанією Oracle;
- реалізація підтримки темпоральної моделі даних «на рівні додатку». Серверна частина, що працює зі створеними базами даних, буде написана мовою програмування C#. Для доступу до даних буде використана ORM Entity Framework Core.

3.2 Опис фізичної моделі бази даних

На базі РТМ для предметної області медицини була спроектована схема БД з підтримкою темпоральності (див. рис. 3.1). На рисунку відображений лише дійсний час (щоб забезпечити прозорість); для підтримки транзакційного часу використовуються таблиці-копії з постфіксом History усіх темпоральних таблиць з двома додатковими стовпцями SysStartTime, SysEndTime (атрибути $t_{start}^{TT}, t_{end}^{TT}$ з математичної моделі).

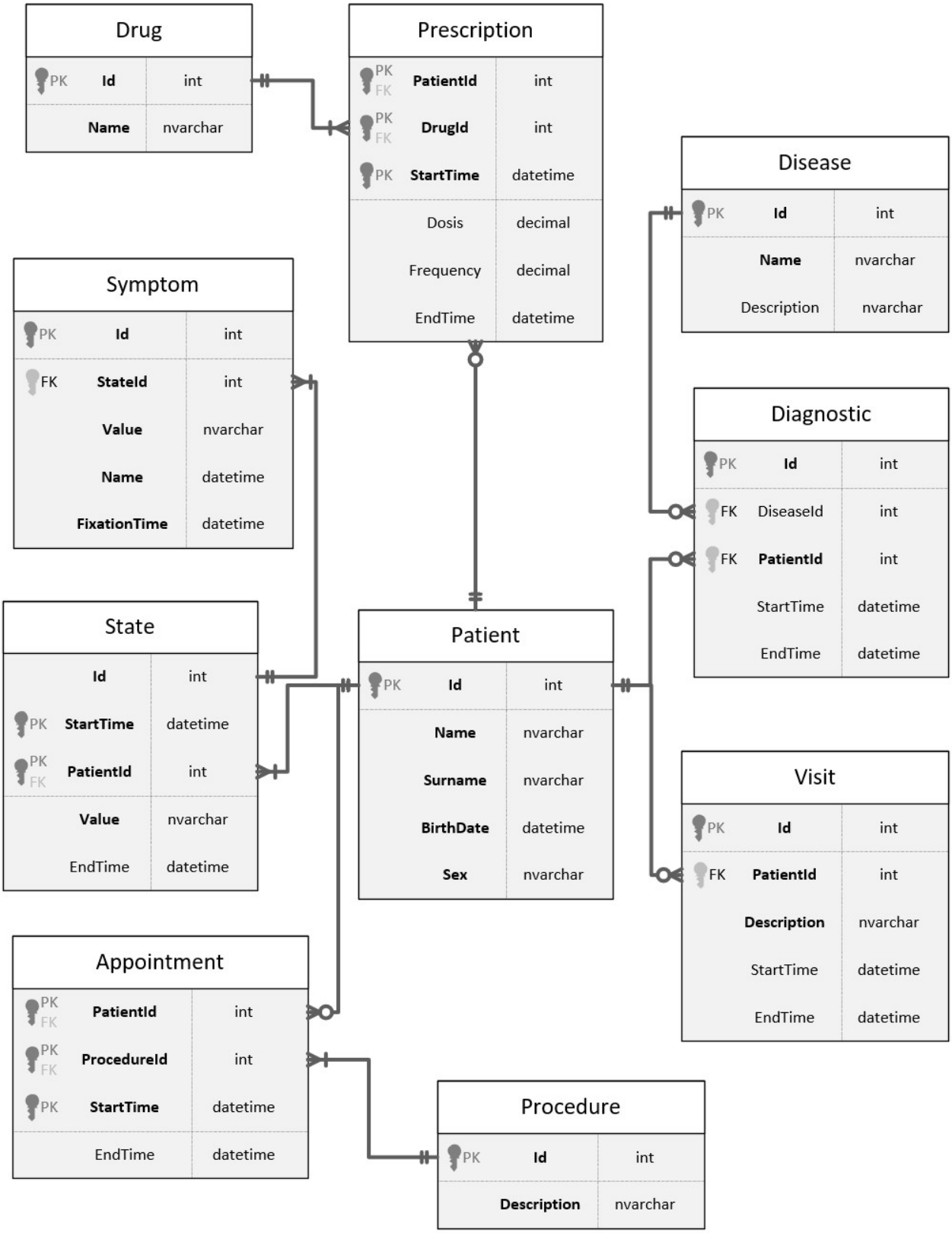


Рисунок 3.1 – Фізична схема бази даних з дійсним часом

Кожна вставка у бітемпоральну таблицю займає одну операцію з проставленням значень у двох колонках – дійсного часу, з якого факт починає

бути істинним (атрибут t_{start}^{VT}), та транзакційного часу, у який цей факт був записаний у БД (атрибут t_{start}^{TT}). Факту, що змінюється з часом, відповідають атрибути $R_1^d, R_2^d, \dots R_m^d$ з моделі.

Кожне оновлення запису у бітемпоральній таблиці супроводжується поновленням попереднього рядка, який був істинним до операції UPDATE (проставлення дати закінчення істинності факту), та вставкою нового з проставленням дати початку істинності нового факту.

Кожне видалення з бітемпоральної таблиці полягає в проставленні дати закінчення істинності факту.

Для будь-якої операції, що змінює стан БД, перевіряються темпоральні обмеження цілісності.

Будь-яка вибірка з БД повертає дійсний стан бази даних на будь-який заданий момент часу.

3.3 Опис програмної реалізації алгоритму

Для реалізації підтримки темпоральності у MS SQL Server були прийняті наступні рішення:

- для підходу «на рівні БД» бітемпоральну підтримку реалізовано самостійно з використанням тригерів INSTEAD OF INSERT / UPDATE / DELETE та перевіркою обмежень у них; рішення використання вбудованої в MS SQL Server підтримки транзакційного часу (відстеження зміни у таблиці, перенос недійсні записи у таблицю StateHistory) було відхилено, тому що для таких таблиць не дозволяється створювати тригери;
- для підходу «на рівні додатку» використано вбудовані можливості MS SQL Server щодо підтримки транзакційного часу, адже тригери вже

непотрібні, усі перевірки цілісності йдуть на рівні додатку. Нижче наведені скрипти, що були розроблені для MS SQL:

- Periods Overlap Function – функція для перевірки пересічності періодів (для відкритих і закритих періодів):

```
CREATE OR ALTER FUNCTION [dbo].[PeriodsOverlapped] (@start1
datetime2, @end1 datetime2, @start2 datetime2, @end2 datetime2)
     RETURNS bit
     WITH EXECUTE AS CALLER
     AS
     BEGIN
        DECLARE @result bit;
SET @result = (SELECT CASE WHEN @start1 between @start2 and @end2
THEN 1 ELSE 0 END);
SET @result = (SELECT CASE WHEN @result = 1 OR (@end1 between
@start2 and @end2) THEN 1 ELSE 0 END)
SET @result = (SELECT CASE WHEN @result = 1 OR (@start1 < @start2
and @end1 > @end2) THEN 1 ELSE 0 END);
SET @result = (SELECT CASE WHEN @result = 1 OR (@start1 > @start2
and @end1 < @end2) THEN 1 ELSE 0 END);

         RETURN(@result);
     END;
```

- Timestamp Is Included in Period Function – функція для з'ясування чи входить момент часу у період (для відкритих і закритих періодів):

```
CREATE OR ALTER FUNCTION [dbo].[IsIncludedInPeriod] (@timeToCheck
datetime2, @start datetime2, @end datetime2)
RETURNS bit
WITH EXECUTE AS CALLER
AS
BEGIN
   DECLARE @result bit;

   if @timeToCheck >  @start and @end is null return 1;

SET @result = (SELECT CASE WHEN @timeToCheck between @start and
@end THEN 1 ELSE 0 END);

    RETURN(@result);
END;
```

- Periods Compare Function – функція для порівняння тривалості періодів (для відкритих і закритих періодів);

- скрипти на створення таблиць (з підтримкою транзакційного часу та бітемпоральних). Приклад створення таблиці Symptom з підтримкою транзакційного часу. У результаті буде створено дві таблиці: Symptom та SymptomHistory (копія таблиці Symptom, що зберігатиме історію змін):

```
CREATE TABLE [dbo].[Symptom](
   [Id] [int] IDENTITY(1,1) NOT NULL,
   [Name] [nvarchar](50) NOT NULL,
   [Value] [nvarchar](max) NOT NULL,
   [StateId] [int] NOT NULL,
   [FixationTime] [datetime] NOT NULL,
   [SysStartTime] [datetime2](7) GENERATED ALWAYS AS ROW START NOT
NULL,
   [SysEndTime] [datetime2](7) GENERATED ALWAYS AS ROW END NOT
NULL,
CONSTRAINT [PK_Symptom] PRIMARY KEY CLUSTERED
(
   [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY],
   PERIOD FOR SYSTEM_TIME ([SysStartTime], [SysEndTime])
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
WITH
(
SYSTEM_VERSIONING = ON ( HISTORY_TABLE = [dbo].[SymptomHistory] )
)
```

- INSTEAD OF INSERT тригери для всіх бітемпоральних таблиць для перевірки обмежень цілісності;
- INSTEAD OF UPDATE тригери для всіх бітемпоральних таблиць для перевірки обмежень цілісності, переносу неактуальних даних у таблицю History та оновлення значень дійсного часу;
- INSTEAD OF DELETE тригери для всіх бітемпоральних таблиць для переносу неактуальних даних у таблицю History та оновлення значень дійсного часу;
- Check <Table Name> Overlap з використанням курсору для всіх бітемпоральних таблиць для перевірки перетинання періодів;
- скрипти на заповнення різних таблиць різними даними різного об’єму. Приклад заповнення таблиці Visit випадковими даними:

```
DECLARE @i int = 250
WHILE @i < 300
BEGIN
      DECLARE @j int = 0
      WHILE @j < 170000
      BEGIN
            SET @j = @j + 2
            INSERT INTO dbo.Visit (PatientId, Description, StartTime,

      EndTime)
VALUES (4, 'Visit #' +cast(@j as varchar), DATEADD(SECOND, -
1000000000 + @j, GETDATE()), DATEADD(SECOND, -1000000000 + @j + 1,
GETDATE()));
      END
      SET @i = @i + 1
END
```

- скрипти для заміру часу виконання запитів;
- скрипти на вибірку даних з таблиць на різні моменти часу.

Підтримка темпоральності у Oracle була реалізована наступним чином:

- для підходу «на рівні БД» бітемпоральна підтримка реалізована самостійно з використанням тригерів: тригери BEFORE створено для перевірки обмежень, а тригери AFTER – для перенесення неактуальних даних в таблицю History; для підтримки періодів використано інструкцію period for при створенні таблиць з періодами (це вбудована підтримка обмежень цілісності щодо перетинання періодів);
- для підходу «на рівні додатку» усі перевірки цілісності реалізовано на рівні додатку.

Для Oracle були розроблені наступні скрипти:

- Periods Overlap Function – функція для перевірки пересічності періодів (для відкритих і закритих періодів);
- Timestamp Is Included in Period Function – функція для з’ясування чи входить момент часу у період (для відкритих і закритих періодів);
- Periods Compare Function – функція для порівняння тривалості періодів (для відкритих і закритих періодів);
- скрипти на створення таблиць (з підтримкою дійсного часу та бітемпоральних);

- BEFORE INSERT тригери для всіх бітемпоральних таблиць для перевірки обмежень цілісності;
- BEFORE UPDATE тригери для всіх бітемпоральних таблиць для перевірки обмежень цілісності;
- AFTER UPDATE тригери для переносу неактуальних даних у таблицю History та оновлення значень дійсного часу;
- AFTER DELETE тригери для всіх бітемпоральних таблиць для переносу неактуальних даних у таблицю History та оновлення значень дійсного часу;
- скрипти на заповнення різних таблиць різними даними різного об’єму;
- скрипти для заміру часу виконання запитів;
- скрипти на вибірку даних з таблиць на різні моменти часу.

Деякі скрипти, що не були наведені в цьому розділі, наведені у додатку Г.

4 ОПИС ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Планування експерименту

У ході експерименту замірялися такі характеристики, як час для SELECT/INSERT/UPDATE/ DELETE операцій у темпоральних таблицях з різною кількістю записів та об’єм зайнятого дискового простору.

Під час дослідження були проведені наступні серії експериментів:

- SELECT/INSERT/UPDATE/DELETE для MS SQL Server з підтримкою темпоральності на рівні БД;
- SELECT/INSERT/UPDATE/DELETE для Oracle з підтримкою темпоральності на рівні БД;
- SELECT/INSERT/UPDATE/DELETE для MS SQL Server з підтримкою темпоральності на рівні додатку;
- SELECT/INSERT/UPDATE/DELETE для Oracle з підтримкою темпоральності на рівні додатку.

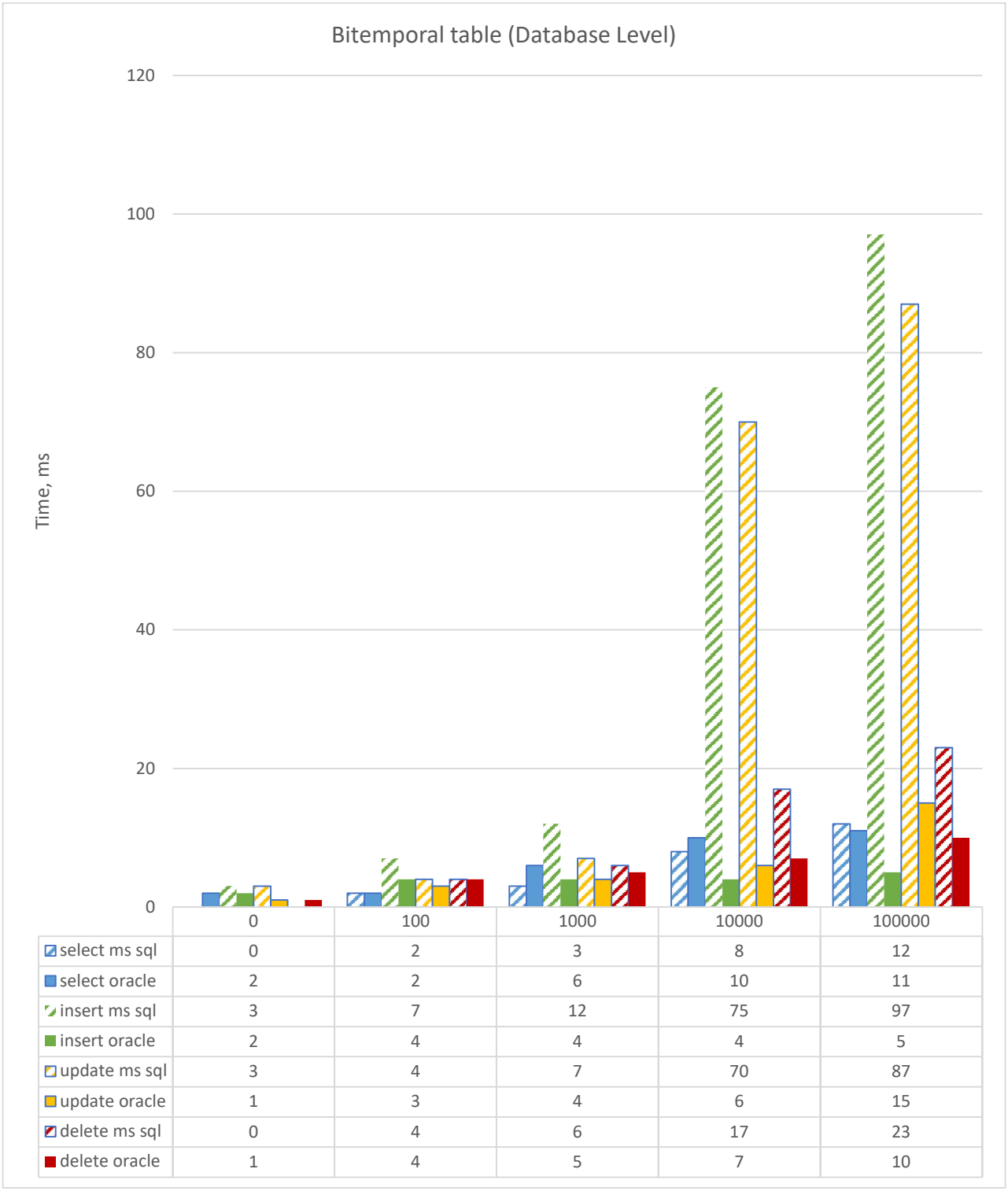
Кожна серія експериментів проводилася на таблицях з 0, 100, 1000, 10000 та 100000 записами.

4.2 Результати проведення експерименту

Результати експерименту перших двох серій з підтримкою на рівні БД зображені на рисунку 4.1. Під графіком надано чисельний еквівалент у мілісекундах щодо кожного виду запиту та легенда кольорів.

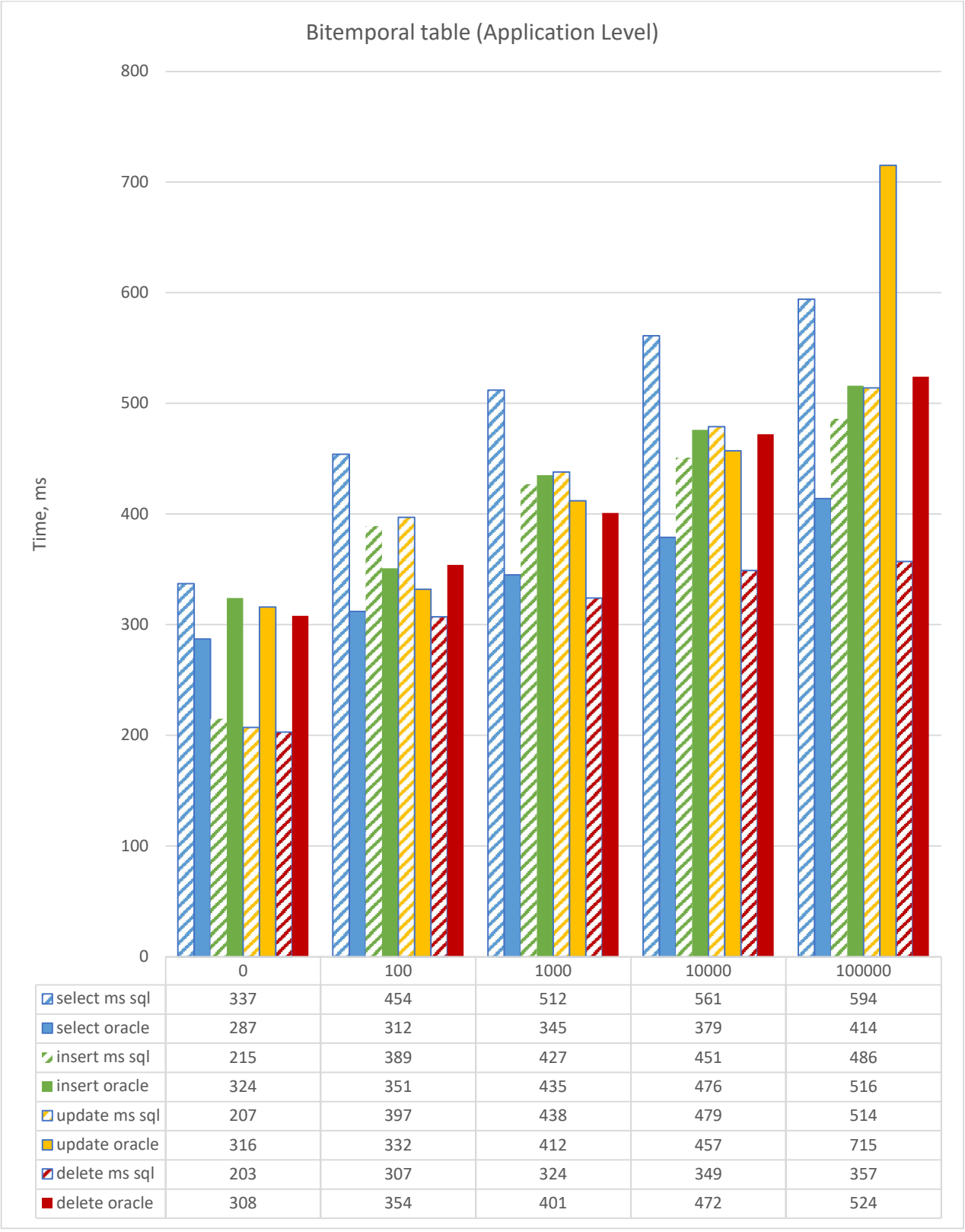
За графіками видно, що MS SQL суттєво програє Oracle на великих обсягах БД. Такі результати замовлено тим, що для перевірки цілісності періодів Oracle має оператор period for при створенні таблиць, тому у тригерах менша кількість

перевірок, у MS SQL всі перевірки реалізуються самостійно за допомогою тригерів, тому INSERT / UPDATE / DELETE запити займають більше часу.



	0	100	1000	10000	100000
select ms sql	0	2	3	8	12
select oracle	2	2	6	10	11
insert ms sql	3	7	12	75	97
insert oracle	2	4	4	4	5
update ms sql	3	4	7	70	87
update oracle	1	3	4	6	15
delete ms sql	0	4	6	17	23
delete oracle	1	4	5	7	10

Рисунок 4.1 – Результати експериментів з Oracle та MS SQL Server для підходу «на рівні БД»



	0	100	1000	10000	100000
select ms sql	337	454	512	561	594
select oracle	287	312	345	379	414
insert ms sql	215	389	427	451	486
insert oracle	324	351	435	476	516
update ms sql	207	397	438	479	514
update oracle	316	332	412	457	715
delete ms sql	203	307	324	349	357
delete oracle	308	354	401	472	524

Рисунок 4.2 – Результати експериментів з Oracle та MS SQL Server для підходу «на рівні додатку»

Результати серій експериментів з підтримкою на рівні додатку наведено на рисунку 4.2. Як бачимо, є залежність від досліджуваних операцій та однозначного

переможця немає. Звісно, що вся робота на рівні додатку збільшує час виконання запитів. Але для MS SQL Server ми використовуємо вбудовану підтримку транзакційного часу, завдяки чому INSERT/UPDATE/DELETE операції виконуються швидше, ніж в Oracle. Бо в Oracle реалізуємо підтримку транзакційного часу на рівні додатку, що є повільнішим.

Результати експериментів також було проаналізовано на базі параметру об'єму диску, що займає БД. І для MS SQL Server, і для Oracle кількість займаного місця з точки зору кількості збережених рядків однакова.

Розглянемо наступний випадок. Нехай ми зробили вставку у таблицю з підтримкою транзакційного часу (незалежно від того, чи підтримується дійсний час, чи ні). Тоді цей запис займає x місця. Збереження самої структури таблиці займає const об'єм. Якщо підтримка темпорального часу реалізована за допомогою History таблиці, то 2 таблиці займають 2const місця. Тобто після першої вставки маємо: $2const + x$ місця.

Далі проводимо операцію оновлення цього запису. Після цієї операції буде зайнято $2const + 2x$ місця. Після наступного оновлення $2const + 3x$ місця. Тобто операція вставки додає x місця, а операція видалення не змінює кількість місця на диску.

Тобто якщо порівняти темпоральну таблицю з підтримкою транзакційного часу та звичайну таблицю, то побачимо наступне.

Нехай v – початковий об'єм записів у таблиці. Після проведення n вставок об'єм буде рівним $v + nx$. Після проведення m операцій оновлення k записів, об'єм буде дорівнювати $v + kx + mkx$. Наприклад, ми оновлюємо 2 записи 3 рази. Після першого оновлення маємо 4 записи (2 актуальних в основній таблиці, 2 у таблиці History). Після другого 6 записів (2 актуальних в основній таблиці, 4 у таблиці History), після третього 8. Звідси і можна вивести формулу:

$$kx + mkx = 2x + 3 * 2x = 8x$$

Операція видалення в звичайні таблиці звільняє px місця. В темпоральних таблиці у гіршому випадку не змінює кількість місця (у гіршому, якщо не було операцій оновлення і вже доданих записів у таблицю History).

Отримані формули наведені у таблиці 4.1.

Таблиця 4.1 – Кількість займаного місця у темпоральних таблицях

	Темпоральна	Нетемпоральна
INSERT	$2const + v + nx$	$const + v + nx$
UPDATE	$2const + v + kx + mx$	$const + v$
DELETE	$2const + v$	$const + v - px$

У таблиці 4.1 v – початковий об'єм записів, n – кількість записів на вставку, k – кількість записів на оновлення, m - кількість операцій оновлення, p – кількість записів на видалення.

Візуалізація отриманих результатів зображена на рисунках 4.3 – 4.5.

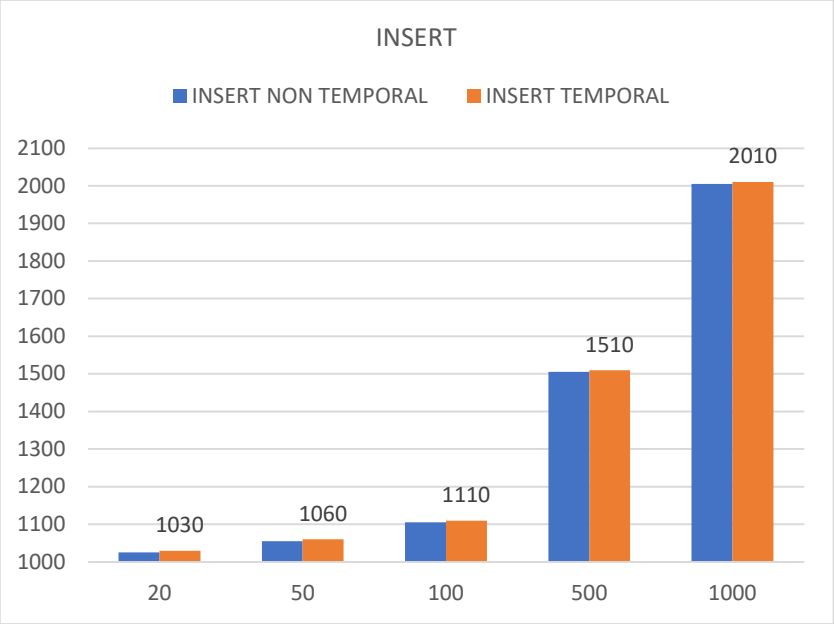


Рисунок 4.3 – Кількість записів після INSERT

Звідси можемо бачити, що таблиці з підтримкою темпоральності займають у рази більше місця, ніж звичайні, тому актуальним стає питання вакуумування записів.

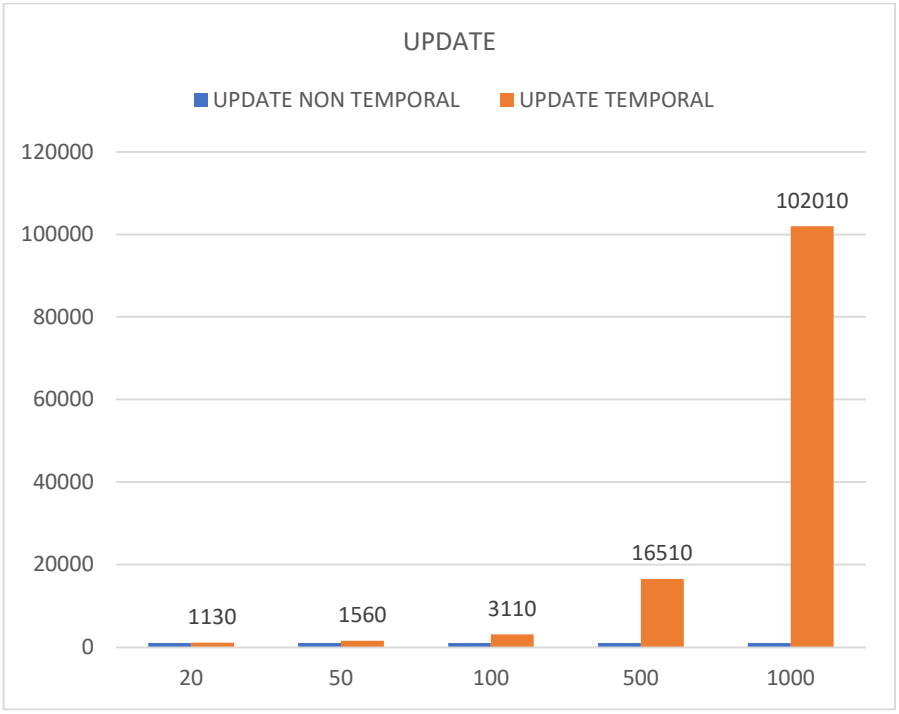


Рисунок 4.4 – Кількість записів після UPDATE

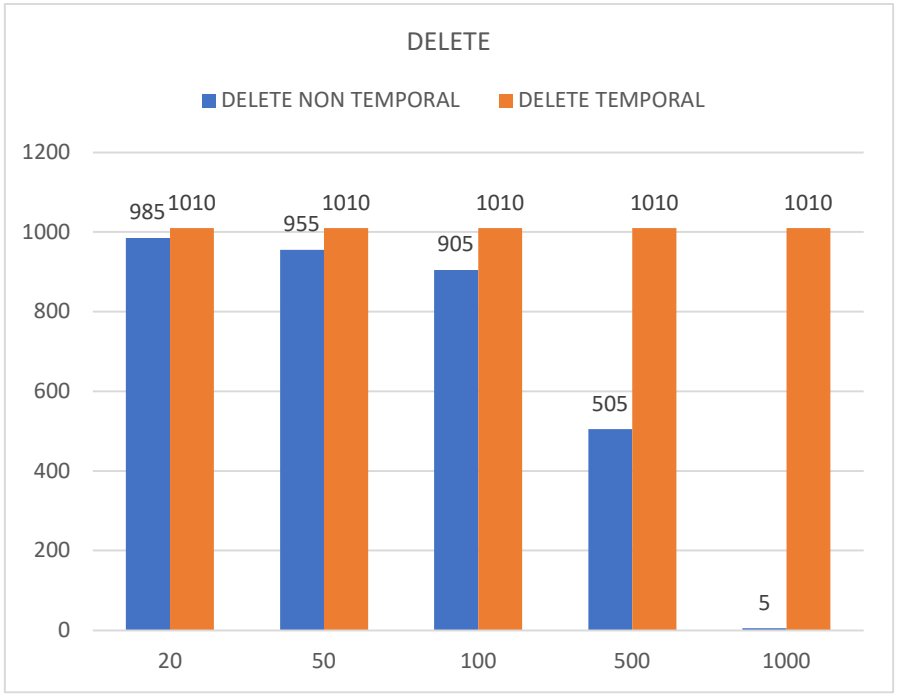


Рисунок 4.5 – Кількість записів після DELETE

Вакуумування – зменшення розміру архівного сховища (History-таблиць), щоб видалити менш бажані рядки, тим самим збільшуючи простір і ефективність роботи з таблицею. Вакуумування в темпоральних БД необхідно проводити

обов’язково. Оскільки вбудованого функціоналу для цього немає ні в MS SQL Server, ні в Oracle, то воно реалізується самостійно.

4.3 Висновки та рекомендації з експериментального дослідження

Провівши дослідження, виконавши заміри та проаналізувавши результати, можна зазначити наступні рекомендації:

а) якщо створюється система, для якої важлива підтримка лише транзакційного часу (ведення журналу відстеження, логування операцій), то краще обрати у MS SQL Server з вбудованою підтримкою транзакційного; тобто використовувати підхід з підтримкою «на рівні БД»;

б) якщо крім підтримки транзакційного часу необхідні темпоральні обмеження цілісності (не лише логування, але й елементи темпоральної логіки), то можна обрати один із трьох варіантів: реалізація перевірок обмежень цілісності на рівні додатку, або реалізація обмежень цілісності на рівні БД за допомогою збережених процедур або функцій, які треба викликати з додатка перед зміною БД, або реалізація підтримки транзакційного часу «на рівні БД»;

в) якщо важлива підтримка лише дійсного часу, то краще обрати Oracle, адже Oracle має інструкцію *period for*, яка реалізує темпоральну перевірку цілісності;

г) якщо треба реалізовувати підтримку і транзакційного, і дійсного часу у повній мірі, то її треба реалізовувати повністю самостійно «на рівні БД» що в Oracle, що в MS SQL Server, тому що:

1) у MS SQL Server неможливо створити тригери на таблицях з вбудованою підтримкою транзакційного часу, а це означає, що неможливо реалізувати перевірку темпоральної цілісності «на рівні

БД», що є дуже важливим при роботі з темпоральними моделями даних. Вбудованої підтримки дійсного часу зовсім немає, її реалізують самостійно;

2) в Oracle немає підтримки транзакційного часу, а підтримка дійсного часу теж не повноцінна.

д) якщо прийнято рішення реалізувати повноцінну бітемпоральну підтримку самостійно у будь-якій СУБД, слід прийняти рішення залежно від архітектури створюваної системи, чи можлива сильна прив'язка системи до певної СУБД, або потрібна можливість легкої зміни СУБД. Це треба вирішити на початковому етапі, тому що логіка на рівні БД повинна буде повністю переписана при використанні іншої СУБД. Якщо ж підтримка темпоральності буде реалізована «на рівні додатку», то перейти на іншу СУБД буде набагато легше, але швидкість роботи буде меншою;

е) у разі реалізації підтримки бітемпоральної моделі «на рівні БД» слід дотримуватися таких рекомендацій:

1) проектування БД здійснюється поетапно, як описується у пункті 4;

2) описується набір правил та обмежень, які повинні бути враховані у темпоральній логіці проектованої системи;

3) формується бібліотека функцій або збережених процедур, які реалізують загальну темпоральну логіку (працюють з періодами, перевіряють обмеження тощо). Надалі їх можна використовувати у тригерах для перевірки цілісності та навіть у інших системах;

4) поступово реалізується підтримка транзакційного і дійсного часу, потім додається перевірка обмежень темпоральної цілісності згідно з бізнес правилами та особливостями модельованої предметної області;

5) обирається та реалізується стратегія вакуумування на History-таблиці.

Таким чином, для вибору оптимальної стратегії реалізації підтримки темпоральності треба брати до уваги такі фактори, як вид часу, підтримку якого треба реалізувати (дійсний, транзакційний або обидва), бізнес-вимоги, вимоги до розширюваності системи, вимоги до часу відклику та розміру можливого займаного об’єму на диску.

5 АНАЛІЗ МОЖЛИВОГО ЗАСТОСУВАННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

У результаті виконання магістерської атестаційної роботи були проаналізовані основні аспекти темпоральних інформаційних систем, сформовані рекомендації щодо вибору способу підтримки темпоральності. Проведене дослідження допоможе обрати стратегію реалізації інформаційної системи з підтримкою темпоральної моделі даних в залежності від потреб клієнта та обраних технологій.

Було запропоновано математичну модель представлення темпоральних даних, яка дає можливість формалізувати систему з темпоральними даними та оперувати загальними поняттями для будь-якої предметної галузі. Базуючись на розробленій моделі можна сформулювати темпоральні обмеження, операції для темпоральних систем з реляційною базою даних. Розроблену модель можна розвивати у напрямку формалізації темпоральних обмежень. Таку доповнену модель можна бути використовувати для опису всіх складових темпоральної моделі даних та використовувати для проектування реляційних систем з підтримкою темпоральності незалежно від предметної області, рівня підтримки темпоральності та обраних технологій реалізації.

Крім цього, запропоновані рішення можуть бути використані як для проектування системи з підтримкою темпоральності з самого початку, так і для впровадження підтримки вже в існуючу систему.

Дані, отримані в результаті проведеного дослідження, можуть бути використані для проведення подальших досліджень щодо підтримки темпоральності: при аналізі інших СУБД або порівнянні ефективності реалізації підтримки темпоральності на рівні додатку залежно від обраних технологій, бібліотек тощо.

Результати проведеного дослідження можуть бути використані для проектування систем для таких галузей, як медицина, продажі, системи з

високоточними обчисленнями, банківська справа. Гарним прикладом останньої є Nykredit із Данії – одна з провідних компаній-постачальників кредитних послуг у Європі. Саме завдяки переходу на використання бітемпоральної підтримки даних, Nykredit змогла якісно обробляти величезні об’єми даних про клієнтів та нерухомість, легко відстежувати помилки у системі, швидко їх виправляти. Компанія відстежувала та аналізувала зміни, які стосувались кожного клієнта чи нерухомості, тому могла запропонувати вигідні пропозиції першою. Завдяки розробленій системі компанія змогла подолати труднощі, які виникли у зв’язку зі зміною закону про іпотечний кредит, який сприяв появі великої конкуренції.

Апробація результатів дослідження наведена у додатку Д у вигляді опублікованих тез та статті.

ВИСНОВКИ

В ході виконання атестаційної роботи були досліджені основні аспекти підтримки темпоральності в реляційних базах даних.

Був проведений аналіз специфіки побудови темпоральних інформаційних систем, виявлені існуючі проблеми, які потребують рішення. Розглянуті способи підтримки темпоральності у реляційних СУБД, виявлені їх переваги та недоліки.

Було запропоновано математичну модель представлення темпоральних даних з урахуванням дійсного та транзакційного часу.

Були спроектовані та реалізовані програмні рішення для проведення дослідження обраних методів. Під час реалізації темпоральної підтримки були реалізовані загальні функції у СУБД Oracle та MS SQL Server, що можуть бути використані при роботі з будь-якими темпоральними моделями даних.

Був спланований та проведений експеримент з реалізацією підтримки темпоральності з використанням двох СУБД – Oracle та MS SQL Server, та двох підходів – «на рівні БД» та «на рівні додатку». Порівнювалися такі параметри, як міра вбудованої підтримки темпоральності, швидкість роботи на різних об’ємах даних та об’єм зайнятого дискового простору.

Результати експериментального дослідження дозволили сформулювати рекомендації щодо вибору підходів до роботи з темпоральними моделями даних у реляційних СУБД.

Отримані рекомендації можуть бути використані при проектуванні систем з підтримкою темпоральності, що поширено в таких прикладних галузях як фінансова, медична, логістична.

За результатами дослідження були опубліковані тези в рамках XVII науково-технічної конференції «Інноваційні технології – 2020» та стаття у науково-технічному журналі «Біоніка інтелекту» (див. додаток В).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

3. Database Trends – SQL vs. NoSQL, Top Databases, Single vs. Multiple Database. URL: https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/ (дата звернення: 13.03.2021)

4. Костенко Б. Б. История и актуальные проблемы темпоральных баз данных. URL: http://citforum.ru/database/articles/temporal/ (дата звернення: 13.03.2021).

5. Steiner A. What is temporal data? URL: http://timeconsult.com/TemporalData/TemporalData.html (дата звернення: 09.01.2021).

6. Балдин А. В., Елисеев Д. В., Агаян К. Г. Обзор способов построения темпоральных систем на основе реляционной базы данных. URL:http://technomag.edu.ru/doc/441884.html, (дата звернення: 07.11.2020).

7. Порай Д. С., Соловйов А. В., Корольков Г. В. Реализация концепции темпоральной базы данных средствами реляционной СУБД // Едиториал. 2013. С. 92-109.

8. В.Г. Григорович, О.Ю. Косовська, О.М. Пігур-Пастернак, А.Ю. Шілінг. Огляд технологій темпоральних баз даних // Вісник Національного університету "Львівська політехніка". 2011.

9. Krishna Kulkarni, Jan-Eike Michels. Temporal features in SQL:2011 // ACM SIGMOD Record. 2012. P. 34.

10. Prashanth Jayaram. Temporal Tables in SQL Server. URL: https://www.sqlshack.com/temporal-tables-in-sql-server/ (дата звернення: 13.03.2021).

11. Implementing Temporal Validity. Документація Oracle. URL: https://www.oracle.com/webfolder/technetwork/tutorials/obe/db/12c/r1/ilm/temporal/temporal.html (дата звернення: 01.03.2021)

12. Multi-temporal Features in Oracle 12c. URL: https://www.salvis.com/blog/2014/01/04/multi-temporal-database-features-in-oracle-12c (дата звернення: 23.01.2021)

13. Historical records with PostgreSQL, temporal tables and SQL:2011. URL: https://clarkdave.net/2015/02/historical-records-with-postgresql-and-temporal-tables-and-sql-2011/ (дата звернення: 23.01.2021)

14. Johann Gamper. Temporal Data Management: An Overview. 2017. URL: https://cs.ulb.ac.be/conferences/ebiss2017/files/slides/gamper_ebiss2017 (дата звернення: 23.01.2021)

15. Snodgrass R. Developing Time-Oriented Database Applications in SQL/Snodgrass R. — Morgan Kaufmann Publishers, 2000. — 504p.

16. Temporal Validity in Oracle Database 12c. URL: https://oracle-base.com/articles/12c/temporal-validity-12cr1 (дата звернення: 17.03.2021)

17. SQL Server Temporal Tables. Documentation. URL: https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver15 (дата звернення: 17.03.2021)

18. MySQL 8.0 Reference Manual. Date and Time Data Types. []URL: https://dev.mysql.com/doc/refman/8.0/en/date-and-time-types.html (дата звернення: 17.03.2021)

19. Postgre SQL SQL2011Temporal. Documentation. URL: https://wiki.postgresql.org/wiki/SQL2011Temporal (дата звернення: 17.03.2021)

20. Teradata Documentation. URL: https://docs.teradata.com/ (дата звернення:17.03.2021)

21. DB-Engines Ranking. URL: https://db-engines.com/en/ranking (дата звернення:12.03.2021)

23. Date C.J., Hugh Darwen, Nikos A. Lorentzos. Temporal Data and the Relational Model: A Detailed Investigation into the Application of Interval and, Relation Theory to the Problem of Temporal Database Management/ Date C.J., Hugh Darwen, Nikos A. Lorentzos. – Morgan Kaufmann, December, 2002. – 480p.