МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
Завідувач кафедри
_____ Скарга-Бандурова І.С.
«____»_____ 20__ р.

МАГІСТЕРСЬКА РОБОТА

НА ТЕМУ:

Дослідження методів оптимізації продуктивності баз даних

Освітньо-кваліфікаційний рівень "Магістр"
Спеціальність 122 - " Комп'ютерні науки"

Науковий керівник роботи: _____ (підпис) І.С. Скарга-Бандурова (ініціали, прізвище)

Консультант з охорони праці: _____ (підпис) Я. О. Критська (ініціали, прізвище)

Студент: _____ (підпис) О. Г. Кучмистий (ініціали, прізвище)

Група: КН-17дм

Сєвєродонецьк 2019

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Комп'ютерних наук та інженерії
Освітньо-кваліфікаційний рівень Магістр
Напрям підготовки _____
(шифр і назва)
Спеціальність 122 - " Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ:

Завідувач кафедри _____
_____ І.С. Скарга-Бандурова
«_____» _____20__ р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Кучмистому Олександру Георгійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів оптимізації продуктивності баз даних

керівник проекту (роботи) Скарга-Бандурова Інна Сергіївна, д.т.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "18" 10 2018 р. № ____
2. Строк подання студентом роботи _____
3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналіз методів і засобів оптимізації продуктивності баз даних
Вивчення технічних аспектів функціонування баз данних
Программна реалізація розглянутих методів (засобів)
Питання охорони праці, екології

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Електронна презентація

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Критська Яна Олександрівна		

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Вивчення літератури за темою роботи	1.09.18 - 2.10.18	
2	Аналіз існуючих програмних та інструментальних засобів	3.10.18 – 9.10.18	
3	Постановка наукової задачі та обгрунтування методики досліджень	10.10.18 – 24.10.18	
4	Аналіз функціонувваня баз даних	25.10.18 – 25.11.18	
5	Програмна реалізація розглянутих засобів оптимізації	13.11.18 – 15.12.18	
6	Оформлення пояснювальної записки	22.12.18 – 28.12.18	
7	Оформлення презентації роботи	29.12.18 – 7.01.19	

Студент _____ Кучмистий О. Г.
(підпис) (прізвище та ініціали)

Науковий керівник _____ Скарга-Бандурова І. С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кучмистий О.Г. Дослідження методів оптимізації продуктивності баз даних.

Через зростання обсягу даних, який є результатом глобалізації ринку, а також комп'ютерного розвитку підприємств, проблеми з продуктивністю на рівні баз даних в існуючих системах стають поширеними і є головним пріоритетом в оптимізації. У цій роботі були представлені методи оптимізації бази даних з урахуванням обмежень, що випливають з існуючої архітектури системи. Були розглянуті як технічні, так і ділові обмеження. Збільшення ефективності в кожному випадку оцінювався за часом, необхідному для виконання конкретної операції з базою даних.

Ключові слова: база даних, оптимізація, SQL запити, підвищення продуктивності існуючих систем.

АННОТАЦИЯ

Кучмистый А. Г. Исследование методов оптимизации производительности баз данных.

Из-за растущего объема данных, который является результатом глобализации рынка, а также компьютерного развития предприятий, проблемы с производительностью на уровне баз данных в существующих системах становятся распространенными и являются главным приоритетом в оптимизации. В этой работе были представлены методы оптимизации базы данных с учетом ограничений, вытекающих из существующей архитектуры системы. Были рассмотрены как технические, так и деловые ограничения. Увеличение эффективности в каждом случае оценивалось по времени, необходимому для выполнения конкретной операции с базой данных.

Ключевые слова: база данных, оптимизации, SQL запросы, повышение производительности существующих систем.

ABSTRACT

Kuchmystyi O. H. Research methods of database performance optimization.

Due to the growing volume of data, which is a result of globalization of the market, as well as computer development of enterprises, problems with performance at the database level in existing systems are becoming common and are the main priority in optimization. In this paper, the methods of database optimization were presented, taking into account the limitations arising from the existing system architecture. Both technical and business constraints were considered. The increase in efficiency in each case was estimated by the time required to perform a specific operation with the database.

Key words: database, optimizations, SQL queries, improving the performance of existing systems.

ЗМІСТ

ВСТУП

Актуальність досліджень. Останні декілька років набирають популярність різноманітні сервіси по збору та обробці інформації, інакше їх ще називають «агрегатори». Вони виконують різноманітні задачі в різних сферах бізнесу: від збору даних з новинних сайтів для подальшої обробки, до пошуку товарів по інтернет магазинам для подальшого аналізу та вибору найбільш прийнятних. Такий підхід досить часто використовують для вибору готелів, прокату автомобілів і т.п. Всі ці сервіси об'єднує одна спільна мета – автоматизувати та спростити процес пошуку і вибору необхідної інформації, або товару. Щоб опрацьовувати великі об'єми даних потрібно постійно слідкувати за продуктивністю системи. У разі ігнорування цього виникає велика ймовірність зіткнутися с поступовою деградацією ефективності роботи програмного засобу. В процесі дослідження буде розглянута вже існуюча система яка реалізує інформаційну підтримку туристичної агенції. Саме на її основі будуть апробовані та впроваджені методи оптимізації продуктивності баз даних. На сьогоднішній день сучасна концепція роботи туристичного бізнесу розглядається як інформаційно-насичена сфера. Основний бізнес процес агентств полягає в підборі існуючих туристичних напрямків, пошуку клієнтів, а також оформлення необхідних документів. Виробниками різноманітних пропозицій в свою чергу є туроператори. Процес пошуку та отримання даних від туроператорів є одним з основних у роботі підприємств даного типу. Саме він вимагає підвищення ефективності та впровадження більш швидших програмних систем.

Актуальність теми полягає у стрімкому зростанню попиту на збір даних та динамічний розвиток бізнесу. Саме це призвело до появи проблем, що постійно з'являються на рівні функціонування баз даних. Тому дуже важливо постійно вивчати нові методи оптимізації та використовувати їх на вже існуючих системах.

Метою даної кваліфікаційної роботи є вивчення методів оптимізації БД та впровадження їх в існуючу програмну систему, яка призначена для організації інформаційного супроводу туристичного агентства.

Щоб досягти поставленої мети необхідно вирішити перелік наступних *завдань*:

1) Зібрати інформацію про найбільш відомі причини уповільнення отримання інформації з бази даних за допомогою декларативного запиту на мові SQL.

2) Провести серію експериментів, в ході яких виявити достовірність ефективності знайдених рекомендацій по оптимізації, а також проаналізувати умови, при яких дані рекомендації доречні в використанні в процесі оптимізації.

3) Проаналізувати отримані результати, підбити підсумки, скласти упорядкований перелік рекомендацій по оптимізації, виявити найбільш ефективні в своїй галузі застосування, виявити найменш ефективні.

4) Вивчити прийоми денормалізації як засіб збільшення продуктивності, провести чисельні експерименти і визначити рішення, що призводять до підвищення продуктивності системи.

5) Провести необхідні впровадження в існуючу систему на основі проведених досліджень.

Об'єкт досліджень: процес оптимізації запитів до бази даних, а також процес модифікації структури бази даних.

Предметом дослідження є алгоритми та рекомендації щодо оптимізації запитів, планів запитів, а також модифікації бази даних.

Методи дослідження: в рамках дослідження використовувалися теорія нормалізації таблиць бази даних, операції реляційної алгебри, теорія експерименту.

Практична значимість, або результатом виконання дослідження є:

- розроблений алгоритм аналізу для оптимізації запитів;
- розроблена методика оцінки ефективності зміни структури бази даних;
- застосування найбільш ефективних рекомендацій до бази даних підприємства.

Публікації. За темою роботи з викладенням її основних результатів опублікована 1 стаття в науковому фаховому виданні України.

Структура та обсяг магістерської роботи. Магістерська робота містить анотацію, вступ, 4 розділи, перлік використаної літератури, додаток. Пояснювальна записка містить 80 сторінок, 15 таблиць та 20 рисунків.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ І ЗАСОБІВ ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ БАЗ ДАНИХ

1.1 Аналіз вимог

Об'єктом дослідження є процес оптимізації запитів до бази даних, а також процес модифікації структури бази даних.

Методи дослідження: в рамках дослідження використовувалися теорія нормалізації таблиць бази даних, операції реляційної алгебри, теорія експерименту.

Завдання дослідження можна виділити наступним чином.

Зібрати інформацію про найбільш відомих причинах уповільнення отримання інформації з бази даних за допомогою декларативного запиту на мові SQL.

Провести серію експериментів, в ході яких виявити достовірність ефективності знайдених рекомендацій по оптимізації, а також проаналізувати умови, при яких дані рекомендації доречні в використанні в процесі оптимізації.

Проаналізувати отримані результати, підбити підсумки, скласти упорядкований перелік рекомендацій по оптимізації, виявити найбільш ефективні в своїй галузі застосування, виявити найменш ефективні.

Вивчити прийоми денормалізації як засіб збільшення продуктивності, провести чисельні експерименти і визначити рішення, що призводять до підвищення продуктивності системи.

Провести необхідні впровадження в існуючу систему на основі проведених досліджень.

1.2 Аналіз програмних та інструментальних засобів

Інструменти аналізу допомагають автоматизувати тяжку роботу з пошуку тих областей, оптимізація або настройка яких могла б дати виграш з точки зору продуктивності сервера. З застосування подібних інструментів дуже добре починати дослідження питань, що відносяться до продуктивності. Якщо який-небудь з них повідомляє про очевидну проблему, то можна приділити їй основну увагу і знайти рішення.

Деніел Ніхтер (Daniel Nichter) підтримує сайт HackMySQL, на якому розміщені деякі корисні інструменти для роботи з MySQL. Так, mysqlreport - це написаний на мові Perl сценарій, який розбирає результат команди SHOW STATUS і перетворює його в зручний для читання звіт, який виводить в файл. Розібратися в цьому досить повному звіті куди простіше і швидше, ніж вивчати те, що вивела команда SHOW STATUS [13].

Ще одним корисним інструментом є програма mysqlsla (MySQL Statement Log Analyzer). З її допомогою можна аналізувати журнал всіх запитів, виконаних сервером, а також журнал повільних запитів (тобто тих, для яких час виконання перевищило заздалегідь задану величину) і будь-який інший. Вона розуміє цілий ряд форматів журналу і може аналізувати кілька журналів одночасно [9].

Комплект інструментів Maatkit – це творіння Берона Шварца (Baron Schwartz). По суті це комплект інструментів командного рядка, які надають важливу функціональність, відсутню в самих продуктах, що поставляються компанією MySQL AB [9].

Одним з інструментів аналізу є сценарій mkqueryprofiler, який вміє виконувати запити, одночасно спостерігаючи за змінними стану сервера. Він роздруковує докладний і зрозумілий звіт про відмінності змінних до і після запиту. Цей звіт дає більш глибоке уявлення про те, як запит впливає на продуктивність, ніж можна було б скласти на підставі одного лише його виконання [9].

Можна подавати запит по конвеєру на стандартний вхід mkqueryprofiler, задавати один або кілька файлів запитів або просто вказати режим спостереження за сервером без виконання запитів (це буває корисно при запуску зовнішнього додатки). Можна також виконувати не запити, а команди оболонки [10].

Звіт mkqueryprofiler розбитий на кілька розділів. Стандартно профілювальник друкує зведену інформацію по всім запитам в пакеті, але може видати звіт по кожному з них окремо або тільки по деяким. Ці звіти легко порівняти за допомогою допоміжного інструменту mkprofilecompact.

Отже, цей звіт дає детальну інформацію про обсяг і характер роботи, виробленої сервером, а це куди цінніше, ніж просте вимірювання швидкості виконання запиту. Наприклад, він може допомогти вибрати один з двох запитів, для яких час виконання на невеликому наборі даних при низькому навантаженні приблизно однаково, але може сильно відрізнятися, якщо даних багато або навантаження велика. Крім того, звіт дозволяє упевнитися в тому, що проведена оптимізація дала ефект. В якомусь сенсі це мініатюрний інструмент еталонного тестування.

1.3 Аналіз математичних моделей і методів для вирішення задачі

Реляційна база даних – це сукупність відносин, що містять всю інформацію, яка повинна зберігатися в БД. Математичний термін «відношення» визначається наступним чином. Нехай дано N множин D1, D2, ..., DN. Відношення R над цими множинами називається безліч впорядкованих N- кортежів виду <d1, d2, …, dn>, де d1 $\in$ D1, …, dn $\in$ DN (N $\geq$ 1). Безлічі D1, D2, ..., DN називаються доменами (областями визначення) відносини R.

Кожен кортеж складається з чотирьох елементів, які вибираються кожен зі свого домену. Порядок елементів у кожному кортежі строго визначений: перший елемент кортежу вибирається з домену D1, другий елемент з домену D2 і т.д. Кожен елемент кортежу є значення одного з атрибутів, відповідного одному з доменів.

З програмної точки зору відношення є файлом, де кожен запис у файлі є кортеж відносин, а поля в записі містять значення відповідних атрибутів або доменів [11].

Кількість атрибутів у кортежі, або число стовпців в таблиці, називається ступенем відносини. Поточне число кортежів, або рядків, називається потужністю відносини і позначається як |R| [11].

Ступінь відношення не змінюється після створення відносини, але потужність відносини буде змінюватися при додаванні нових і видаленні старих кортежів. Схемою відносини R називається перелік атрибутів Ai, даного відносини із зазначенням домену Di, до якого вони належать [11]:

$$S_R = (A_1, A_2, \ldots, A_N), \text{де } A_i \in D_i, 1 \leq i \leq N \quad (1.1)$$

За визначенням всі кортежі розрізняються. Для однозначної ідентифікації конкретного кортежу використовується так званий первинний ключ відносини. Первинний ключ - це атрибут, або набір з мінімального числа атрибутів, який однозначно ідентифікує конкретний кортеж і не містить додаткових атрибутів [12]. Це означає, що якщо окремий довільний атрибут виключити з первинного ключа, то що залишилися атрибутів буде недостатньо для однозначної ідентифікації окремих кортежів.

У реляційної БД таблиці взаємопов'язані і співвідносяться один з одним як головні і підлеглі. Одному рядку головної таблиці можуть відповідати кілька рядків підпорядкованої таблиці (рис. 1.1). Зв'язок головної і підлеглої таблиць здійснюється через первинний ключ (primary key) головною таблиці і зовнішній ключ (foreign key) підпорядкованої таблиці. Зовнішній ключ це атрибут, або набір атрибутів, підпорядкованої таблиці, який в головній таблиці є первинним ключем. Зв'язок головної і підлеглої таблиць схематично зображується лінією, що з'єднує первинний і зовнішній

ключі цих таблиць, із зазначенням однієї стрілки з боку головної таблиці і двох стрілок з боку підпорядкованої таблиці [11].

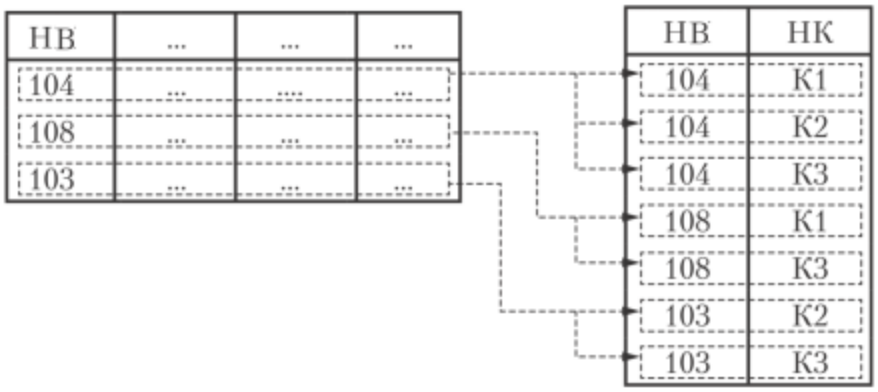


Рисунок 1.1 Зв'язок головної таблиці ВИКЛАДАЧ з підлеглою таблицею ЧИТАЄ [11]

Файл, в якому зберігається відношення, може містити досить багато записів, які відповідають кортежам. Для прискорення доступу до потрібного кортежу файл індексується. Як індексного ключа використовується атрибут або набір атрибутів, визначений у відношенні. Зокрема, індексним ключем може бути первинний ключ.

В результаті індексування створюється додатковий індексний файл, упорядкований за значеннями індексного ключа. Структура індексного файлу може бути різною, але повинна забезпечувати швидкий пошук.

Реляційна алгебра служить математичною основою для реляційної бази даних. Так як відношення є множиною, то реляційна алгебра є алгеброю взаємозв'язків між особливими множинами, званими відносинами [10].

Реляційна алгебра - це замкнута система операцій над відносинами. Ставленням в даному випадку є набором даних, представлених у формі поєднання заголовка і тіла відносини. Тема складається з атрибутів, а тіло з кортежів, тобто безпосередньо рядків з даними. Графічним представленням відносини є таблиця [6].

З основних властивостей відношень можна виділити наступні [9]:

- У відношеннях немає двох однакових елементів (кортежів).
- Порядок кортежів у відношенні не визначений.
- Порядок атрибутів в заголовку відносини не визначений.

Реляційна алгебра являє собою набір таких операцій над відношеннями, що результат кожної з операцій також є відношенням. Це властивість алгебри називається замкнутістю. Оскільки реляційна алгебра є замкнутою, в якості операндів в реляційні

операції можна підставляти інші вирази реляційної алгебри. У реляційних виразах можна використовувати вкладені вирази як завгодно складної структури [4].

Операції над одним відношенням називаються унарними (такий операцією є проекція), над двома відносинами - бінарними (наприклад, операція об'єднання) [3].

Деякі реляційні операції, зокрема, операції об'єднання, перетину і віднімання, вимагають, щоб відносини мали однакові заголовки (схеми). Це означає, що співпадають кількість атрибутів, назви атрибутів і тип однойменних атрибутів.

1.3.1 Пошук вузьких місць: еталонне тестування і профілювання

Найкращою стратегією є пошук і посилення найслабшої слабких місць в ланцюзі компонентів додатку. Це особливо корисно, якщо не відомо, що саме заважає досягненню більш високої продуктивності зараз або може перешкодити в майбутньому.

Еталонне тестування (benchmarking) і профілювання (profiling) - ось два найважливіших методу визначення вузьких місць. Вони пов'язані один з одним, але між ними є різниця. Еталонне тестування вимірює продуктивність системи. Це в свою чергу дозволяє визначити пропускну здатність, зрозуміти, які зміни істотні, а які ні, і з'ясувати, як продуктивність програми залежить від характеру даних [9].

У свою чергу, профілювання допомагає знайти місця, де додаток витрачає найбільше часу і споживає найбільше ресурсів. Іншими словами, еталонне тестування відповідає на питання «наскільки хороша продуктивність?», А профілювання - на питання «чому продуктивність саме така?»

Існують дві основні стратегії тестування продуктивності: тестувати додаток цілком або тільки аспекти, які стосуються SQL. Ці стратегії відповідно називаються повним і по компонентним тестуванням [2].

Однак іноді немає необхідності збирати інформацію про весь додаток. Може знадобиться просто протестувати продуктивність SQL, на сам перед, на початковому етапі. Таке еталонне тестування корисно, якщо:

- потрібно порівняти різні схеми або запити;
- потрібно протестувати конкретну проблему, виявлену в додатку;
- потрібно уникнути тривалих тестів, обмежившись коротким тестом, який дозволить швидко виміряти результати внесених змін.

Крім того, еталонне тестування корисно, коли виконуються характерні для свого застосування запити на реальному наборі даних. Як самі дані, так і розмір набору повинні

бути реалістичними. По можливості необхідно використовувати миттєвий знімок реальних робочих даних.

На жаль, настройка реалістичного еталонного тесту може виявитися складною і тривалою справою, тому, якщо можна отримати копію робочого набору даних, треба використовувати її. Звичайно, це може виявитися неможливим - наприклад, якщо створюється новий додаток, за допомогою якого мало людей і в якому ще недостатньо даних. Якщо потрібно знати, як буде працювати система, коли розростеться, то немає іншого вибору, крім як згенерувати більший обсяг даних і навантаження.

Перед початком тестування потрібно визначити цілі - власне, це слід зробити навіть до початку проектування тестів. Тоді можна буде вибрати інструменти і методики для отримання точних, осмислених результатів.

Можливо, це не здасться очевидним, але іноді потрібні різні підходи, щоб виміряти різні речі. Наприклад, для вимірювання мережевих затримок і пропускної здатності потрібні різні еталонні тести [1].

У підсумковому аналізі потрібно протестувати те, що важливо для користувачів. Тести вимірюють продуктивність, але для різних людей поняття «продуктивність» має несхожий сенс. Потрібно зібрати різні вимоги (формально чи неформально) до того, як система повинна масштабуватися, який допустимий час реакції, який вид конкуренції очікується, і т.п. Потім можна спробувати спроектувати тести так, щоб прийняти до уваги всі вимоги, не обмежуючи кут зору і не фокусуючись на одних аспектах на шкоду іншим.

1.3.2 Оптимізація схеми і індексування

Оптимізація невдало спроектованої або погано проіндексованою схеми може збільшити продуктивність на порядки. Якщо потрібна висока швидкість, потрібно розробити схему і індекси під ті конкретні запити, які будуть запускатися. Крім того, слід оцінити вимоги до продуктивності для різних типів запитів, оскільки зміни в одному з них або в одній частині схеми можуть мати наслідки в інших місцях. Оптимізація часто вимагає компромісів. Наприклад, додавання індексів для прискорення вибірки даних уповільнює їх зміну. Аналогічно денормалізованна схема прискорює деякі типи запитів, але уповільнює інші. Додавання таблиць лічильників і зведених таблиць є хорошим способом оптимізації запитів, але їх підтримка може коштувати досить дорого.

Іноді доводиться виходити з ролі розробника і досліджувати вимоги бізнесу, який доручено автоматизувати. Люди, які не є експертами в області баз даних, часто формулюють бізнес-вимоги, не розуміючи, як вони вплинуть на продуктивність. Якщо

можна пояснити, що незначна, на перший погляд, функція збільшить вимоги до обладнання, вони можуть вирішити, що без неї цілком можна обійтися.

Оптимізація схеми і індексування потребують як погляду на картину в цілому, так і уваги до деталей. Потрібно розуміти всю систему, щоб розібратися, як кожна її частина впливає на інші.

MySQL підтримує безліч типів даних. Вибір правильного типу для зберігання інформації критичний з точки зору збільшення продуктивності.

Індекси являють собою структури, які допомагають MySQL ефективно отримувати дані. Вони критичні для досягнення гарної продуктивності, але багато часто забувають про них або погано розуміють їх зміст, тому індексування є головною причиною проблем з продуктивністю в реальних умовах [4].

Зазвичай існує багато способів представити наявні дані: від повної нормалізації до повної денормалізації з усіма проміжними варіантами. У нормалізованої базі даних кожен факт представлений один і тільки один раз. У денормалізованній базі даних, навпаки, інформація дублюється або зберігається в декількох місцях.

Істина полягає в тому, що повністю нормалізовані і повністю денормалізованні схеми подібні лабораторним дослідженням: вони рідко мають щось спільне з реальним світом. На практиці часто доводиться поєднувати обидва підходи, застосовуючи частково нормалізовані схеми, кешуючі таблиці, і інші прийоми.

1.3.3 Оптимізація запитів

В попереднім розділі було розказано про оптимізацію схеми. Така оптимізація є одним з необхідних умов досягнення високої продуктивності. Але одного цього недостатньо - треба ще правильно конструювати запити. Якщо запит складено погано, то навіть найкраща схема не допоможе вирішити проблеми продуктивності.

Головна причина, через яку запит може виконуватися повільно, - занадто великий обсяг оброблюваних даних. Звичайно, існують запити, які за своєю природою повинні «перемелювати» дуже багато всіляких значень, і тут нічого не можна зробити. Але це досить рідкісна ситуація більшість запитів можна змінити так, щоб вони зверталися до меншого об'єму даних. Аналіз повільно виконується запиту потрібно виробляти в два етапи [1]:

– Зрозуміти, не отримує чи додаток більше даних, ніж потрібно. Зазвичай це означає, що занадто велике кількість відбираються рядків, але не виключено, що відбираються також зайві стовпці.

– Зрозуміти, чи не аналізує сервер MySQL більше рядків, ніж це необхідно.

Метою оптимізації проблемних запитів повинно стати пошук альтернативних способів отримання необхідного результату, хоча далеко не завжди це означає отримання точно такого ж результуючого набору від MySQL. Іноді вдається перетворити запит в еквівалентну форму, досягнувши більш високої продуктивності. Але слід подумати і про приведення запиту до виду, що дає інший результат, якщо це дозволяє підвищити швидкість виконання. Можна навіть змінити не тільки запит, але і код програми.

1.3.4 Оптимізація на рівні додатку

Прагнення до більш високої продуктивності починається з простого факту: додаток реагує занадто повільно, і з цим треба щось робити. Звичайні вузькі місця - це довго виконуються запити, блокування, перевантаження процесора, мережеві затримки і файловий ввід / вивід. Будь-яке з них може стати проблемою, якщо додаток неправильно налаштоване або неналежно використовує ресурси.

Насамперед потрібно знайти причину. Це буде набагато простіше зробити, якщо в додаток вбудовані засоби профілювання. Якщо дана умова дотримується, але все одно не видно, в чому причина падіння продуктивності, то, можливо, доведеться включити додаткові звернення до профілювальнику.

Якщо додаток змушене чекати, тому що його продуктивність обмежена потужністю процесора, і занадто висока ступінь конкуренції, то причиною уповільнення може виявитися «втрачений час», про який говорили раніше. Тому іноді корисно виконувати профілювання в умовах обмеженої конкуренції.

Мережеві затримки можуть «з'їдати» багато часу навіть у локальній мережі. Профілювання на рівні додатку вже включає такі затримки, тому профілювальник покаже, як впливають на продуктивність звернення по мережі. Наприклад, якщо для показу сторінки потрібно виконати 1000 запитів, то всього половина мілісекунди на кожне віддалене звернення додасть до часу реакції цілих пів секунди. Це дуже багато для високопродуктивного ПО [2].

У разі наявності всередині програми досить докладної системи профілювання знайти джерело проблеми буде неважко. Якщо такий системи немає, потрібно додати її.

При аналізі різних додатків ми постійно натикаємося на одні й ті ж проблеми часто тому, що використовуються погано спроектовані готові системи або популярні середовища, що спрощують розробку. Хоча іноді дійсно простіше і швидше скористатися

чимось написаним іншими людьми, але на цьому шляху підстерігає небезпека: ми не знаємо точно, як все влаштовано всередині.

1.3.5 Додаткові засоби MySQL

У версіях MySQL 5.0 і 5.1 з'явилося чимало засобів, які доступні у інших СУБД. Наприклад, збережені процедури, подання та тригери. Додавання в MySQL цих можливостей привернуло безліч нових користувачів. Однак вплив на продуктивність подібних нововведень залишалося неясним до початку широкого застосування.

Багато СУБД вміють кешірувати плани виконання, що дозволяє серверу пропустити стадії розбору і оптимізації запитів, які вже зустрічалися раніше. MySQL при деяких умовах теж може це робити, але, крім того, у неї є кеш особливого виду (так званий кеш запитів), в якому зберігаються повні результуючі набори, створені командами SELECT [5].

Теоретично визначити, корисний кеш чи ні, можна, порівнявши обсяг роботи, який сервер повинен виконати з включеним і відключеним кешем. Якщо кеш відключений, то сервер виконує всі запити на читання і на запис, причому в першому випадку – це генерує і повертає результати. Якщо кеш включений, то при обробці будь-якого запиту на читання необхідно спочатку перевірити кеш, а потім або повернути збережений результат, або (якщо його немає в кеші) виконати запит, згенерувати результуючий набір і відіслати його клієнту. При обробці будь-якого запиту на запис його слід спочатку виконати, а потім перевірити, чи потрібно оголосити недійсними якісь записи в кеші.

MySQL дозволяє зберігати код на стороні сервера в формі тригерів, збережених процедур і збережених функцій. У версії MySQL 5.1 можна також зберігати код в періодично виконуваних завданнях, які називаються подіями. Збережені процедури і функції разом називаються «збереженими підпрограмами» [4].

Для всіх чотирьох видів коду що зберігається в БД застосовується спеціальний розширена мова SQL, що містить такі процедурні конструкції, як цикли і умовні оператори. Найсуттєвіше відмінність між різними видами це контекст, в якому цей код виконується, тобто те, звідки надходять вхідні дані і куди прямують вихідні. Збережені процедури і функції можуть отримувати параметри і повертати результати, тригери і події не можуть [4].

В даний час MySQL пропонує односпрямовані (з прокруткою вперед) серверні курсори тільки для читання, і використовувати їх можна лише в збережених процедурах. Курсор дозволяє через підрядник обійти результат запиту, витягуючи рядки в змінні для

подальшої обробки. Процедура дозволяє відкривати відразу кілька курсорів, причому вони можуть бути «вкладені» одна в одну (у вкладених циклах).

Починаючи з версії MySQL 4.1 сервер підтримує підготовлені (prepared) команди; які використовують розширений двійковий клієнт-серверний протокол, що забезпечує ефективну передачу даних між клієнтом і сервером. Отримати доступ до функціональності підготовлених команд дозволяють бібліотеки, що підтримують новий протокол, наприклад MySQL C API. Бібліотеки MySQL Connector / J і MySQL Connector / NET пропонують ті ж можливості для Java і .NET відповідно. Існує також SQL-інтерфейс для підготовлених команд [4].

Уявлення – це широко поширений в СУБД механізм, який був доданий в MySQL, починаючи з версії 5.0. В MySQL уявлення – це таблиця, в якої не зберігаються дані. Замість цього інформація, «що знаходиться» в таблиці, береться з результатів обробки SQL-запиту.

Більшість розробників не вважають, що уявлення можуть якось підвищити продуктивність, однак в MySQL це не так. Крім того, уявлення можуть бути підмогою для інших методів підвищення продуктивності. Наприклад, якщо рефакторинг схеми відбувається поетапно, то іноді за допомогою уявлень можна зберегти працездатність коду, який звертається до таблиць зі зміною структурою [2].

Зовнішні ключі обходяться не дарма. Як правило, їх наявність означає, що сервер повинен заглядати в іншу таблицю при кожній зміні певних даних. Хоча для прискорення операції InnoDB примусово будує індекс, це зовсім не усуває всі негативні наслідки подібних перевірок [8]. При цьому може навіть вийти дуже великий індекс, який має вкрай низьку селективність. Припустимо, наприклад, що у величезній таблиці є стовпець status, і потрібно, щоб він міг утримувати тільки коректні значення, а таких усього три. Необхідний для цього додатковий індекс помітно збільшить загальний розмір таблиці – навіть якщо розмір самого стовпчика малий і, особливо, якщо велика довжина первинного ключа; при цьому сам індекс не потрібен ні для чого, крім перевірки зовнішнього ключа.

І все ж в деяких випадках зовнішні ключі можуть підвищити продуктивність [8]. Якщо життєво необхідно гарантувати, що дані в двох взаємопов'язаних таблицях несуперечливі, то ефективніше доручити перевірку сервера, а не займатися цим на рівні додатку. Зовнішні ключі корисні також для каскадного видалення та оновлення, хоча ці операції виконуються через підрядник, тобто повільніше, ніж запит з оновленням декількох таблиць або пакетна операція.

1.4 Постановка наукової задачі та обґрунтування методики досліджень

Стрімко зростаючий попит на збір даних та динамічний розвиток бізнесу, призвели до появи проблем, що постійно з’являються на рівні функціонування баз даних. Бази даних використовуються в різноманітних областях застосування і пропонують зберігання складних, як правило, багатовимірних об'єктів. Велика кількість зібраних даних є лише одною проблемою із існуючих. Не менш важливі фактори це збільшення кількості користувачів, розвиток системи шляхом додавання нової функціональності, інтеграція з зовнішніми системами, проблеми з підтримкою серверів, неправильно спроектована схема бази даних и т.п.

Оскільки великі виробничі системи, як правило, мають дуже тривалий життєвий цикл і кількість їх компонентів дуже часто підраховується тисячами, не так рідко зустрічаються додаткові помилки. Велике значення має вміння виявляти такі проблеми та вирішувати їх як можна раніше.

Під час вивчення матеріалу із інших джерел було виявлено ряд особливостей оптимізації роботи баз даних. Найбільш поширена проблема знаходиться на рівні проектування моделі бази даних. Помилки під час цього етапу чи зміни в результаті розробки системи, дуже часто виявляють недоліки моделі. Інший дуже важливий аспект це невірна структура даних, занадто велика індексація стовпців. Дуже висока динаміка та зростання бази даних в поєднанні з багатьма іншими показниками можуть помітно впливати на виконання інструкцій INSERT, UPDATE і DELETE. Деякі проблеми продуктивності виникають через недбале використання тригерів бази даних. Це потужний механізм, який дозволяє автоматизувати процеси. Однак це також може спричинити серйозні проблеми із базою даних які досить важко виявити в існуючій системі.

Основним завданням даної роботи є опис проблем і запропонування можливих підходів до їх вирішення. Ці аспекти описані в контексті реальної існуючої системи, побудованої на базі MySQL Server. Однак варто зазначити, що розглянуті механізми, проаналізовані в роботі, можна застосувати на будь-якому сервері баз даних.

Враховуючи запропоновані засоби необхідно було перевірити їх на існуючій системі. Для цього була створена інформаційна система, призначена для автоматизації процесу збору та обробки інформації. Основна мета якої була спрямована на організацію інформаційного супроводу туристичного агентства де в ролі джерела виступають різноманітні туристичні оператори, які в свою чергу надають дані своїм клієнтам (агентам).

Реалізована програмна система повинна задовольняти потреби туристичних агентств у наданні інформаційної підтримки їх діяльності, успішно стимулювати залучення нових клієнтів, а також треба розглядати можливість подальшого розвитку підприємства і цієї системи зокрема.

Інформаційна система повинна бути реалізована як веб-додаток і виконувати наступні завдання (функції):

- збір даних від різноманітних туроператорів;
- виконувати приведення отриманих даних до заданого шаблону;
- автоматично архівувати та управляти застарілою інформацією (турами);
- виконувати очистку від застарілих даних (турів старіше 3 місяців з дня їх архівації);
- очікувана потужність - не менше 1000 записів в сек. (під одним записом мається на увазі одна позиція туру);
- авторизація та доступ до панелі адміністрування даних;
- веб-інтерфейс для залучення потенційних клієнтів агентства;
- перегляд клієнтами сторінок готелів, турів, курортів і т.п.;
- повідомляти турагента по засобам СМС та електронної пошти про надходження заявки від клієнта.

1.5 Висновки до першого розділу

Розвиток існуючих систем передбачає необхідність розробки та вдосконалення механізмів оптимізації баз даних. Методи оптимізації, описані в роботі, є достатніми для визначення та вирішення дефіциту продуктивності. Доступні оптимізаційні механізми допомагають мінімізувати ризики, пов'язані з тим, що система працює у виробничих середовищах. Підхід до кожної проблеми має відповідати загальним правилам, але також слід приймати окремі міркування та конкретний аналіз ситуації. Єдині процедури оптимізації всіх проблем можуть призвести до їх неправильної класифікації, а отже до неправильного методу їх вирішення. Контекст існуючої системи викликає суттєві обмеження щодо доступних методів оптимізації баз даних. Бізнес ризики, пов'язані з підривом стабільності робочої системи, є важливим фактором у остаточному виборі методів. Як правило, оптимізація спрямована на зменшення кількості конкретних ресурсів, необхідних для отримання результату, що впливає на час виконання для SQL-запитів або частин коду.

1.6 Література до першого розділу

1. Powell, G., Oracle Performance Tuning for 10gR2, Elsevier Inc., 2007.
2. Koper, R., Analysis of the database optimization methods in the context of existing systems, Master's thesis, Lodz University of Technology, 2015.
3. David Kroenke, David Auer. Database Concepts (3rd Edition). – M.: , 2007.
4. Michael Widenius. MySQL Reference Manual. – M.: , 2002.
5. Hugh E Williams. Web Database Applications with PHP & MySQL. – M.: , 2002.
6. Darl Kuhn, Sam R. Alapati and Bill Padfield, Expert Oracle Indexing and Access Paths, Apress Inc., 2016.
7. Ibrar Ahmed, Gregory Smith, Enrico P., PostgreSQL 10 High Performance, Amazon Digital Services LLC, 2017.
8. Benjamin Nevarez, High Performance SQL Server, Apress Inc., 2016.
9. Б. Шварц, П. Зайцев, В. Ткаченко, Дерек Дж. Бэллинг, MySQL. Оптимизация производительности, 3-е издание, Символ-Плюс, 2014.
10. Илюшечкин В. И., Основы использования и проектирования баз данных, Гриф УМО, 2014.
11. Ржеуцкая С.Ю., Базы данных. Язык SQL: учебное пособие, ВоГТУ, 2013.
12. Л.В. Рудикова, Базы данных. Разработка приложений. – СПб.: БХВ-Петербург, 2006.

РОЗДІЛ 2
ЗАСОБИ ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ БАЗ ДАНИХ

2.1 Визначення поняття ефективності та життєздатність

Комп'ютерна промисловість фактично визнає два споріднені терміни: продуктивність і життєздатність. Продуктивність відображається часто в числових термінах, як саме працює комп'ютер або програмне забезпечення. Наприклад, можна виразити продуктивність процесора з точки зору коефіцієнта використання або продуктивності запиту з точки зору часу виконання. Важливо пам'ятати про продуктивність, що це абсолютна величина: вона не може буди граною або поганою. Наприклад, процесор який використовують зі 100 відсотками не є, з точки зору продуктивності, поганим. Життєздатність, однак, використовує дані про продуктивність і реальні очікування від системи. Життєздатність визначає яка продуктивність вважається гарною або нормальною, а яка погана або незадовільна. Життєздатність часто використовує діапазони значень продуктивності. Наприклад, процесор з використанням від 0 до 70 відсотків може вважатися таким який функціонує у здоровому режимі, тоді як використання від 70 до 80 відсотків може вважатися зоною попередження, а використання більше 80 відсотків може вважатися шкідливим.

Комп'ютерні адміністратори всіх типів завжди прагнуть покращити продуктивність своїх серверів, як і адміністратори СУБД яких часто називаються адміністраторами баз даних (DBA). Проблема більшості налаштувань продуктивності полягає в тому, що вона стосується лише продуктивності, яка по суті є відокремленою, тому що сама по собі не означає всю суть. Замість цього треба почати думати про продуктивність як спосіб поліпшення життєздатності сервера.

Ось гарний приклад в рамках SQL серверу: є комп'ютер з встановленою СУБД, який підтримує основний бізнес-процес компанії. Середній запит на сервері виконується всього за пару мілісекунд, що, як правило, є цілком прийнятним. Проте користувачі програми скаржаться, що програма виглядає занадто повільною і що, здається, вона стає все повільнішою. Насправді порівнюється продуктивність – абсолютна швидкість запитів зі життєздатністю - відносні відчуття користувачів у реагуванні програми. Це нагадує порівняння яблук і апельсинів, тільки з продуктивністю та життєздатністю. Замість цього

потрібно створити загальне порівняння, визначивши критерії для серверів. Наприклад, можна створити діапазони, які визначають швидкість виконання запиту менше 1 секунди як здорову. У такому випадку комп'ютер з SQL сервером буде вважатися абсолютно здоровим, незважаючи на скарги користувачів.

Суть життєздатності полягає у визначенні прийнятних рівнів продуктивності для кожного аспекту комп'ютера або програми. У цьому прикладі швидкість виконання запиту є корисною, але користувачі все ще скаржаться, тобто потрібно подивитися на якийсь інший аспект програми, щоб покращити загальну продуктивність програми. Визначення критеріїв життєздатності, на відміну від перегляду вихідних даних продуктивності, допомагає швидко визначити, коли певний аспект сервера знаходиться за межами прийнятних діапазонів, і зосередитися на цьому аспекті для покращення продуктивності. У той же час, життєздатність дозволяє визнавати, коли певний компонент знаходиться в межах діапазону прийнятного функціонування, і зосередитися на інших компонентах для поліпшення продуктивності.

SQL сервер – це неймовірно складна система, де десятки різних компонентів сприяють загальній продуктивності бази даних. Якщо бути зосередженим виключно на показниках продуктивності, часто опиняється ситуація зайвої боротьби. Намагаються вдосконалити кілька додаткових показників продуктивності з серверного компонента, який працює вже досить гарно. Створюючи стандарти життєздатності для цих різних компонентів, можна буде класифікувати продуктивність кожного компонента як прийнятне або неприйнятне, а також швидко зосередитися на компонентах, які потребують уваги.

2.2 Як працює SQL сервер

2.2.1 Журнал транзакцій

Коли надсилаються зміни до серверу – за допомогою оператора INSERT, UPDATE або DELETE – данні фактично не записуються до файлу на диску але зміни зберігаються. Натомість сервер записує зміни в журналі транзакцій, що буквально є списком всіх операцій (або транзакцій), виконаних на сервері. Далі сервер вивантажує данні на диск і завантажує виконані сторінки даних в оперативну пам'ять. СУБД робить зміни на сторінках даних в оперативній пам'яті, а потім записує ці сторінки на диск. Ідея полягає в тому, що зберігання сторінок в пам'яті дозволить серверу робити додаткові зміни до них швидше, оскільки може швидше звертатися до пам'яті, ніж до файлів на диску [1].

Зрештою, сервер збереже змінені сторінки даних назад на диск. Тому що дані, що містяться на цих сторінках, є досить безпечними, оскільки дискове сховище набагато менш мінливе, ніж оперативна пам'ять. Тому після збереження даних на диск сервер перевіряє транзакції в журналі транзакцій, які були пов'язані з цими даними. Контрольна точка дозволяє знати, що ця транзакція успішно виконана та збережена на диск [1].

Журнал транзакцій підтримує порядок отримання транзакцій сервером. СУБД завжди зберігає сторінки даних назад на диск у тому порядку, в якому вони відбулися. Іншими словами, якщо виконується десять оновлень на сервері, ці оновлення будуть виконуватися, зберігатися на диску і перевірятися в тому ж порядку, в якому їх визначили [1].

Журнал транзакцій, очевидно, відіграє важливу роль у відновленні стану після аварії. Якщо сервер зазнає помилки, наприклад, відключення електроенергії або прямий збій, будь-які модифіковані сторінки даних, які ще знаходяться в оперативній пам'яті, будуть втрачені. Коли СУБД перезавантажується, він перевіряє журнал транзакцій на будь-які транзакції з необробленими посиланнями і негайно повторно виконує ці транзакції. Така поведінка гарантує, що метод зберігання сторінок даних у пам'яті на деякий час не призведе до втрати усіх даних у разі збою сервера. Оскільки журнал транзакцій використовується майже для кожної транзакції, він може стати вузьким місцем для продуктивності. Наприклад, якщо журнал транзакцій розташований на більш повільному жорсткому диску, швидкість цього диска обмежить швидкість, з якою сервер може приймати вхідні транзакції.

2.2.2 Індекси

Індекси відіграють ключову роль у продуктивності SQL. Коли людина намагається знайти інформацію в особливо великій друкованій книзі, досить легко оцінити важливість індексів. Уявімо, наприклад, що є книга на 2000 сторінок про теорію проектування баз даних і що потрібно знайти першу сторінку, яка згадує про нормалізацію бази даних. Є два варіанти пошуку інформації. Один із способів полягає в тому, щоб виконати сканування, яке просто означає перевіряти кожну сторінку, поки не помітимо термін «нормалізація бази даних». Інший метод полягає у використанні індексу книги, який дозволяє шукати термін «нормалізація бази даних» в алфавітному списку інших термінів і надає перехресне посилання на сторінки, на яких можна знайти термін. Використання індексу для пошуку термінів у такій великій книзі, очевидно, буде набагато швидше, ніж сканування [2].

Проте індекси не завжди є надійним способом підвищення продуктивності. Якщо потрібно знайти термін у 10-сторінковій брошурі, використання індексу може бути повільніше, ніж просто сканування на потрібний термін. І хоча індекси можуть значно покращити продуктивність запитів у великій кількості даних, вони зменшують продуктивність при оновленні даних. Тому що індекс потрібно оновлювати кожного разу, коли дані оновлюються.

SQL дозволяє створювати індекси на окремих стовпцях таблиці. Можна індексувати один стовпець або індексувати кілька стовпців (називається складним індексом). Ефективність запитів зазвичай покращиться, якщо ваш запит містить індексовані стовпці. Проте кожен доданий індекс зменшить продуктивність сервер при оновленні даних, оскільки кожен індекс - це ще одна річ, яку сервер повинен оновити при внесенні змін до фактичних даних. Фактично MySQL підтримує два типи індексів: кластерні і некластерні [2].

2.2.3 Кластерні індекси

Кластерні індекси фізично змінюють порядок збереження даних. Повертаючись до прикладу книжки, кластерний індекс фізично змінив би всі слова в книзі, щоб вони з'являлися в алфавітному порядку, якщо б звичайний індекс книги був фактичним змістом книги. Кластерні індекси дуже швидкі, оскільки пошук елемента в межах індексу означає, що знаходиться фактичний елемент [2].

Однак, оскільки кластерні індекси впливають на фізичний порядок даних, що зберігаються на диску, кожна таблиця може мати лише один кластерний індекс.

Насправді кожна таблиця у SQL має кластерний індекс, незважаючи на інші дії з індексами. Якщо явно не створюється кластерний індекс, сервер використовує фантомний кластерний індекс, який просто розміщує рядки у порядку за їх порядковим номером, що в основному є порядком створення рядків. Цей порядок зазвичай не відображається взагалі.

2.2.4 Некластерні індекси

Некластеризований індекс зберігає копію індексованого стовпця (або стовпців) у порядку, а також вказує на фактичні дані. Це означає, що некластерізовані індекси дозволяють швидко знайти певний термін або іншу частину даних, але вимагають, щоб сервер зробив додатковий крок, щоб фактично знайти дані. Цей додатковий крок називається обходом і представляє основну відмінність між кластерними та

некластерними індексами. Коли знаходяться дані в кластерному індексі, сервер вже знаходиться на фактичній сторінці, що містить повний рядок таблиці. Натомість коли знаходяться дані в некластерному індексі, все, що має сервер, це є вказівник на повну таблицю рядка і має виконувати обхід, щоб знайти цей вказівник і отримати фактичні дані з рядка таблиці. На рис. 2.1 показано взаємозв'язок між кластерними та некластерними індексами [2].

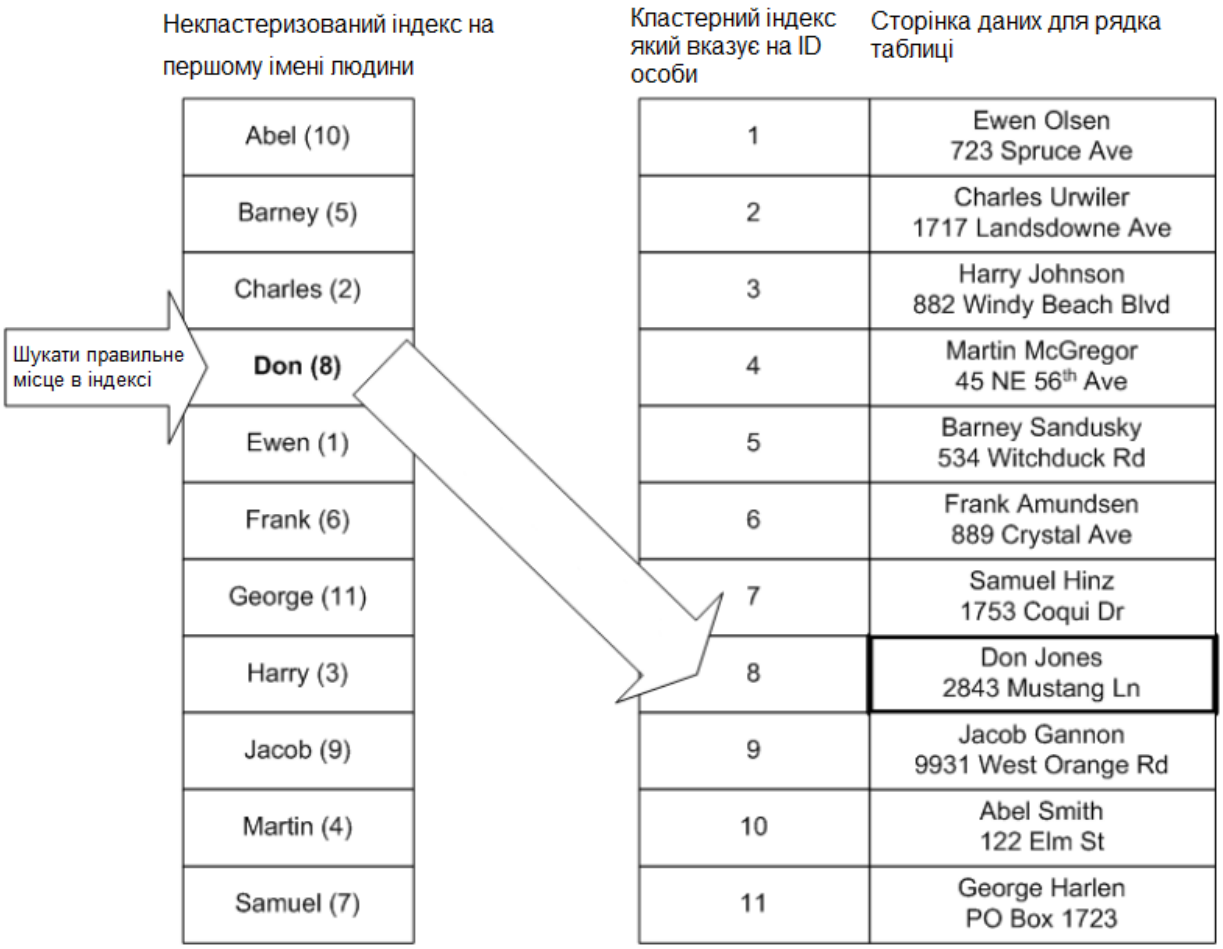


Рисунок 2.1 Взаємозв'язок між кластерними та некластерними індексами [2]

Пошуки в некластеризованому індексі все ще мають бути перехресними посиланнями на фактичні сторінки даних. Це перехресне посилання фактично зроблено до кластерного індексу таблиці.

Визначення того, які стовпці таблиці слід індексувати, безумовно, є складним стратегічним процесом. Перш ніж почати натискання індексів на таблицю, потрібно зрозуміти запити, які будуть виконуватися проти таблиці. Для початку можна застосувати декілька «великих правил», за допомогою яких можна застосувати індекси до таблиці, а

сервер містить деякі корисні інструменти, які допоможуть визначити стовпці, які потрібно індексувати, або визначити індекси, які не використовуються ефективно.

2.2.5 Двигун запитів і оптимізатор

Механізм запитів SQL є серцевиною продукту і відповідає за прийняття та виконання кожного запиту, який виконується на сервері. Сам механізм запиту складається з двох основних функцій: оптимізатора запитів і фактичного механізму виконання запитів. Незважаючи на те, що механізм запитів значною мірою є самодостатнім, оптимізатор запитів є важливим компонентом налаштування продуктивності і відкритий для маніпулювання кваліфікованим розробником або адміністратором бази даних.

Завдання оптимізатора – вивчити запити та визначити, як їх виконуватиме механізм виконання. Аналіз включає в себе індекси, які повинен використовувати движок, порядок, у якому повинні виконуватися різні пункти запиту, і так далі. Оптимізатор використовує статистику, створену SQL щодо розміру і складу таблиць і індексів, і підключає ці статистичні дані до моделі статистики. Модель виробляє розрахункові часи виконання для різних сценаріїв запитів, таких як час, необхідний для виконання запиту без використання індексу, час, необхідний для використання конкретного індексу, тощо. Модель шукає план виконання з найменшим розрахунковим часом виконання і подає цей план виконання двигуну виконання запиту [3].

Оптимізатор запитів досить розумний щодо пошуку швидких планів виконання. Він починається з вивчення запиту для будь-яких очевидних планів виконання.

```
SELECT FirstName FROM Customers
```

Наприклад, прості запити, такі як цей, часто легко оптимізувати просто тому, що вони настільки прості. Оптимізатор може розглянути вартість декількох досить очевидних планів виконання, і якщо один з цих планів має вартість, меншу, ніж визначений поріг, оптимізатор вирішить, що план є достатньо дешевим і виконає його.

Якщо жоден з дійсно очевидних планів «недостатньо дешевий», оптимізатор розглядатиме більш складні запити на основі своєї статистичної моделі. Ця друга фаза вимагає трохи більше часу, але якщо вона виробляє план, який є «досить дешевим», оптимізатор подасть план на виконання. Якщо, однак, друга фаза оптимізації не дає достатньо дешевого плану, оптимізатор почне використовувати метод оптимізації грубої сили, в якому він просто генерує кожен мислимий план виконання, поки не знайде той, який має меншу вартість ніж заданий поріг. Оптимізатор має ліміт часу для цієї третьої

фази, і якщо він не знайде «досить дешевого» плану виконання до кінця ліміту часу, він просто піде з найдешевшим планом, який він придумав раніше.

Метою всього цього процесу оптимізації є отримання найшвидшого плану виконання в найкоротші терміни. Оптимізатор визнає, що кожна наступна фаза оптимізації триває довше, тому він виконуватиме лише ті фази, якщо це займе менше часу, ніж плани запитів, які він розглянув до цього.

Проте оптимізатор не просто покладається на статистику. Наприклад, при розгляді питання про використання паралелізму, оптимізатор враховує поточний рівень використання процесора на сервері. Для багатьох з цих обчислень, залежних від ресурсів, оптимізатор приймає різні рішення при кожному виконанні запиту. Таким чином, хоча більшість рішень оптимізатора базується на статистичному моделюванні, оптимізатор впливає на конфігурацію сервера та навантаження при визначенні вартості запропонованого плану виконання [3].

2.2.6 Кешування

Операції вводу / виводу диска (I/O) є прокляттям такого продукту, як MySQL. Найчастіше дискова підсистема є вузьким місцем, що вимагає від сервера очікувати, поки інформація буде отримана з диска. СУБД включає в себе кілька функцій, таких як кешування наперед, що дозволяє зменшити негативний вплив продуктивності диска. Однак, зрештою, майже завжди існує диск, який створює проблеми. Фактично, оптимізатор запитів SQL вказує на оціночну кількість фізичних прочитань (кількість інформації, яка повинна бути прочитана з диска) у всіх витратах плану виконання запиту.

Щоб полегшити вплив дискового вводу-виводу, сервер конфігурує декілька кеш-пам'яті, в яких інформація, прочитана з диска, може бути збережена в пам'яті для подальшого використання. SQL завжди дивиться на свої кеші - через процес, який називається логічним читанням - перш ніж спробувати прочитати інформацію з диска перевіряє у кеш-пам’яті [1].

СУБД використовує кеші, щоб уникнути не тільки фізичного читання, але й тривалої оптимізації запитів. Всякий раз, коли двигун створює план виконання запиту, він кешує цей план для майбутнього використання. Будь-яке подальше використання точно такого ж запиту не вимагатиме виконання через оптимізатор запитів; замість цього сервер використовуватиме кешований план виконання. Кеші можуть включати [1]:

– збережені процедури;
– підготовлені звіти;

- спеціальні запити;
- процедури реплікації;
- тригери;
- перегляди;
- таблиці користувачів;
- системні таблиці;
- перевірки.

Кожен з цих об'єктів може зберігатися в кешах для покращення продуктивності.

2.3 Розуміння апаратних компонентів продуктивності

Зрештою, вся продуктивність зводиться до апаратних засобів. Хоча адміністратори баз даних часто налаштовують запити для кращого часу виконання, вони роблять це лише тому, що це простіше, ніж оновлення серверного обладнання. Насправді, більшість налаштувань продуктивності, які робить розробник баз даних або адміністратор, будуть робити програми більш швидшими на програмному рівні.

Оскільки всі проблеми з продуктивністю в кінцевому підсумку зводяться до апаратних засобів, потрібно знати як сервер взаємодіє з обладнанням, на якому він працює. Це допоможе вибрати краще обладнання для нових комп'ютерів і допоможе краще використовувати обладнання на існуючих. З точки зору продуктивності існують чотири основні апаратні компоненти: диски, мережі, процесори та оперативна пам'ять.

2.3.1 Диск I/O

Як вже згадувалось, пропускна здатність диска часто є вузьким місцем на комп'ютерах. Навіть якщо використовується 8 Гб оперативної пам'яті, бази даних можуть легко споживати у сто разів більше місця, тобто сервер приречений на постійний доступ до диска. Є, однак, деякі хитрощі для поліпшення дискового вводу-виводу.

Наприклад, потрібно створити рукописну копію книги на 1000 сторінок. Можна скористатися одним помічником, щоб кожен мав копію 500 сторінок, або скористатися дев'ятьма помічниками, щоб кожна людина відповідала лише за 100 сторінок. Оскільки кожен з них може копіювати фіксовану кількість сторінок у будь-який певний проміжок часу, чим більше помічників буде, тим краще. Те ж саме стосується дисків: чим більше, тим краще. Будь-який диск може передавати фіксовану кількість даних кожну секунду. Якщо треба прискорити роботу дискової підсистеми, просто збільшіть кількість дисків,

які подають дані до серверу. Один простий спосіб збільшити кількість дисків - за допомогою надлишкового масиву недорогих (або незалежних) дисків (RAID). У деяких масивах RAID, таких як RAID 5, дані фізично поширюються по дисках масиву, роблячи кожен диск відповідальним за меншу частину загального навантаження даних [4].

СУБД також надає можливість поширювати навантаження на диск на декількох (або навіть на декількох масивах RAID). Наприклад, сервер зберігає свої бази даних та журнали транзакцій в окремих файлах. Переміщення цих файлів на окремі диски (або масиви) дозволить швидше отримати доступ. Також можна розділити базу даних на декілька файлів, що дозволяє поширювати ці файли на декілька дисків (або масивів) для поліпшення продуктивності диска. На рис. 2.2 показана єдина база даних, що поширюється на три фізичні файли. Кожен файл, а також журнал транзакцій бази даних розташований на окремому диску (або масиві), збільшуючи кількість «рук», які обробляють дані.

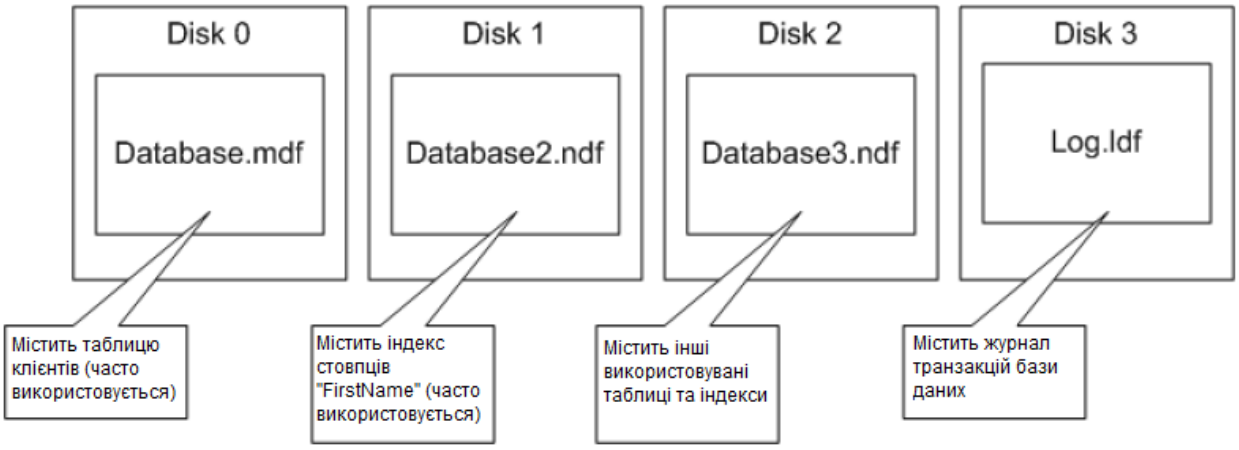


Рисунок 2.2 Одна база даних поширюється на три фізичні файли [4]

2.3.2 Мережевий I/O

Мережевий ввід / вивід - швидкість, з якою сервер може перекачувати дані у вашу мережу та отримувати дані з вашої мережі - це, як правило, не перше місце, де з’єднуються комп'ютери. Більшість компаній використовують мережу на 100 Мбіт/с або швидше, і часто встановлюють сервери навіть на більш швидких магістралях, які працюють на частоті 1 Гбіт/с. Проте, залежно від конкретних моделей використання, перевантаження мережі може зробити комп'ютер з SQL сервером схожим на те, що він працює повільніше, ніж слід. Нижче наведено кілька порад щодо створення мережі для підтримки масштабного використання [4]:

- Підключити комп'ютери з сервером до комутатора та поширити клієнтські комп'ютери через інші порти комутатора. В ідеалі, використовувати багатоканальний комутатор, який дозволяє підключатися на більш високій пропускній здатності, ніж зазвичай клієнти підключаються. Це дозволить комутатору одночасно мультиплексувати кілька клієнтських розмов з SQL сервером.
- Серверні операційні системи Windows (OS) мають максимальну пропускну здатність близько 500 Мбіт/с на серверному апаратному забезпеченні (основним обмеженням є шина PCI, що використовується мережевими адаптерами, а не сама Windows). Треба мати на увазі це обмеження, і, як тільки використання вашого сервера наблизиться до межі, треба розглянути можливість переходу на декілька серверів для обробки навантаження.
- Використовувати мережеві адаптери шини периферійного компонента (PCI), що володіють шиною, які здатні отримати доступ до пам'яті сервера без використання процесора сервера. В ідеалі треба встановити мережеві адаптери на іншу пусту шину PCI, щоб забезпечити максимальну пропускну здатність адаптера для пам'яті сервера.
- Уникайте запуску інших служб, таких як IIS або HTTP Apache, на комп'ютері з SQL сервером. Ці служби конкуруватимуть за пропускну здатність мережі і можуть штучно обмежити пропускну здатність мережі.
- Створюйте запити, які повертають мінімальну кількість інформації, тим самим мінімізуючи використання мережі.

2.3.3 Процесори

Процесори є найпоширенішими причинами проблем з продуктивністю. SQL сервер, безумовно, є важким користувачем процесора - кожен запит, виконуваний сервером, накладає певну навантаження на процесор сервера. SQL оптимізований для того, щоб скористатися перевагами декількох процесорів, а широкомасштабні реалізації часто включають 4-полосні або навіть 8-позиційні процесори [5].

Багатопроцесорні комп'ютери дозволяють серверу використовувати паралелізм, через який можна загрузити кілька процесорів для виконання одного запиту. Паралелізм часто дозволяє вносити деякі неочевидні рішення для виконання запитів. Наприклад, припустимо, що у нас є середня таблиця з кількома тисячами рядків. Ми виконуємо запит, який міг би використовувати індекс, але СУБД замість цього виконує базове сканування

таблиці. Рішення про сканування замість використання індексу здається дивним, поки ми не загадаймо про паралелізм. Двигун SQL може визначити, що швидше використовувати 8 процесорів для виконання сканування таблиць - кожен процесор ефективно приймає восьму частину таблиці, ніж витрачати час на відкриття індексу, пошук відповідного рядка і перехід на сторінку фізичних даних.

До дискового вводу-виводу, використання процесорів часто є першим вузьким місцем у продуктивності. На жаль, це також зазвичай перше вузьке місце, в якому важко щось змінити. Тому що, на відміно від дисків або мережевих адаптерів, сервери можуть приймати лише фіксовану кількість процесорів і, як правило, можуть отримувати лише певну швидкість процесора. Після того, як буде встановлено певну кількість, буде не можливо оновити підсистему процесорів. У більшості випадків це обмеження означає, що потрібно буде почати тонке налаштування операцій бази даних, щоб використовувати процесори менш важко або ефективніше.

2.3.4 Оперативна пам'ять

Пам'ять є одним з найважливіших атрибутів СУБД. Завдяки кешам, пам'ять може допомогти вирішити проблеми вводу-виводу на диску і навіть допомогти полегшити деякі симптоми надмірного використання процесора. У типовому випадку SQL сервер запитує певний об’єм пам'яті за необхідністю у операційній системі і звільняє назад, коли ця пам'ять більше не потрібна. Теоретично, така поведінка означає, що сервер має добре працювати з іншими встановленими програмами, оскільки не намагається автоматично «захопити» всю наявну пам'ять для себе. На практиці, можна побачите, що багато реалізацій SQL виробничого класу потребують всієї пам'яті, яку може надати ОС [4].

2.4 Концепції масштабування

Масштабованість є показником здатності рішення легко розширюватися, щоб задовольнити додаткове навантаження. Іншими словами, масштабованість - це просто, наскільки важко зробити щось більш ефективне. У випадку з СУБД, наприклад, масштабованість говорить, як важко буде підтримувати додаткових користувачів, розміщення додаткової бази даних або запуску більшої кількості складних запитів.

Існує два способи масштабування або розширення. Перший метод - це той, який ми схильні подумати вперше - купувати більше обладнання. Цей метод називається масштабуванням, тому що це просто модернізація існуючого обладнання для підтримки

більшої потужності. SQL добре працює в масштабних ситуаціях, але тут є свої мінуси. Більшість серверів побудовано для розміщення лише певної кількості процесорів, певної кількості пам'яті тощо. Неможливо постійно збільшувати кількість процесорів, пам’яті, тощо. Ми зажди маємо обмеження в цьому плані [6].

Інший метод масштабування – розширення (додавання додаткових серверів до рішення). Кожен новий сервер виконує ті самі завдання, що й оригінальний сервер. Веб-ферми є класичним прикладом масштабної архітектури, в якій адміністратор просто продовжує додавати веб-сервери, доки не отримаєте їх достатньо для обробки кількості користувачів, які намагаються досягти роботи веб-сайту. Всупереч поширеній думці, SQL сервер може масштабуватися, хоча це, безумовно, набагато складніше, ніж масштабування веб-ферми. У SQL масштабування означає використання федеративних баз даних, реплікації, пов'язаних серверів та інших методів, які дозволяють декільком комп'ютерам виглядати як один сервер, навіть якщо вони не розміщують ідентичні копії бази даних [6].

Масштабування іноді може бути найпростішим рішенням проблем з продуктивністю. Якщо поточна система SQL не може працювати з робочим навантаженням, треба або оновити його, або додати іншу потужність, щоб допомогти впоратися з навантаженням. Масштабування не вимагає будь-якого химерного налаштування запитів, складних процедур усунення несправностей чи будь-якого іншого. На жаль, масштабування - особливо з SQL - майже завжди дорого, тому що це передбачає придбання більшої кількості серверного обладнання. У випадку з SQL масштабування також може означати багато додаткових адміністративних накладних витрат, оскільки додається більше комп'ютерів, які потрібно підтримувати та виправляти. Отже, хоча масштабування є найпростішим рішенням проблем із продуктивністю, це не завжди найкраще рішення. Найкращим рішенням найчастіше є більш ефективне використання апаратного забезпечення, яке мається у розпорядженні компанії.

2.4.1 Цілісне вдосконалення

Завжди треба пам'ятати, що постійно треба дивитися на загальну картину, коли справа доходить до продуктивності. Багато адміністраторів потрапляють у те, що називають пасткою продуктивності - шукаючи одну маленьку деталь, яка, як тільки виправлена, помітно покращить роботу кожного аспекту. Інші люди називають це магічною кулею, яка, коли звільниться, вб'є інші проблеми з продуктивністю. Сучасні сервери є системами, що означає, що вони складаються з декількох блокуючих підсистем, які працюють разом. Зміна однієї підсистеми може просто перенести проблему

продуктивності в іншу підсистему. СУБД також є системою з кешами пам'яті, оптимізатором запитів, статистикою бази даних і багато іншого. СУБД тісно взаємодіє як з ОС, так і з серверним апаратним забезпеченням, тому зміна будь-якого конкретного компонента продуктивності має вплив на інші аспекти.

Таким чином, не можна зосередитися на індивідуальних лічильниках продуктивності в системному моніторі. Замість цього, треба дивитися на всю картину продуктивності, тому і її називають цілісною. У наступних кількох розділах буде розказано про різні компоненти, які можуть, часто незримо, сприяти проблемам продуктивності на сервері.

2.4.2 Устаткування сервера

Серверне обладнання є основою для всього, що працює на сервері. Напевно всі знайомі з деякими основами продуктивності комп'ютерного обладнання, як наприклад, наявність багатого об’єму оперативної пам'яті та швидкий процесор. Але сервери мають різні потреби обробки, ніж настільні комп'ютери; таким чином, серверне обладнання повинно бути оптимізовано трохи інакше. Ось деякі речі, які потрібно шукати на високо потужному сервері:

- Серверно-оптимізовані процесори (наприклад, серія Intel Pentium Xeon) – на відміну від процесорів, орієнтованих на десктоп, процесори, оптимізовані для сервера, орієнтуються менше на прискорення мультимедіа і більше на паралельну обробку інструкцій і симетричну багатопроцесорну обробку, на якій базується ОС.
- Процесори з швидким, великим кешем другого рівня (L2) мають більш швидкий доступ до пам'яті і витрачають менше часу на очікування передачі даних з пам'яті. Навіть якщо зберігається лише тисячна частка секунди, вона може скластись дуже швидко в хвилини та години.
- Майже всі сервери використовують кешування пам'яті для швидкого доступу до пам'яті, але існують різні типи архітектури кешу. Поточні кеші вимагають, щоб процесор розмістив запит з кешем, який повинен мати доступ до основної пам'яті незалежно від того, чи запитані дані не знаходяться в кеші. Незважаючи на те, що кеш-пам'ять трохи повільніше, ніж кеш для перегляду, кеш-пам'ять запобігає використання непотрібних запитів до основної пам'яті.
- Сервери повинні включати пам'ять, що виправляє помилки, що дозволяє контролерам пам'яті сервера автоматично виправляти багатобітові помилки без необхідності відпрацьовувати процесорні цикли, вимагаючи пам'яті. Проста

паритетна пам'ять, знайдена на більшості настільних комп'ютерів, повідомить контролеру, що помилка присутня, але все ще вимагає, щоб контролер виправив пам'ять, щоб отримати правильні дані.

– Високошвидкісні дискові контролери з вбудованими кеш-накопичувачами можуть значно поліпшити продуктивність сервера. Ці контролери приймають дані з ОС так швидко, як це дозволить архітектура шини PCI, дозволяючи операційній системі (і, отже, SQL серверу) швидко перейти до інших завдань. Потім контролер записує дані на диск так швидко, як дозволять диски. Вбудована батарея контролера гарантує, що дані не будуть втрачені, якщо живлення вийде з ладу перед тим, як кеш опуститься.

2.4.3 Параметри програмного забезпечення

Можна знизити продуктивність SQL за допомогою начебто невинних дій. Наприклад, кожна база даних має опцію конфігурації Auto Close або тощо. Як правило, СУБД постійно зберігає файли баз даних відкритими (тому не можливо використовувати звичайну утиліту резервного копіювання файлів для баз даних). Однак, якщо ввімкнено функцію автоматичного закриття, сервер закриває дискові файли бази даних, коли останній користувач, підключений до бази даних, роз'єднується. Коли наступний користувач намагається отримати доступ до бази даних, сервер автоматично відкриває файл. Цей процес накладає чимало накладних витрат на СУБД. Крім того, в базі даних, до якої регулярно звертаються лише декілька користувачів, функція автоматичного закриття може викликати значно повільнішу роботу [1].

Інші типи програмного забезпечення можуть негативно вплинути на продуктивність з часом. Наприклад, одним з параметрів конфігурації індексу є його коефіцієнт заповнення. Подібно до рядків бази даних, індекси зберігаються на 8 КБ сторінках. При створенні нового індексу можна вказати коефіцієнт заповнення у відсотках: 50 залишає половину кожної сторінки індексу (близько 4 Кб) порожньою. Візьмемо для прикладу місцеву телефонну книгу. Це індекс, створений зі 100-відсотковим коефіцієнтом заповнення - на жодній сторінці немає вільного місця. Коли хтось новий переміщується по сусідству, ми не можемо ввести його або його ім'я безпосередньо на потрібній сторінці. Замість цього потрібно перевернути порожню сторінку в задній частині книги та написати назву нової особи. Ми також повинні внести нотації на правильній сторінці, щоб більше інформації знаходилося на задній частині книги. Тепер у нас є розділена сторінка або фрагментований індекс, на якому інформація більше не

знаходиться в потрібному порядку (в даному випадку, за абеткою). Використання фрагментованих індексів набагато повільніше, тому сервер дозволяє вказувати коефіцієнт заповнення. Якщо у телефонній книзі є коефіцієнт заповнення 80, кожна сторінка буде 20% порожньою, що дозволить зберігати багато місця для запису інформації про нових сусідів, не розбиваючи сторінку та переходячи до задньої частини книги. Звичайно, індекс буде набагато більшим внаслідок всього вільного простору.

Можна виявити, що певні так звані «особливості» фактично зменшують продуктивність. Компанія створила автоматизоване завдання для оновлення статистики пізно ввечері один раз на тиждень (що відображає кількість змін, які вони вносять до бази даних щодня). Компанія виявила, що деякі незрозумілі, середньострокові проблеми з продуктивністю пішли. Вони могли бути викликані швидким перервою серверу, щоб оновити статистику бази даних у середині дня. Інший клієнт виявив, що сервер розробляв дуже погані плани виконання запитів на своєму 8-процесорному сервері. Відключення функції паралельності призвело до того, що клієнт очікував швидше виконання планів, вказуючи, що оптимізатор запитів SQL недооцінював вартість використання паралелізму, коли розглядав різні плани виконання.

2.4.4 Проблеми проектування

Дизайн бази даних може знищити продуктивність SQL. Наприклад, є компанія з нерухомості, яка зберігає, серед іншого, інформацію про адресу. Адреси, очевидно, важливі для компаній з нерухомості, оскільки адреси являють собою продукти компанії (будинки та нерухомість для продажу). Компанія створила базу даних, яка використовувала не менше дев'яти таблиць для представлення однієї адреси, розбиваючи такі елементи, як тип вулиці (дорога, вулиця, проспект тощо) на одну таблицю, напрямок вулиці (N, NW і т.д.) в іншу таблицю тощо. Мета клієнта полягала в забезпеченні того, щоб користувачі могли вибирати лише з варіантів, включених до таблиць, які покращили якість даних у базі даних.

Таким чином, ідея гарна. Звичайно, можна поліпшити якість даних, надаючи користувачам розкривні списки для таких елементів, як дорога, вулиця, проспект тощо. Дозволити користувачам просто вручну вводити цю інформацію неминуче призведе до помилок, таких як помилка на шляху дорожнього руху або аеродром для проспекту. Крім того, витягання вмісту цих розкривних списків з таблиці бази даних дозволяє легко додавати нову інформацію, коли це необхідно. Однак всю цю інформацію слід денормалізувати в єдину таблицю для остаточної адреси. Таким чином, SQL може

запитувати адресну інформацію з однієї таблиці, замість того, щоб приєднуватися до дев'яти таблиць.

Цілком прийнятно зберігати денормалізовану інформацію в різних стовпцях. Наприклад, утримання дороги в іншому стовпці від номера вулиці та імені полегшить оновлення рядка в майбутньому. Єдине, коли є сенс зберігати інформацію нормалізованою, це коли вона може змінитися, і не потрібно, щоб вона мінялася на мільйонах різних місць. Наприклад, у базі даних для обробки замовлень не можна копіювати ім'я клієнта в кожному замовленні, оскільки ім'я клієнта може змінюватися (наприклад, якщо клієнт одружується). Якщо залишити ім'я нормалізованого в окремій таблиці, усі замовлення клієнта залишаться пов'язаними з цим клієнтом, навіть якщо зміна імені відбувається в майбутньому. Нормалізація також забороняє змінювати кожну з попередніх замовлень клієнта, коли відбувається зміна імені. Однак, якщо інформація не може бути змінена, можна покращити продуктивність SQL, денормалізуючи дані та усунувши приєднання таблиць до продуктивності. Треба зберігати таблицю прийнятних значень (називається таблицею пошуку), оскільки вона покращує якість даних, але не змушує приєднуватися до таблиці пошуку лише для отримання такої інформації.

2.4.5 Рекомендації для клієнтів

Додатки можуть викликати багато проблем з продуктивністю, які майже не мають нічого спільного з SQL. Нижче наведено деякі найпоширеніші практики, які демонструють клієнтські програми:

- Клієнтські програми іноді розробляються для запиту більшої кількості інформації, ніж їм потрібно. Наприклад, для клієнтських додатків не рідко запитувати сотні рядків даних, а потім шукати ці рядки, щоб знайти кілька рядків, з якими вони дійсно повинні працювати. Можливо, швидше, щоб клієнт вказав більш точний запит і надіслав лише дані, які клієнт дійсно потребує.
- Створення дворівневої (клієнт / сервер) програми для обслуговування тисяч користувачів не дозволяє краще використовувати ресурси SQL. Зрештою, більшість клієнтів не потребують постійного підключення до серверу, що вимагає, щоб сервер відкладав невелику частину пам'яті для кожного відкритого підключення, незалежно від того, чи виконує це з'єднання. Краща архітектура є багаторівневою, в якій клієнти звертаються до сервера середнього рівня для запиту даних. Сервер середнього рівня може відкривати кілька з'єднань з SQL сервер і використовувати безкоштовне з'єднання для отримання даних від імені клієнтів.

Цей метод називається пулу з'єднань і є дуже ефективним способом дозволити великій кількості клієнтів отримати доступ до одного комп'ютера SQL без негативного впливу на продуктивність.

– Загалом, SQL може виконувати збережені процедури набагато швидше, ніж спеціальні запити, тому що зберігає плани виконання збережених процедур, тобто вони не повинні виконуватися через оптимізатор запитів кожного разу, коли вони виконуються. Клієнтські програми повинні максимально взаємодіяти з SQL через збережені процедури.

Очевидно, що це лише деякі з багатьох міркувань, пов'язаних із продуктивністю, які можна перевірити у клієнтських програмах.

2.5 Підсумок найважливіших методів оптимізації

Одна з проблем, що найбільш часто зустрічається в робочих системах - повільна робота у місцях функціонування базових операції. Однією з таких операцій є пошук. Цей крок зазвичай ініціює бізнес-процес. Під час тестів, які зазвичай проводяться на невеликій базі даних, ця проблема не розкривається і відповідно запобіжні заходи не виправлять цю ситуацію.

Розглянемо процедуру пошуку клієнтів у базі даних. З розвитком системи кількість клієнтів в базі даних значно зросла, що вплинуло на час, необхідний для пошуку певного запису.

Наприклад, ми маємо базу даних, в якій зберігається приблизно 1 000 000 даних про клієнтів. Ідентифікація клієнта може здійснюватися за допомогою імені або чисел. Проблема занадто довгих запитів була виявлена під час пошуку за ідентифікаційними номерами. Таблиця 2.1 описує структуру таблиці «CLIENTS» в базі даних.

Таблиця 2.1 Структура таблиці CLIENTS

Назва колонки	Тип колонки	Обмеження
ID	NUMBER(10)	Первинний ключ
NAME	VARCHAR(30)	-
SURNAME	VARCHAR(30)	-
AGE	NUMBER(3)	-
ID_DOC_NO	NUMBER(20)	Зовнішній ключ що посилається на DOCUMENTS

Конструкція SELECT для пошуку по таблиці CLIENTS може бути записана наступним чином:

```
SELECT * FROM CLIENTS WHERE ID = :CLIENT_ID;
```

Відповідно план запитів, представлений на рисунку 2.3, та статистичні запити були зібрані в першому рядку таблиці 2.2.

Як можна помітити, план запиту не є оптимальним, тому що для кожного запиту потрібен двигун бази даних для повного сканування таблиці.

Це впливає на велику кількість дискових зчитувань і на використовуванні буферу. Варто зазначити, що незважаючи на те, що час виконання не довгий, отримані статистичні дані не є задовільними, оскільки слід враховувати кількість запитів на день та одночасні тривалі сесії. Крім того, зростання кількості клієнтів призведе до деградації показників.

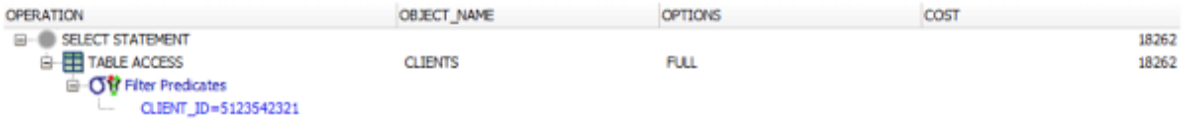


Рисунок 2.3 План запиту для конструкції SELECT без використання індексів [1]

Таблиця 2.2 Статистика запитів для таблиці CLIENTS

Тип запиту	**Час (сек.)**	**Зчитування диску**	**Використання буферу**
Без використання індексів	1.8	64 630	128 771
З індексами	0.01	86	12

Для покращення ефективності пошуку єдиним можливим рішенням є встановлення індексів в стовпці, на якому виконується пошук, за допомогою наступного синтаксису:

```
CREATE INDEX CLIENTS_INDX1 ON CLIENTS(NAME);
```

Це рішення не призводить до зміни логіки застосування. Побічними ефектами створення індексів є: додаткове місце на диску та збільшення часу модифікації та додавання нових елементів. Проте завантаження нових даних в систему, як правило, виконується у фоновому режимі, а модифікація їх відбувається досить рідко. Цей підхід не викликає помітного або різкого падіння продуктивності для інших операцій.

План запиту для пошуку в таблиці CLIENTS з використанням визначеного індексу був змінений оптимізатором, а кількість запитів була зменшена в 70 разів. Середня тривалість запитів після введення індексу для незалежних операцій пошуку показана у другому рядку таблиці 2.2.

Оптимізація запитів шляхом побудови індексів призвела до зменшення в 700 разів читання буфера для одержання однакових результатів. Єдиним побічним ефектом додавання нового індексу був дисковий простір - 204 Мб.

Інше питання, що розглядається як проблема трапляється при використанні реляційної системи бази даних. Це об'єднання двох або більше великих таблиць.

Звіт про продажі, згрупованих за віком клієнтів базується на таблицях «CLIENTS» (див. таблицю 2.1) та «SALES_HISTORY» (див. таблицю 2.3), що містить близько 4 000 000 записів. Приєднана операція базується на колонці «CLIENT_ID», яка вказується в таблиці «CLIENTS». Звіт зазвичай створюється для віку від 15 років. Під час виконання запиту було помічено що на обробку потрібно забагато часу, і тому необхідно провести оптимізацію.

Таблиця 2.3 Структура таблиці SALES_HISTORY

Назва колонки	**Тип колонки**	**Обмеження**
ID	NUMBER(10)	Первинний ключ
SALE_DATE	DATE	-
CLIENT_ID	NUMBER (10)	Зовнішній ключ що посилається на CLIENTS
TOTAL_AMOUNT	NUMBER (10,4)	-

Щоб прискорити операцію приєднання, можна зменшити вхідні параметри звіту. [2] Тобто змінити нижню та верхню межу віку або часові рамки. Такий підхід змінює функціональність системи і тому має розглядатися як крайній засіб. Більше того, це не подолає проблему, а просто зменшує вплив факторів, які її формують.

Для підвищення ефективності запиту аналізуються характеристики даних, які зазвичай вибираються для запиту. Було встановлено, що вік розподілу клієнтів зазвичай є однорідним і становить від 20 до 40 років. Тому можна припустити, що у звіті використовується понад 50% відсотків таблиці «CLIENTS» для визначення діапазону клієнтів. З огляду на характеристику даних, план запитів повинен бути проаналізований, щоб визначити, чи є він оптимальним чи ні.

```
SELECT * FROM CLIENTS CL, SALES_HISTORY SH WHERE CL.CLIENT_ID =
        SH.CLIENT_ID AND CL.AGE >= 22 AND CL.AGE <= 36;
```

Для більш повного аналізу пропонується розглянути результати виконання операції EXPLAIN. Ця операція використовується в різних реляційних базах даних і містить інформацію про те, як SQL двигун виконує запит.

Таблиця 2.3 Результат виконання операції EXPLAIN

ID	Select type	Table	Type	Possible keys	Key	Key len	Ref	Rows	Extra
1	SIMPLE	CL	ALL	NULL	NULL	NULL	NULL	1	Using where
1	SIMPLE	SH	ALL	NULL	NULL	NULL	NULL	1	Using where; Using join buffer (flat, BNL join)

Перший рядок таблиці 2.4 відображає час обробки запитів, зчитування диску та використання буфера.

Таблиця 2.4 Статистика запиту для з'єднання таблиць CLIENTS та SALES_HISTORY

Тип запиту	Час, сек	Зчитування диску	Використ. буферу	Вартість запиту
Без індексів	240	518 485	6 311 813	70 345 256
З індексами	120	165 061	1 485 492	32 315 199
З HASH індексом	33	223 673	165 196	186 981

Давайте розглянемо створення індексу таблиці SALES_HISTORY, щоб запобігти повному доступу до таблиці. Для цього потрібно впевнитися що CLIENT_ID є унікальним і зв'язаний з таблицею CLIENT та атрибутом ID. Створення індексу зменшило вартість і час запиту, а також зчитування диску та використання буферу, але вони залишаються дуже високими. Це пов'язано з специфікою даних, оскільки це приєднання відноситься до більш ніж 50% даних. Крім того, індекс займає 740 Мб дискового простору, і тому будь-які оновлення можуть бути дуже помітними.

Розглядаючи план запитів та характеристики даних, на яких він працює, ми можемо констатувати, що проблема полягає в об'єднанні вкладеного циклу. Запит

виконується на великих обсягах даних, і тому приєднання по хешу виглядає більш оптимальним. Щоб оптимізувати такий запит, було використано HASH індекс, при цьому вартість запиту значно знизилася. Треба відмітити що використання такого типу індексів можливе тільки для наступних типів двигуна БД: MEMORY/HEAP, NDB.

Як бачимо, незважаючи на показники, двигун бази даних вибрав повну перевірку обох таблиць [4]. Це найбільш сприятливе, оскільки в більшості випадків ми використовуємо більше 50% даних, тому більша частина таблиці завантажується в пам'ять з блоків даних. Якщо оптимізатор використовував індекс, йому потрібно буде виконати повне сканування, а потім передавати дані за рядком. Як результат, вся таблиця буде завантажена в пам'ять, а час буде значно збільшуватися [2]. Отже, індекс CLIENT_ID, визначений для таблиці SALES_HISTORY, непотрібний і може бути видалений.

Порівняно з варіантом з індексом, на 1/3 більше загальних дискових зчитувань, але кількість використання буферу зменшується в 10 разів, що сприяє збільшенню часу виконання запиту. З точки зору бізнесу, рішення не викликає жодних ризиків, оскільки в логічному слою запити не були змінені.

Проблема періодичного падіння продуктивності дуже поширена в контексті існуючих систем. Вони можуть відрізнятися, тому важливо точно визначити, яка операція в базі даних є проблемною. Експериментальні дослідження показали який запит є проблемним. Процедура GET_QTY відповідальна за тимчасові затримки. Процедурний кодекс виглядає таким чином:

```
CREATE OR REPLACE PROCEDURE GET_QTY(store_number IN NUMBER, product IN
NUMBER, qty OUT NUMBER)
AS
BEGIN
     SELECT qty INTO qty FROM STORE_ITEM
     WHERE store_id = store_number AND product_id = product;
END;
```

Процедура виконує лише один запит у таблиці STORE_ITEM. Структура таблиці представлена в таблиці 2.5. Було створено індекс STORE_ITEM_I1. Зміст таблиці містить близько 600 000 записів даних.

Аналіз проблеми, виконаної на момент дослідження показав, що план запиту був виконаний несприятливим чином. У цьому випадку оптимізатор використовував індекс, який був ефективним для невеликих частин табличних даних. На жаль, інший пошук для більшої вибірки також використовував індекс, оскільки запит вже зберігався в кеш-пам'яті бази даних.

Таблиця 2.5 Структура таблиці STORE_ITEM

Назва колонки	Тип колонки	Обмеження
STORE_ID	NUMBER(10)	Унікальний індекс створений для STORE_ITEM_I1
PRODUCT_ID	NUMBER(20)	Зовнішній ключ, що посилається на табл. RODUCTS
QTY	NUMBER(30)	-

Якщо запит не був розміщений у кеш-пам'яті бази даних, а план був встановлений з нуля, існували великі шанси, що план буде правильно налаштовано, і сама операція триватиме довго. Можливо, при проектуванні цього розділу системи передбачалося, що розподіл продуктів у магазинах рівномірно розподілений і характеризується високою селективністю, і тому були створені індекси STORE_ID та STORE_ITEM_I1. Однак, це показало, що припущення було неправильним і, в деяких випадках, можуть спричинити більше проблем, ніж переваг.

Ми можемо розглянути три різні способи вирішення цієї проблеми:

- видалити проблемний індекс STORE_ITEM_I1;
- примусити робити повне сканування таблиці для кожного оператора вибору;
- створення нового індексу, який буде більш придатним для всіх виділених заяв.

Перше з наведених вище рішень є найбільш трудомістким та небезпечним у контексті існуючої системи, оскільки видалення індексу в складних системах може спричинити проблеми продуктивності та інтеграції в інших частинах системи. Оскільки виробничі системи, як правило, є багатофункціональні, і їх розробка та підтримка потребує багатьох людей, цей підхід слід відхилити [7].

Примусове сканування таблиці не веде до комерційного ризику. Це рішення повністю виключить проблему продуктивності. Проте повне сканування збільшить час вибору з меншого набору даних. Тим не менш, час підйому від 0,003 до 0,26 секунди буде незначним [5]. Навіть якщо число користувачів збільшиться, двигун MySQL досить розумний, щоб кешірувати подібні, досить навантажені, запити.

Найкращим варіантом може бути створення індексу, який включатиме стовпці STORE_ID і PRODUCT_ID [1].

Можна помітити, що індекси мають помітний вплив на час виконання, а також зчитування диска та використання буферу. Вартість запиту значно зменшилася через те, що за допомогою індексу двигун бази даних точно знає, які блоки даних повинні посилатись на завантаження отриманих даних. Тому з диска відсутні надмірні зчитування.

Однак, враховуючи те, що пошукові операції виконуються багаторазово в системі, можна стверджувати, що завантаження всієї таблиці також є корисним. Це пов'язано з тим, що кожен наступний пошук, скоріш за все, скористається таблицею, яка вже завантажена в буфер, або, в крайньому випадку, зчитає лише невелику частину таблиці з диска.

Пакетна обробка - це дуже поширена проблема в робочих системах. Вона зосереджується на завантаженні та, в деяких випадках, обробці вхідних даних з зовнішніх систем. Обсяги даних часто дуже великі, і їх завантаження або обробка повинні бути закінчені протягом короткого часу.

Розглянемо процес, який завантажує зовнішні дані, доповнені певними атрибутами, на основі цільової системи. Оскільки обсяг вхідних даних та таблиці великий, потрібна оптимізація. Варто зазначити, що дані не є критичними з точки зору бізнесу. Крім того, невелика зміна логіки прийнятна, оскільки дані перевіряються після завантаження.

У системі є 1 000 000 найменувань і 10 місць. Кожен елемент є в ієрархії і має батька. Багато предметів може мати одного й того ж батька. Структурні таблиці представлені у таблицях 2.7, 2.8 та 2.9. Щомісяця завантажується файл, в якому входять всі елементи з кількістю одиниць, що продаються з кожного місця. Ці елементи призначені для будь-якого батька, і їх необхідно об'єднати з відповідними записами з таблиці ITEMS. Неможливо виконати одну операцію UPDATE, оскільки ця система отримує щомісяця близько 4 000 000 даних.

Таблиця 2.7 Структура таблиці ITEMS

Назва колонки	**Тип колонки**
ITEM_ID	NUMBER(20)
ITEM_PARENT	NUMBER(20)
DESCRIPTION	VARCHAR(200)

Таблиця 2.8 Структура таблиці LOCATIONS

Назва колонки	**Тип колонки**
ITEM_ID	NUMBER(20)
ITEM_PARENT	NUMBER(20)
DESCRIPTION	VARCHAR(200)

Запит, що оновлює дані, можна записати таким чином:

```
UPDATE EXTERNAL_SALE scp SET ITEM_PARENT = (SELECT ITEM_PARENT FROM
                ITEMS I WHERE I.ITEM_ID = SCP.ITEM_ID)
```

Таблиця 2.9 Структура таблиці EXTERNAL_SALE

Назва колонки	**Тип колонки**
ITEM_ID	NUMBER(20)
LOCATION	NUMBER(20)
ITEM_PARENT	NUMBER(20)
QTY	NUMBER(3)

Запит, виконаний для повного діапазону даних 40 000 000, не закінчився протягом 70 хвилин. Подальше очікування виконання запиту марно і не може бути продовжено. Крім того, він несе ризик помилки бази даних через відсутність табличного простору.

Одне з рішень полягає в тому, щоб розділити весь діапазон даних для менших підмножин і повторити одну й ту ж дію для кожного з них. Вся операція була розділена на частини з будівництва так званих BULK COLLECT. Крім того, пропозиція FORALL використовується для того, щоб виконати оновлення всієї підмножини, і COMMIT додано для запобігання проблемам з відмовою табличного простору. [2] Код програмування був визначений таким чином:

```
DECLARE
      CNT NUMBER := 1;
      CURSOR SCOPE_CUR IS
      (SELECT ROWID, ITEM_ID FROM EXTERNAL_SALE);
      TYPE rowid_type IS TABLE OF rowid;
      ROWID_TAB ROWID_TYPE;
      TYPE ITEM_TYPE IS TABLE OF
      EXTERNAL_SALE.ITEM_ID%TYPE INDEX BY PLS_INTEGER;
      ITEM_TAB ITEM_TYPE;
BEGIN
      OPEN SCOPE_CUR;
      LOOP
      FETCH SCOPE_CUR BULK COLLECT
      INTO ROWID_TAB, ITEM_TAB LIMIT 100000;
      FORALL I IN ROWID_TAB.FIRST..ROWID_TAB.LAST
      UPDATE external_sale scp SET ITEM_PARENT =
      (SELECT ITEM_PARENT FROM ITEMS I
      WHERE I.ITEM_ID = item_tab(i))
      WHERE SCP.ROWID = ROWID_TAB(I);
      IF(MOD(CNT, 10) = 0) THEN
      COMMIT;
      END IF;
      CNT := CNT + 1;
      EXIT WHEN ROWID_TAB.COUNT = 0;
      END LOOP;
      COMMIT;
END;
```

Таблиця 10 показує статистику оновлення даних, що виконуються за допомогою єдиного виразу SQL та BULK COLLECT.

Таблиця 2.10 Статистика UPDATE запиту

Тип	Час виконання	Зчитування диску	Використання буферу
UPDATE	4200 (незакінч.)	-	-
BULK COLLECT	960	460 596	126 939 278

Можна помітити, переглядаючи таблицю 10, що оптимізована команда BULK COLLECT, читає необхідні дані з диску, а потім обробляє їх за допомогою буферу [1].

2.6 Висновки до другого розділу

Налаштування продуктивності є фінансово мотивованою процедурою. Користувачі скаржаться, що база даних повільна, тому IT-персонал може додати більше пам'яті або декілька додаткових процесорів, які можуть допомогти. Що робити, якщо просто потрібен новий індекс на великій таблиці? Що робити, якщо додавання додаткового обладнання навіть не вирішить проблему? Чітке розуміння принципів налаштування продуктивності бази даних може допомогти уникнути дорогих оновлень серверів, коли просто потрібна конфігурація програми або бази даних. Цей розділ надав розуміння того, в чому полягає справжня проблема і що можна зробити з ними. Налаштування продуктивності може здатися досить важким та довгим процесом, але це надасть збереження компанії від непотрібного оновлення обладнання.

2.7 Література до другого розділу

1. Peter Gulutzan, Trudy Pelzer. SQL Performance Tuning (1st Edition). – M.: 2012.
2. David Kroenke, David Auer. Database Concepts (3rd Edition). – M.: , 2007.
3. Hugh E Williams. Web Database Applications with PHP & MySQL. – M.: , 2002.
4. Koper, R., Analysis of the database optimization methods in the context of existing systems, Master's thesis, Lodz University of Technology, 2015.
5. Michael Widenius. MySQL Reference Manual. – M.: , 2002.
6. Powell, G., Oracle Performance Tuning for 10gR2, Elsevier Inc., 2007.
7. Benjamin Nevarez, High Performance SQL Server, Apress Inc., 2016.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗГЛЯНУТИХ ЗАСОБІВ ОПТИМІЗАЦІЇ

3.1 Проектування системи

3.1.1 Опис процесу розробки

Початком проекту можна назвати підготовчий етап. Мета цього етапу полягала в тому, щоб на основі розглянутих пропозицій створити деяку концепцію майбутньої системи. Підштовхуючись від неї провести деяку оцінку затребуваності і можливості реалізації проекту.

Потім проект, концепція якого виглядала задовільною для реалізації, виходить на етап розробки вимог. На цьому етапі виконавець розвинув перелік усіх явних і прихованих вимог. Мішенями цього етапу є виявлення всіх прихованих потреб, вирішення конфліктів вимог, формування монолітного технічного рішення та аналіз реалізованості підготовленого рішення.

Коли технічне рішення було виявлено, наступний етап полягав в розробці архітектури майбутньої системи. Мета цієї стадії – формування логічної і фізичної архітектури, яка цілком покриває всі вимоги. Під час розробки архітектури проводилося рецензування та уточнення концепції, вимог і попереднього технічного рішення, що дало можливість за стерегтися від найбільш небезпечних ризиків.

Після завершення проектування архітектури варто було знову провести інспектування основних параметрів проекту і вирішити, чи існує можливість закінчити проект. В момент коли баланс було знайдено, і вдалося створити допустиму архітектуру системи, можна переходити до реалізації і постачання системи. Також у цих умовах потрібно було ділити реалізацію на декілька стадій, щоб в кінці кожного розробник мав готовий до постачання продукт. При цьому найважливіший, основний елемент повинен був розроблятися на ранніх етапах, а надбудови, що функціонують поверх цих основних компонентів, слід реалізовувати останніми. У такому випадку найбільш ризиковані для системи помилки були виправлені на перших етапах, і ризик того, що прикладна функціональність системи буде реалізована на нестабільній основі, було зменшено.

Після доставки повністю завершеною системи проект зазвичай переходить до етапу експериментальної експлуатації. Мета цього етапу полягає у випробуванні якості роботи розробленої системи в справжніх умовах експлуатації. Найбільш часто, на цьому етапі виконавець спільно з замовником проводить вимір числових метрик, що дозволяють

розкрити якість створеної системи. В першу чергу перевіряються функціональні характеристики якості, потім - не функціональні. У разі наявності незбіжностей виконавець виправляє код системи [2].

Цілковито налагоджена і налаштована система включається в промислову експлуатацію. Як правило, виконавець зобов'язаний супроводжувати систему впродовж терміну гарантії. Виявленні невідповідності повинні бути виправленими. Користувачі і обслуговуючий персонал замовника повинні одержувати оперативну консультативну підтримку.

Кожний проект, в кінцевому рахунку, приходить до свого логічного закінчення. Етап припинення проекту має на меті аналіз результатів, внесення змін в процес розробки на основі отриманого досвіду і доповнення бази знань розробників новими ефективними рішеннями, а також новими готовими компонентами, які можна буде використовувати в майбутніх проектах.

3.1.2 Опис структури програмного засобу

Виходячи з поставлених задач, можна побачити, що система повинна бути розділена на деяку підмножину модулів, а саме:

Перша і найважливіша з них - система отримання, обробки і поділу даних, які надають туристичні оператори своїм турагентам. Саме наслідок її діяльності зможуть споживати інші підсистеми.

Інша підсистема надає можливість зміни даних адміністратором. У цьому може з'явитися необхідність в момент оптимізації процесу залучення клієнтів. Цей модуль вже є веб-інтерфейсом на відміну від останнього.

Остання частина системи так само є веб-інтерфейсом, але вже призначеним для роботи з потенційними клієнтами. Його головна ціль - наддати комфортний спосіб отримання інформації.

Конструкція даної програмної системи в загальному вигляді представляє собою набір мікросервісів, які реалізують такі завдання:

- сервіс отримання даних від туроператорів;
- сервіс для взаємодії з клієнтами;
- сервіс для взаємодії з адміністратором;
- сервіс, що надає спільні дані, використовувані для двох вище зазначених сервісів.

На рис. 3.1 представлено основні напрямки взаємодій між сервісами. Стрілка показує, звідки і куди проходить передача даних (також дані можуть йти і в зворотному порядку).

Рисунок 3.1 Напрямки взаємодій між сервісами

Тут зображено, що один сервіс збирає дані в базі даних, а всі інші лише користуються результатом роботи останнього. Безумовно, тут є попередження, база даних може містити дані, які необхідно модифікувати користувачам.

3.1.3 Проектування схеми і опис бази даних

База даних представляє собою поіменний набір структурованих даних, що співвідносяться до конкретної предметної області. Під предметною областю прийнято мати на увазі частину реального світу, що піддається вивченню для організації управління і автоматизації виробництва. Аналіз предметної області дозволяє визначити, які дані знаходяться в базі даних. Користувачами БД можуть бути різноманітні прикладні програми, програми-комплекси, а також спеціалісти предметної області, які іменуються кінцевими користувачами.

Модель предметної області - це наші знання про предметну область. Знання можуть бути як у виді неформального знання, так і проявлятися формально за допомогою будь-яких засобів [1]. В якості таких засобів можуть використовуватися текстові описи предметної області, набори посадових інструкцій, правила ведення бізнесу в компанії і

т.д. Досвід показує, що текстовий засіб представлення моделі предметної області вельми неефективний. Набагато більш інформативними і вигідним в розробці баз даних є описи предметної області, що виконуються за допомогою спеціалізованих графічних відміток. Існує величезна кількість методів опису предметної області.

Модель предметної області окреслює швидше процеси, що проходять в предметній області і дані, використовувані цими процесами. Від того, наскільки правильно змодельована предметна область, залежить успіх наступної розробки програми, а також успіх в метаморфозах, проведених для наступних версій програмної системи.

Інфологічна модель - це опис предметної області, засновано на аналізі семантики об'єктів і явищ, зроблених без орієнтації на вживання в майбутньому програмного забезпечення або технічних засобів обчислювальної техніки [3].

Так як система складається з основної бізнес логіки - робота з даними туру, а так само допоміжної (контроль доступу до панелі управління, організація «міграцій» і т.п.) було прийнято рішення по розподілу моделі на два цих об'єкта.

Інфологічна модель для даних БД проектувалася, як модель «сутність-зв'язок». Одна з основних цілей полягає в тому, щоб наслідки аналізу предметної області були відображені в досить простому, наочному, але в той же час формалізованому та досить інформативному виді.

Виділимо основні елементі сутності для даної бази даних:

- сутність «тур»;
- сутність «туроператор»;
- сутність «готель» або «пансіонат»;
- сутність «ресурс»;
- сутність «курорт (місто)»;
- сутність «місто відправлення»;
- сутність «перевізник»;
- сутність «користувач»;
- сутність «міграція».

Основний сутністю в даній предметній області є «тур». Тур розміщає зв'язку, що свідчать на інші сутності які його визначують.

Сутність перевізник включає інформацію про перевезення клієнтів в певний тур. Перевізник в свою чергу обслужує певний тур.

Туроператор розміщує безліч турів, і в свою чергу тур припадає строго до одного туроператору.

Місто відправлення засвідчує, з якого строго чіткого міста (з безлічі) проводиться вибуття. У цій реалізації це буде постійно одне місце, але варто врахувати подальше зростання потреб системи.

Сутність готель характеризує заклад який приймає клієнта. Це може бути як рядовий готель, так і пансіонат. Дана сутність бере участь в прийомі різноманітних турів.

Під ресурсами мається на увазі файли, які зберігаються на сервері для різних значимостей. Так, наприклад, для готелів це можуть бути фотографії.

У свою чергу існує «курорт», який визначає місце знаходження готелю і, отже, туру. Курорт так само можна охрестити містом прибуття. Ці поняття є ідентичними в даній предметній області.

Такі суті як «користувач» і «міграції» в головному забезпечують роботу панелі адміністратора. Думаю, не варто зациклюватися на їх задачах. Зазначу лише що «міграції» необхідні для організації подальшої модифікації і версіонуванні баз даних.

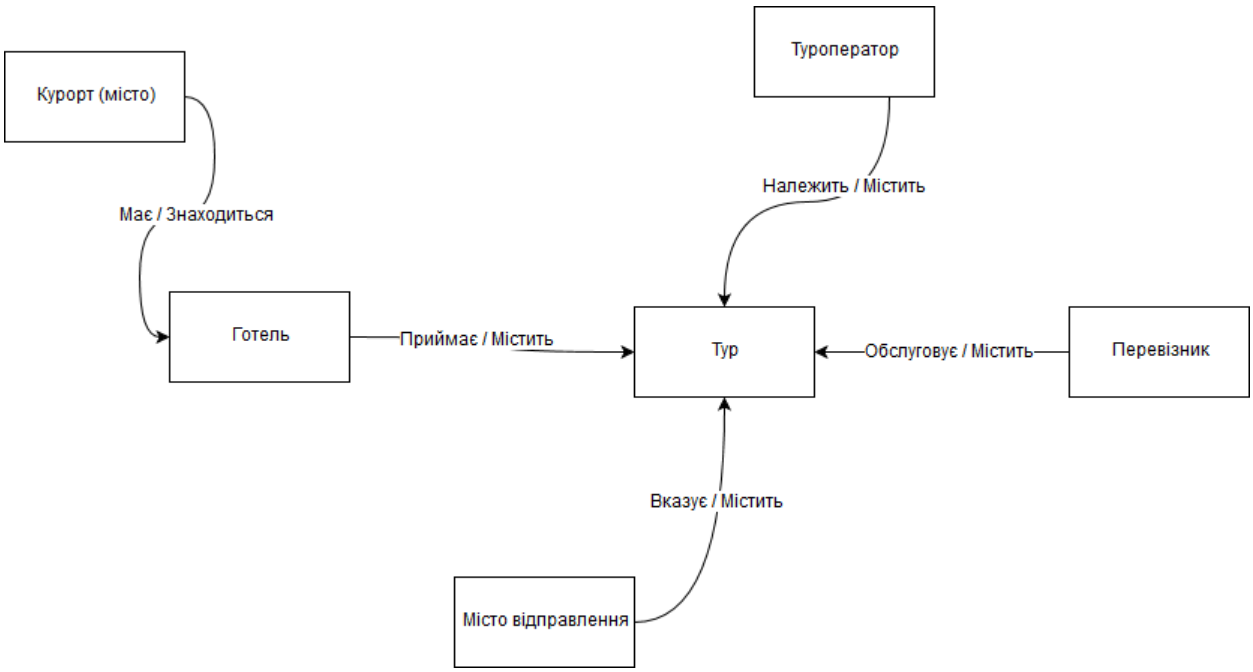


Рисунок 3.2 Інфологічна модель предметної області

Сходячи з вимог, пов'язаних з продуктивністю, база даних має копію таблиці турів під назвою – «тури недоступні». Єдина відмінність – це те, що таблиця містить дату додавання запису. Дана сутність описує тури, які не є актуальними за датою або були додані адміністратором вручну. Розуміється, можна і потрібно було б зберігати все в одній таблиці. Виходячи з проведених тестів, можна зробити умовивід, що MySQL (движок

InnoDB) сильно втрачає в продуктивності при записі нових кортежів, якщо обсяг БД більше 1 Gb.

На етапі даталогічного проектування необхідно установити відповідність між сутностями і характеристиками предметної області, взаємовідносинами і атрибутами. Для цього потрібно кожної сутності і характеристикам поставити у відповідність набір відносин (таблиць) і їх атрибутів (полів).

Після опису сутностей і установки зв'язків отримаємо наступну даталогічну модель основної бізнес-логіки (рис. 3.3):

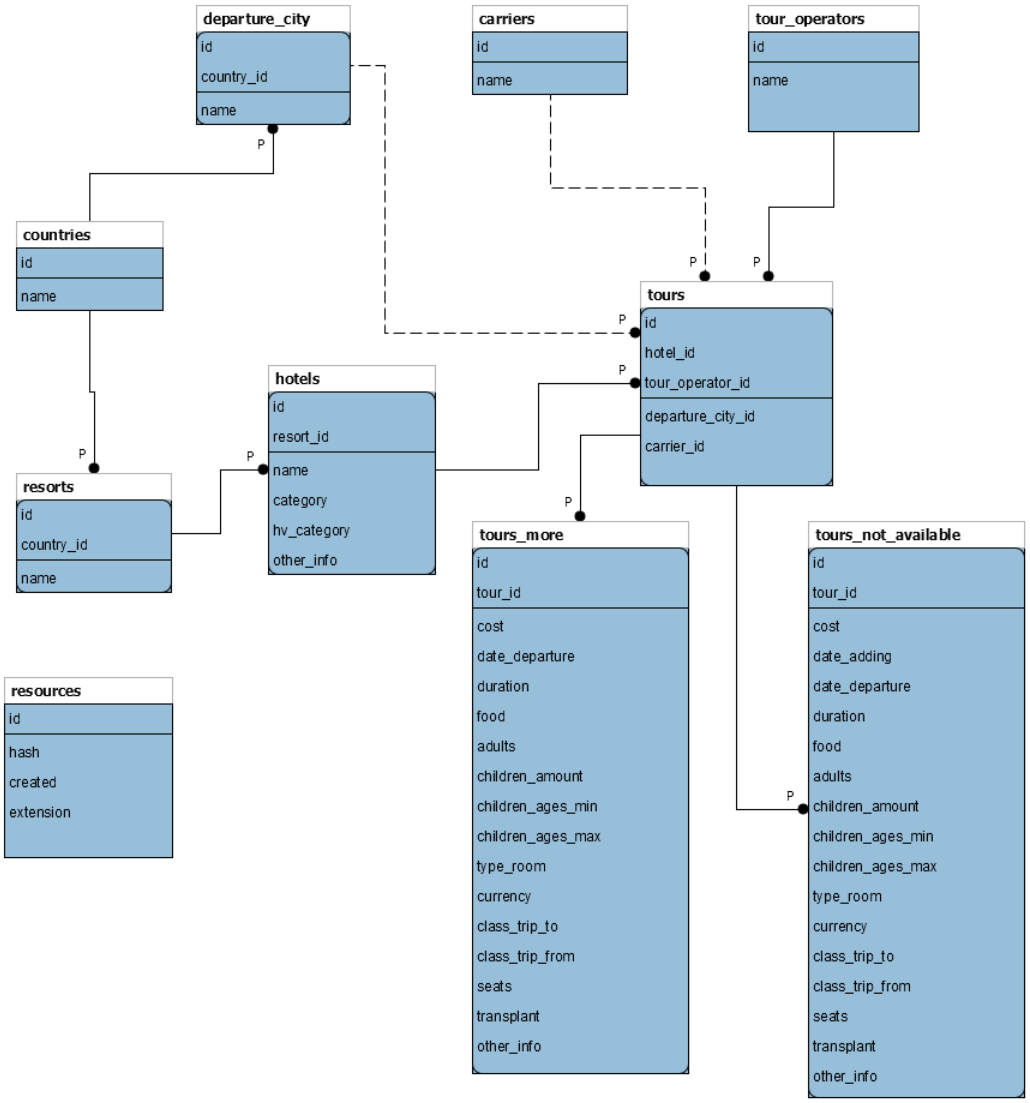


Рисунок 3.3 Даталогічна модель основної бази даних, представлена в IDEF1X

Як помітно з подання сутність «тури» розділена на додаткову таблицю «tours_more» і «tours_not_available» для недоступних турів. Це пов'язано з тим, що багато

даних дублюються в багатьох кортежі. Таким чином, вдалося заощадити займане дисковий простір, а також розмір самих таблиць.

На рис. 3.4 приведена даталогічна модель допоміжної бази даних для успішної роботи з користувачами і проведення міграцій.

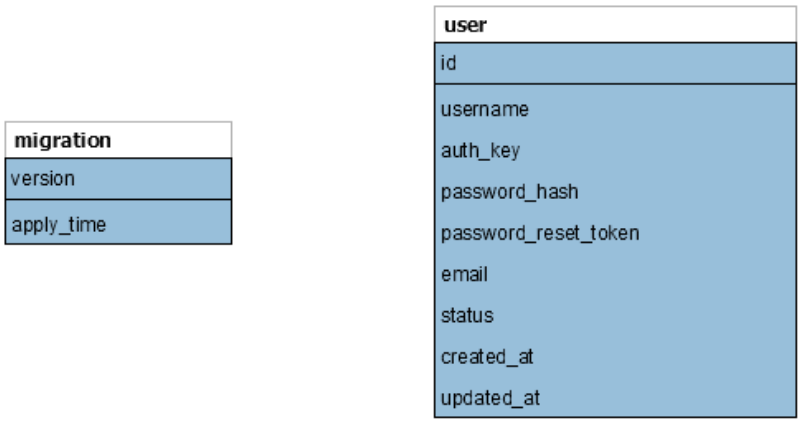


Рисунок 3.4 Даталогічна модель допоміжної бази даних, представлена у формі IDEF1X

Фізична модель даних окреслює дані за допомогою реальної СУБД [4]. Будемо вважати, що фізична модель даних реалізується за допомогою реляційної СУБД, хоча, як вже пригадувалося вище, це не є необхідним. Зв'язки, розроблені на етапі формування логічної моделі даних, перетворюються в таблиці, атрибути стають стовпцями таблиць, для ключових атрибутів створюються унікальні індекси, домени перетворюються в типи даних, прийняті в конкретній СУБД.

Обмеження, наявні в логічної моделі даних, реалізуються різними засобами СУБД, наприклад, за сприянням індексів, декларативних обмежень цілісності, тригерів, збережених процедур. При цьому знову-таки рішення, прийняті на рівні логічного моделювання визначають деякі межі, в межах яких можна розвивати фізичну модель даних. Наприклад, стосунки, що містяться в логічної моделі даних, повинні бути перетворені в таблиці, але для кожної таблиці можна додатково оголосити різні індекси, які збільшують бистроту доступу до даних. Тут багато що залежить від конкретної СУБД.

І, нарешті, як наслідок попередніх етапів з'являється власне сама база даних. База даних реалізована у реальній програмно-апаратної основі, і вибір цієї основи дозволяє істотно підняти швидкість роботи з базою даних.

Наприклад, можна вибирати різні типи комп'ютерів, змінювати кількість процесорів, обсяг оперативної пам'яті, дискові підсистеми і т.п. Дуже важлива також конфігурація СУБД в обраній апаратно-програмній платформі.

Таким чином, очевидно, що рішення, прийняті на кожному етапі моделювання і розробки бази даних, будуть впливати на подальші етапи. Тому особливу роль відіграє прийняття точного рішення на ранніх стадіях моделювання.

Приступимо до процесу фізичного проектування. Зазначимо атрибути, визначимо типи даних і розмірність атрибутів. Варто відзначити, що при виборі типів даних і способу їх розміщення було приділено багато часу аналізу продуктивності при заповненні БД модулем збору даних. Результат виконаної роботи представлений в додатку А.

Після того як були створені поля таблиці, застосовані до них властивості і заповнені атрибути, складається схема даних.

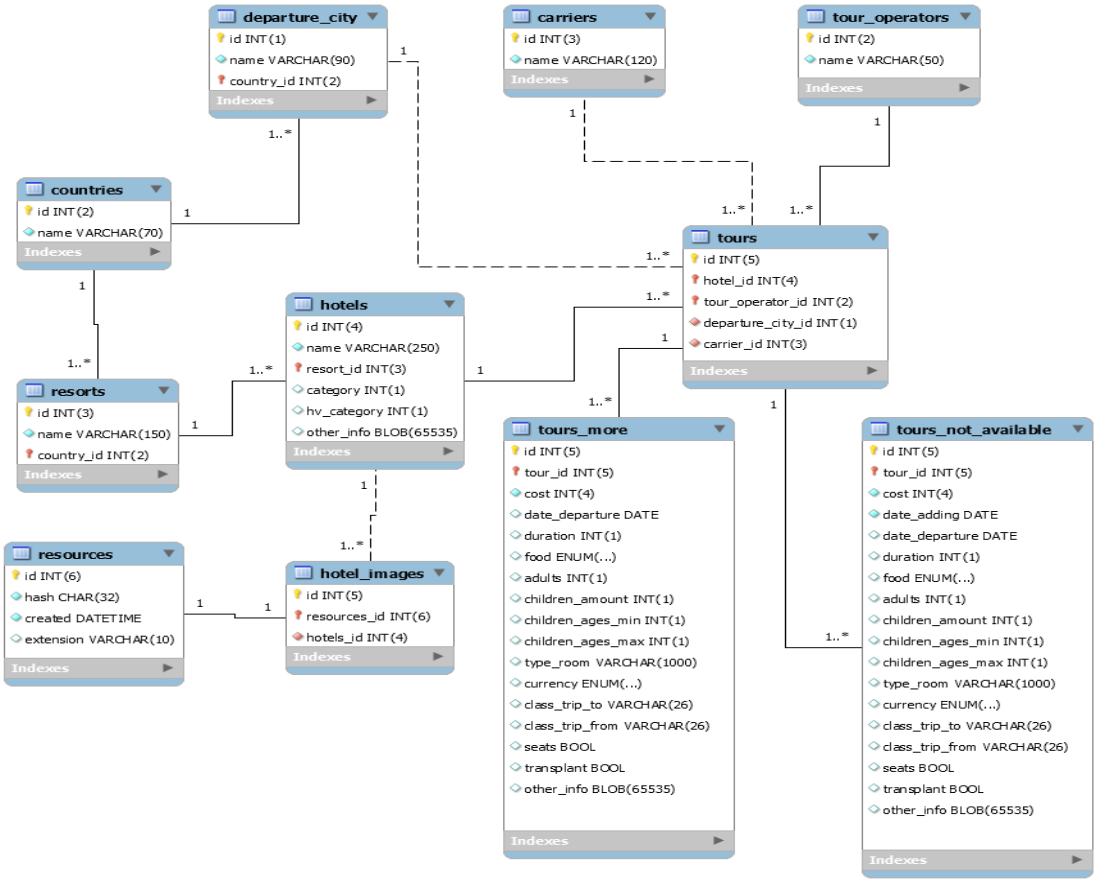


Рисунок 3.5 Зв'язки між таблицями в базі даних і типи атрибутів

Як видно на рис. 3.5, з'явилося відношення «багато до багатьох» між ресурсами і готелями, яке здійснюється за допомогою проміжної таблиці. Також були додані деякі атрибути виходячи з вимог, які були описані раніше і можливостей які надаються даної СУБД.

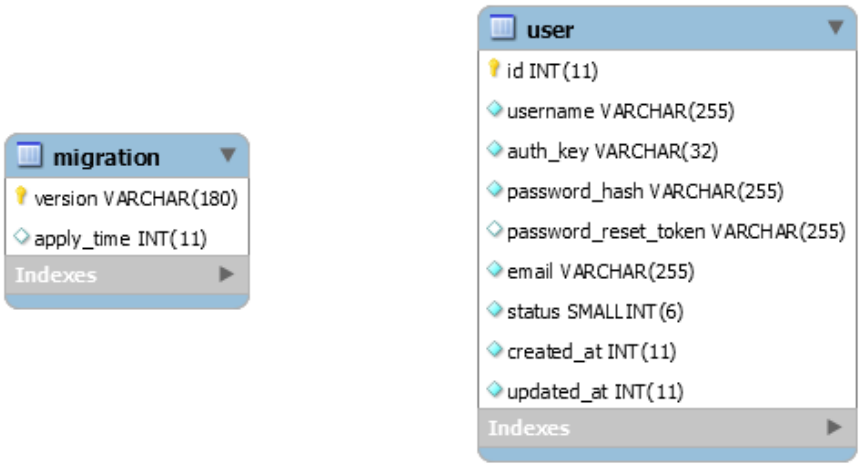


Рисунок 3.6 Атрибути для таблиць «користувач» і «міграція»

На рис. 3.6 можна побачити схему помічної бази даних. Нагадаю, що дані таблиці організовують доступ адміністратора до системи, а також можливість версіонування і модифікації БД в майбутньому.

3.1.4 Клієнтська частина системи збору та обробки даних

Дана система оперує даними одержуваними від програмних інтерфейсів які надаються туроператорами. Під програмним інтерфейсом (далі API, від. англ. application programming interface) будемо розуміти веб-служби, що приділяють набір способів отримання необхідних даних. Такі інтерфейси використовується в веб-розробці, як правило, певний набір HTTP-запитів, а також визначення структури HTTP-відповідей, для вираження яких використовують XML або JSON формати.

Існує безліч протоколів доступу, які використовується при розробці API: RPC, SOAP, REST. На жаль, у кожного туроператора свої способи взаємодії і найчастіше вони сильно різняться між собою.

Звичайно, існують спроби випустити єдиний стандарт для даної галузі під назвою - TourML. Цей формат - спільна розробка «Мегатек» і «Само-Софт». Дозволяє передавати прайс-листи, інформацію про присутність місць, зупинках продажів, а також здійснювати бронювання. Формат документований, для нього розроблена xsd-схема для перевірки валідності xml-документів.

На жаль, даний стандарт не набув розповсюдження і далеко не всі туроператори користуються цим підходом. На сьогоднішній день кожен туроператор сам вирішує, яким

чином реалізовувати свій інтерфейс. Найчастіше це SOAP з описом у вигляді WSDL або REST з повною документацією.

Для вирішення завдання уніфікації було прийнято створювати класи-обв'язки інкапсулюючі взаємодію всередині себе. Свого роду це реалізація інтерфейсу з певними методами і атрибутами.

На рис. 3.7 нижче можна розглянути процес протікання даних. Він складається з наступних елементів:

– клас-обв'язка як говорилося раніше, реалізує стандартизовану форму представлення даних туроператора;
– форматування - набір функцій робить приведення отриманих даних в задану форму;
– обробка даних - основоположна частина системи, яка включає в себе безліч дій, приміром додаток даними з інших джерел;
– внутрішній кеш використовується для мінімізації запитів до БД.

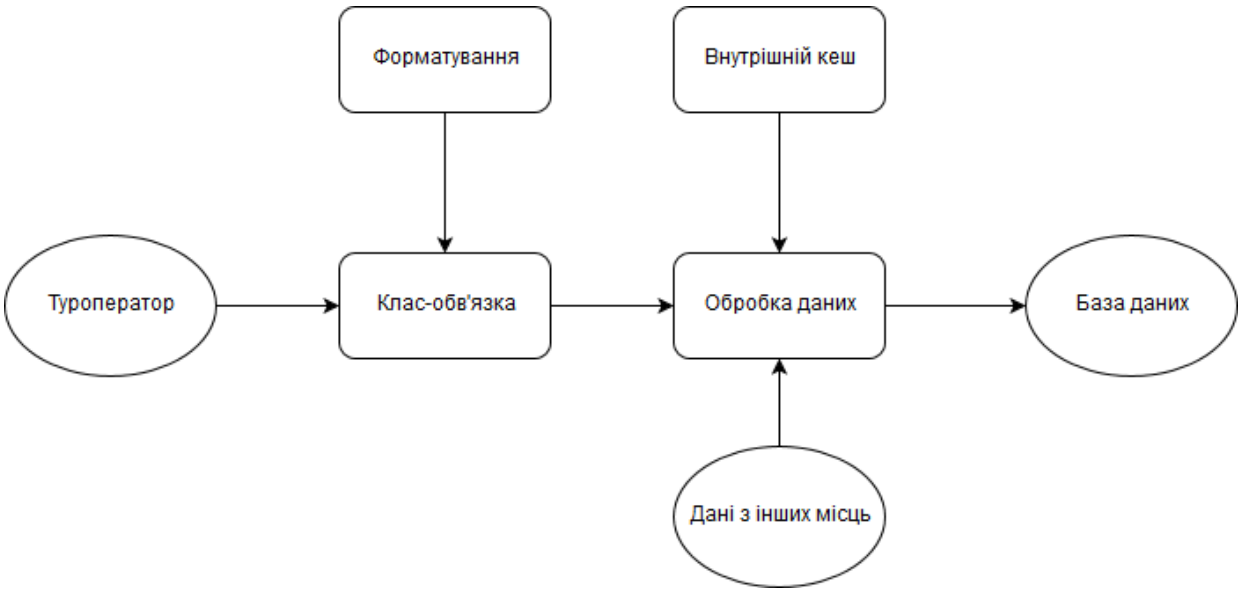


Рисунок 3.7 Рух даних від туроператора до БД

Під час розробки з'явилася необхідність в пошуку та виборі найбільш результативних алгоритмів і способів обробки даних. Для вирішення цієї проблеми був обраний емпіричний спосіб оцінки. Був спроектований абстрактний клас, при спадкуванні від якого можна було з легкістю продавати тестування різних алгоритмів або способів їх виконання.

Для тестування досить успадковуватися від даного класу і перевизначити метод «go», потім викликати метод «start» і отримати результат за допомогою «results».

Опціонально можна змінити поведінку, при тестуванні перевизначивши інші методи і використовуючи зазначені атрибути на UML діаграмі (див. рис. 3.8).

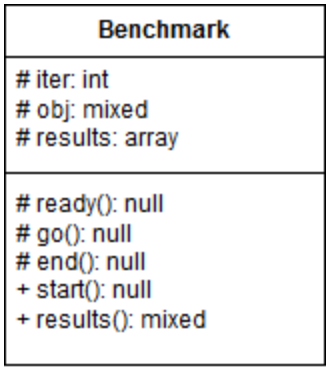


Рисунок 3.8 UML діаграма для базового класу тестування продуктивності

За допомогою тестування продуктивності було вибрано безліч нестандартних рішень. Так, наприклад, існує клас для будови запитів на додавання даних. Його завдання полягає в накопиченні виразів до певної величини і подальшому побудова рядка для запиту. Причина такого підходу полягає в часі, яке потрібно додатком для обробки кожного INSERT-запиту. Виконується перевірка з'єднання, відправка повідомлення в БД, блокування таблиці, повернення управління або помилки. Все це можна накопичувати і опрацьовувати пакетами (розмір пакета не повинен перевищувати обмеження БД).

Основним вузьким місцем в продуктивності є робота з БД. Як зазначалося раніше, для мінімізації кількості запитів була реалізована можливість кешування даних.

Деякі туроператори не уявляють можливості отримання зображень готелів без водяних знаків. На що вони рекомендують зменшити зображення при необхідності. Одну з таких завдань розв'язує частина системи під назвою «форматування». Він стає на бік, з якої повинно зробити урізання і розмір в пікселях.

Одним з цікавих рішень є спосіб обробки даних туроператорів, які вимагають більш точні параметри при виборі даних. Виникає необхідність перебору всіх можливих варіантів туру. Тут прийшлося вдатися до комбінаторики, а так само до алгоритму обходу дерева можливих варіантів. Так, наприклад два параметра, з яких перший має 3 можливих значення, а другий – тільки 2, отримаємо 6 можливих значень. Висловивши це у вигляді розміщень з повтореннями, одержуємо формулу:

$$A_1 \,{}^{1}_{n_1} \times A_2 \,{}^{1}_{n_2} \times A_3 \,{}^{1}_{n_3} \cdots A_m \,{}^{1}_{n_m},$$ (3.1)

Розписавши формулу розміщення і спростивши за правилом множення, отримуємо наступне:

$$n_1 \times n_2 \times n_3 \cdots n_m, \tag{3.2}$$

де n – кількість варіантів;

m – кількість параметрів.

Збудуємо дерево варіантів для прикладу, описаного вище.

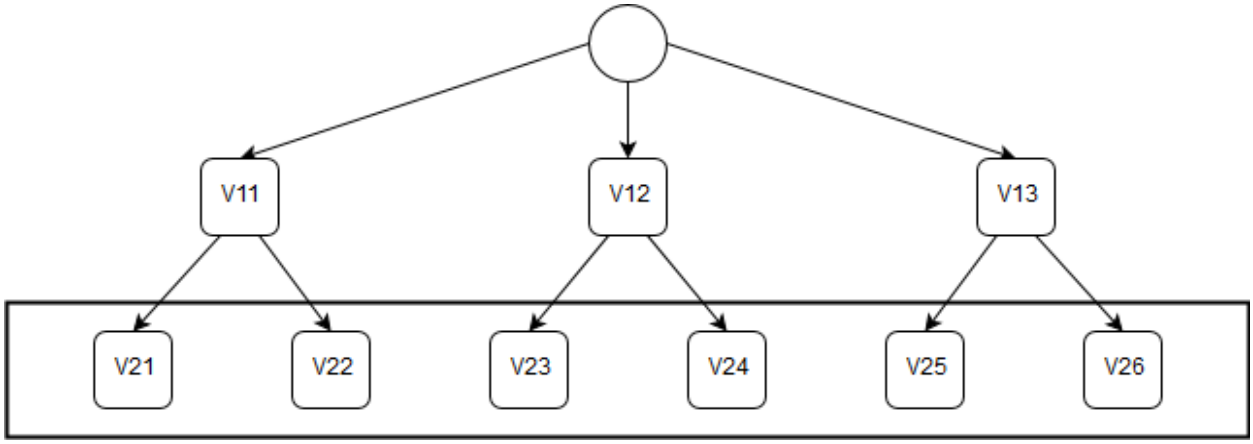


Рисунок 3.9 Дерево варіантів для двох параметрів

Виходячи з описаного дерева на рис. 3.9 видно, що кількість варіантів (листя дерева) буде стрімко зростати при збільшенні як самого кількість параметрів (рівнів дерева), так і кількості їх варіантів (ширина рівня).

Одним із способів є реалізація вкладених циклів кількістю рівним глибині дерева. Однак даний підхід не досить гнучкий, з огляду на, що кількість параметрів може змінюватися. Інший спосіб – рекурсивний обхід дерева позбавляє від попередньої проблеми, але обмежує через можливість переповнення стека.

Вивчивши можливості концепції паттерна генераторів, було реалізовано гнучке і, в той же час, нічим не обмежене рішення: використовувати рекурсивні генератори, де кожному рівню дерева відповідає свій унікальний генератор даних. Це дозволяє репродукувати практично необмежені комбінації параметрів, що було і необхідно.

Іншим важливим завданням є організація завантаження ресурсів і їх збереження для подальшого доступу. Так в процесі розробки було прийнято рішення використовувати асинхронне завантаження, так звані ліниві потоки для збереження даних, а також розподіл файлів на підкаталоги із записом інформації в відповідну таблицю.

3.2 Результат розробки програмного засобу

Інтерфейс представлений у вигляді веб-інтерфейсу. Його реалізація була виконана на класичному стеку технологій: HTML, CSS (з використанням препроцесора Less), JS і т.д.

Розділ роботи з записами на увазі роботу з турами (рис. 3.10). Він розділений на кілька підрозділів: імпорт - імпортування даних з файлу; експорт - отримання даних з БД; інші два розділи призначені для перегляду активних і архівних турів.

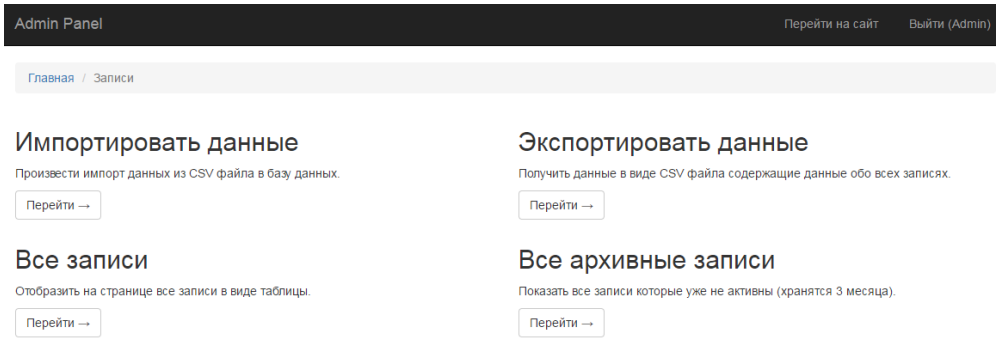


Рисунок 3.10 Підрозділ роботи з записами

Сторінка перегляду списку турів набирається з фільтра для створення вибірок та таблиці результатів (рис. 3.11). Для архівних турів сторінка аналогічна.

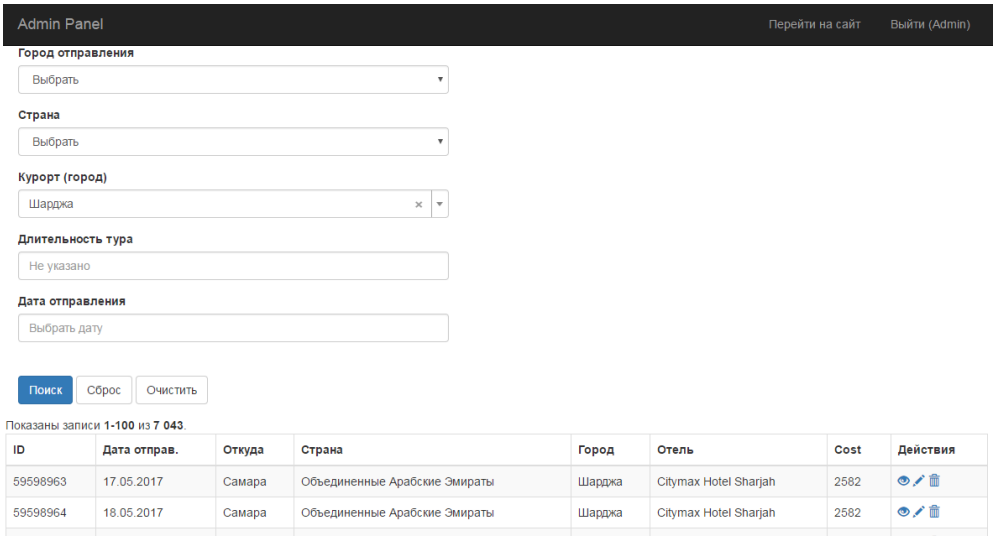


Рисунок 3.11 Сторінка перегляду списку турів

У таблиці доступні три дії над туром: перегляд туру на окремій сторінці, його редагування і видалення в архівний тур.

Ознайомившись з основними сторінками панелі адміністрування, перейдемо до тієї частини системи, за підтримкою якої взаємодіє клієнт тур агентства (див. рис. 3.12).

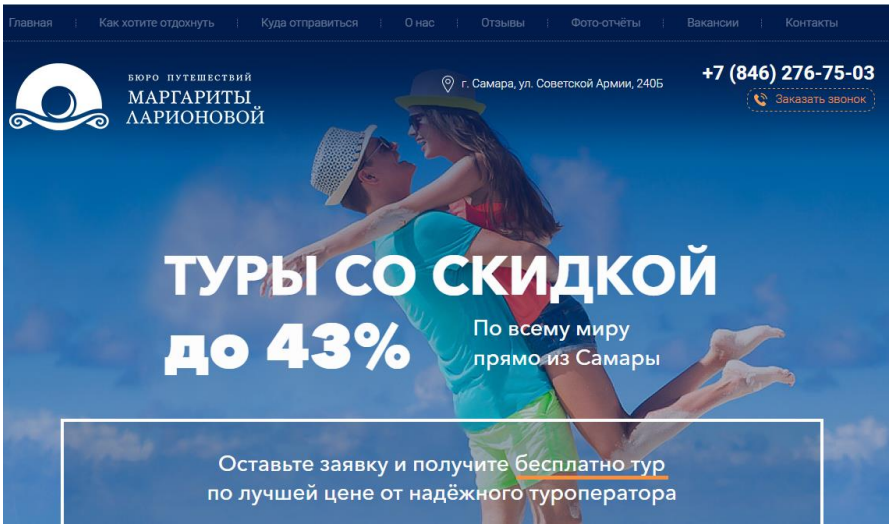


Рисунок 3.12 Головна сторінка клієнтської частини

Пошук і фільтрація турів здійснюється за підтримкою відповідного розділу (рис. 3.13). При виборі більшість параметрів проходить зміна всіх інших виходячи з уже встановлених значень. Так само фільтр володіє розширеним пошуком з великою кількістю параметрів.

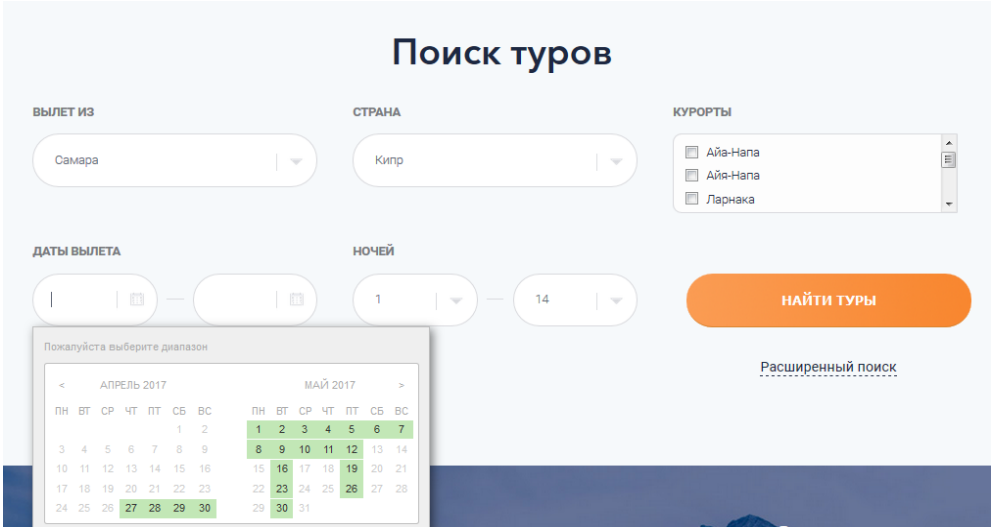


Рисунок 3.13 Фільтр для пошуку турів

Результат роботи фільтра відображається нижче самого фільтра без переходу на іншу сторінку (див. рис. 3.14).

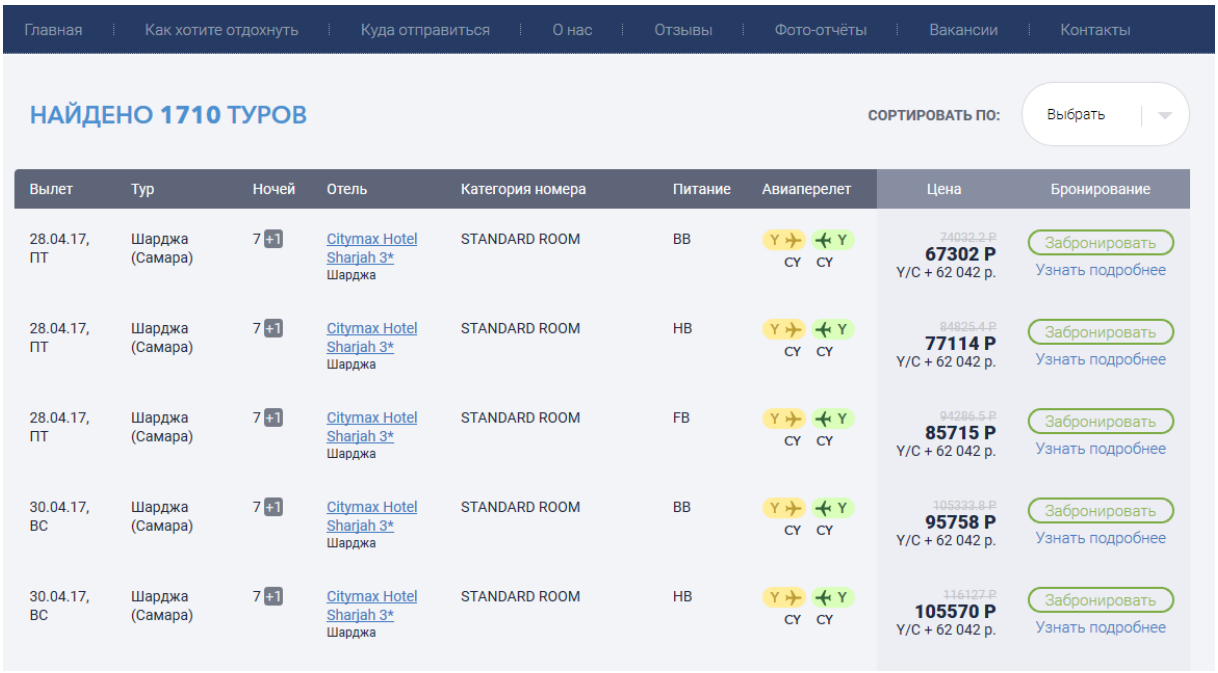


Рисунок 3.14 Результат роботи фільтра

Сторінка туру пропонує більш докладний опис з фотографіями готелю і іншу інформацію (рис. 3.15).

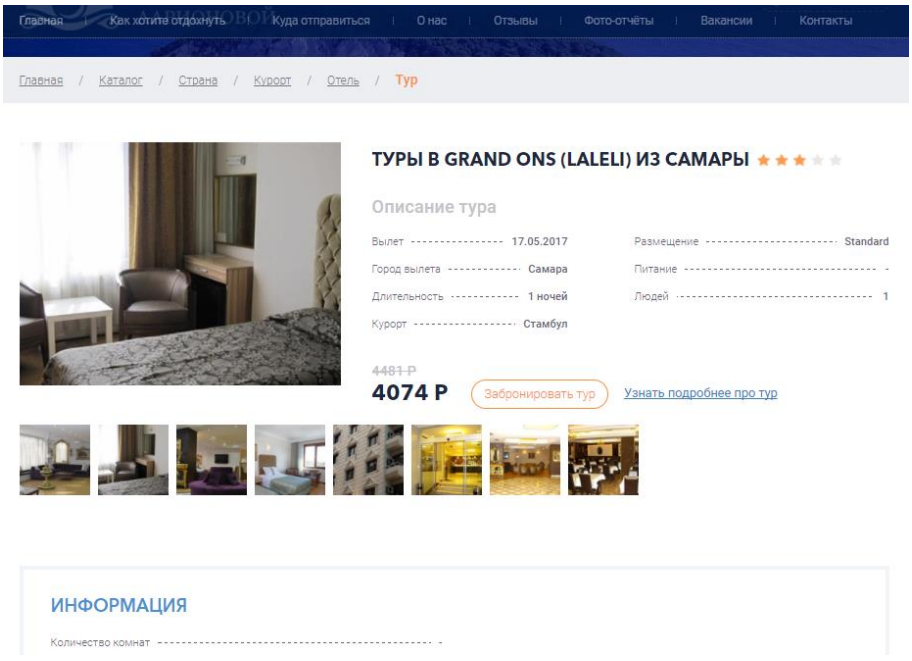


Рисунок 3.15 Частина сторінки туру

Так само можна перейти на сторінку готелю для отримання більш тотальної інформації (рис. 3.16).

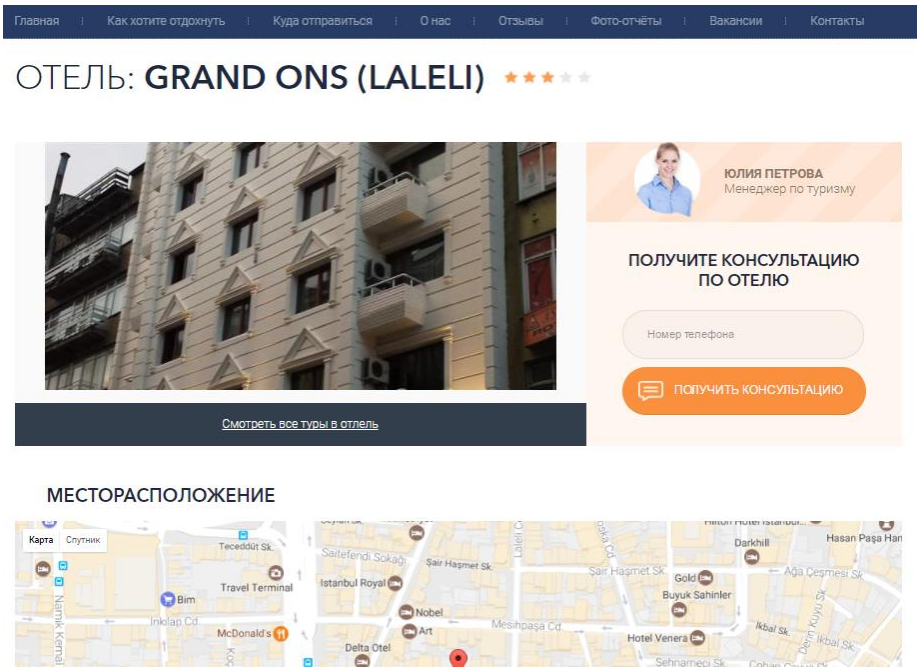


Рисунок 3.16 Сторінка готелю

На сторінці готелю можна розшукати опис готелю, фотографії, розташування на карті, схожі готелі (поруч розташовані), інші тури в цей готель і т.д.

3.3 Висновки до третього розділу

В даному розділі проведена реалізація програмного засобу, а саме:

розглянуті і вибрані необхідні інструментальні способи, а так же технології потрібні для розробки даної системи;

описаний процес реалізації клієнтської частини з урахуванням всіх аспектів проектування згаданих у другому розділі даної роботи;

досліджено процес публікації кодової бази на сервер, запуску системи, організація роботи і т.д .;

розвинуто докладний опис користувальницького інтерфейсу системи (панель адміністрування і веб-інтерфейс для клієнтів агенції), а також вказані фотознімки екрану для більшої наочності;

розглянуті засоби тестування реалізованого товару, а так само згадано про різні методи тестування (розробка через тестування, розробка через поводження).

3.4 Література до третього розділу

1. А.М. Вендров «Современные методы и средства проектирования информационных систем» – М.: Финансы и статистика, 1998. – 176 с.
2. Г. Калянов «Номенклатура CASE-средств и виды проектной деятельности» – М.: ЛОРИ, 1996, – 242с.
3. Орлов С. А., Программная инженерия. Учебник для вузов. 5-е издание обновленное и дополненное. Стандарт третьего поколения. — СПб.: Питер, 2016. — 640 с.: ил, — (Серия «Учебник для вузов»).
4. Модели жизненного цикла разработки ИС – belani.narod.ru [Електронний ресурс] – Режим доступу: http://belani.narod.ru/1/S1.htm

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ. ЕКОЛОГІЯ

В даному розділі проведено аналіз потенційних небезпечних та шкідливих виробничих факторів, причин пожеж. Розглянуті заходи, які дозволяють забезпечити гігієну праці і виробничу санітарію. На підставі аналізу розроблені заходи з техніки безпеки та рекомендації з пожежної профілактики.

Завданням даної магістерської роботи було вивчення методів оптимізації БД та впровадження їх в існуючу програмну систему, яка призначена для організації інформаційного супроводу туристичного агентства, і як результат було розроблено перелік необхідних засобів оптимізації. Так як в процесі проектування використовувалося ПК, то аналіз потенційно небезпечних і шкідливих виробничих чинників виконується для персонального комп'ютера на якому буде використовуватися розроблена об'єктна модель.

4.1 Загальні питання з охорони праці

В законі України «Про охорону праці» визначається, що охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

При роботі з обчислювальною технікою змінюються фізичні і хімічні фактори навколишнього середовища: виникає статична електрика, електромагнітне випромінювання, змінюється температура і вологість, рівень вміст кисню і озону в повітрі. Повітря забруднюється шкідливими хімічними речовинами антропогенного походження за рахунок деструкції полімерних матеріалів, які використовуються для обробки приміщень та обладнання. Неправильна організація робочого місця сприяє загальному і локальній напрузі м'язів шиї, тулуба, верхніх кінцівок, викривлення хребта і розвитку остеохондрозу. На всіх підприємствах, в установах, організаціях повинні створюватися безпечні і нешкідливі умови праці.

4.1.1 Правові та організаційні основи охорони праці

Основним організаційним напрямом у здійсненні управління в сфері охорони праці є усвідомлення пріоритету безпеки праці і підвищення соціальної відповідальності держави, і особистої відповідальності працівників.

Користувачі персональних комп'ютерів, для яких ця робота є головною, підлягають медичним оглядам: попереднім — під час влаштування на роботу і періодичним — протягом професійної діяльності раз на два роки. Жінок з часу встановлення вагітності та в період годування дитини грудьми до роботи з ПК не допускають.

Обов'язки працівників щодо додержання вимог нормативно-правових актів з охорони праці (ст. 14), відповідальність робітників всіх категорій за порушення вимог щодо охорони праці (ст. 44) та структура організації/виробництв системи управління охорони праці визначені у [1].

4.1.2 Організаційно-технічні заходи з безпеки праці

В організації/підприємстві проводиться навчання і перевірка знань з питань охорони праці відповідно до вимог Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці, затвердженого наказом Держнаглядохоронпраці України від 26.01.2005 N 15, зареєстрованого в Міністерстві юстиції України 15.02.2005 за N 231/10511 [2].

Також впроваджені організаційні заходи з пожежної безпеки - навчання і перевірку знань відповідно до вимог Типового положення про інструктажі, спеціальне навчання та перевірку знань з питань пожежної безпеки на підприємствах, в установах та організаціях України, затвердженого наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 29.09.2003 N 368, зареєстрованого в Міністерстві юстиції України 11.12.2003 за N 1148/8469 [3].

4.2 Аналіз стану умов праці

Робота над створенням об’єктної моделі забезпечення оцінки кібербезпеки, розрахунок уразливості системи і визначення зв'язків атак і захистів проходитиме в

побутовому приміщенні. Для даної роботи достатньо однієї людини, для якої надано робоче місце зі стаціонарним комп'ютером.

4.2.1 Вимоги до приміщень

Геометричні розміри приміщення зазначені в табл. 4.1.

Таблиця 4.1 - Розміри приміщення

Найменування	**Значення**
Довжина, м	3
Ширина, м	3
Висота, м	2,5
Площа, $м^2$	9
Об'єм, $м^3$	22,5

Згідно з [4] розмір площі для одного робочого місця оператора персонального комп'ютера має бути не менше 6 кв. м, а об'єм — не менше 20 куб. м. Отже, дане приміщення цілком відповідає зазначеним нормам.

Для зручності спільної роботи з іншими працівниками (обговорення ідей, з'ясування проблем і т.д.) в кімнаті є дивани і журнальний стіл, обставлені живими квітами. Також робочий процес пов'язаний з багатьма документами, теками, журналами для чого приміщення облаштоване принтером і шафою для зручності. Задля дотримання визначеного рівня мікроклімату в будівлі встановлено систему опалення та кондиціонування.

Для забезпечення потрібного рівного освітленості кімната має вікно та систему загального рівномірного освітлення, що встановлена на стелі. Для дотримання вимог пожежної безпеки встановлено порошковий вогнегасник та систему автоматичної пожежної сигналізації.

4.2.2 Вимоги до організації місця праці

При порівнянні відповідності характеристик робочого місця нормативним основні вимоги до організації робочого місця за [5] (табл. 4.2) і відповідними фактичними значеннями для робочого місця, констатуємо повну відповідність.

Таблиця 4.2 - Характеристики робочого місця

Найменування параметра	Фактичне значення	Нормативне значення
Висота робочої поверхні, мм	750	680 ÷800
Висота простору для ніг, мм	730	не менше 600
Ширина простору для ніг, мм	660	не менше 500
Глибина простору для ніг, мм	700	не менше 650
Висота поверхні сидіння, мм	470	400 ÷500
Ширина сидіння, мм	400	не менше 400
Глибина сидіння, мм	400	не менше 400
Висота поверхні спинки, мм	600	не менше 300
Ширина опорної поверхні спинки, мм	500	не менше 380
Радіус кривини спинки в горизонтальній площині, мм	400	400
Відстань від очей до екрану дисплея, мм	800	700 ÷800

4.2.3 Навантаження та напруженість процесу праці

Під час виконання робіт використовують ПК та периферійні пристрої (лазерні та струменеві), що призводить до навантаження на окремі системи організму. Такі перекоси у напруженні різних систем організму, що трапляються під час роботи з ПК, зокрема, значна напруженість зорового аналізатора і довготривале малорухоме положення перед екраном, не тільки не зменшують загального напруження, а навпаки, призводять до його посилення і появи стресових реакцій.

Найбільшому ризику виникнення різноманітних порушень піддаються: органи зору, м’язово скелетна система, нервово-психічна діяльність, репродуктивна функція у жінок.

Рекомендовано застосування екранних фільтрів, локальних світлофільтрів (засобів індивідуального захисту очей) та інших засобів захисту, а також інші профілактичні заходи на ведені в [5].

Роботу за дипломним проектом визнано, таку, що займає 50% часу робочого дня та за восьмигодинної робочої зміни рекомендовано встановити додаткові регламентовані перерви:

– для операторів персональних комп’ютерів тривалістю 15 хв через дві години роботи.

4.3 Виробнича санітарія

На підставі аналізу небезпечних та шкідливих факторів при виробництві (експлуатації), пожежної безпеки можуть бути надалі вирішені питання необхідності забезпечення працюючих достатньою кількістю освітлення, вентиляції повітря, організації заземлення, тощо.

4.3.1 Аналіз небезпечних та шкідливих факторів при виробництві (експлуатації) виробу

Роботу, пов'язану з ЕОМ з ВДТ, у тому числі на тих, які мають робочі місця, обладнані ЕОМ з ВДТ і ПП, виконують із забезпеченням виконання [6], які встановлюють вимоги безпеки до обладнання робочих місць, до роботи із застосуванням ЕОМ з ВДТ і ПП. Переважно роботи за проектами виконують у кабінетах чи інших приміщеннях, де використовують різноманітне електрообладнання, зокрема персональні комп'ютери (ПК) та периферійні пристрої.

Основними робочими характеристиками персонального комп'ютера є:

- робоча напруга U=+220В +-5%;
- робочий струм I=2А;
- споживана потужність P=350 Вт.

Робочі місця мають відповідати вимогам Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затверджених постановою Головного державного санітарного лікаря України від 10.12.98 N 7 [5].

За умов роботи з ПК виникають наступні небезпечні та шкідливі чинники: несприятливі мікрокліматичні умови, освітлення, електромагнітні випромінювання, забруднення повітря шкідливими речовинами, шум, вібрація, електричний струм, електростатичне поле, напруженість трудового процесу та інше.

Аналіз небезпечних та шкідливих виробничих факторів виконується у табличній формі (табл. 4.3).

Таблиця 4.3 - Аналіз небезпечних і шкідливих виробничих факторів

Небезпечні і шкідливі виробничі фактори	Джерела факторів (види робіт)	Кількісна оцінка	Нормативні документи
1	2	3	4
фізичні			
- підвищена температура поверхонь обладнання	експлуатація ЕОМ, принтерів, сканерів чи/або серверного обладнання для роботи	2	ДСН 3.3.6.042-99 [4]
- підвищений рівень шуму на робочому місці	-//-	2	ДСН 3.3.6.042-99 [4]
- підвищена або знижена вологість повітря	-//-	2	ДСН 3.3.6.042-99 [4]
- підвищена або знижена рухливість повітря	-//-	1	ДСН 3.3.6.042-99 [4]
- підвищений рівень напруги електричної мережі, замикання якої може відбутися через тіло людини	-//-	4	ГОСТ 12.1.030-81 [7] ГОСТ 13109-97 [8]
- недостатність природного світла	порушення умов праці (вимог до приміщень)	2	ДБН В.2.5-28:2015 [9]
- недостатнє освітлення робочої зони	порушення гігієнічних параметрів виробничого середовища	3	ДБН В.2.5-28:2015 [9]
- підвищена яскравість світла	порушення умов праці (організації місця праці - налагодження моніторів)	1	ДСанПіН 3.3.2.007-98[5]
- понижена контрастність	-//-	1	ДСанПіН 3.3.2.007-98[5]

Продовження таблиці 4.3

психофізіологічні:			
- нервово-психічна перевантаження (розумове, перенапруження аналізаторів-зорових)	- пошук інформації для постановки теми; - пошук та аналіз аналогів і літератури; - пошук наявних технологій, моделювання та аналіз алгоритмів; - виконання роботи за темою диплома, тестування; - оформлення роботи	4	НПАОП 0.00-1.28-10[6] ДСанПіН 3.3.2.007-98[5]
- фізичні (статичне – сидіння)	порушення умов праці (організації місця праці - сидіння користувача,) та організації робочого часу - безперервна робота)	2	НПАОП 0.00-1.28-10[6] ДСанПіН 3.3.2.007-98[5]

4.3.2 Пожежна безпека

Для гасіння пожеж в офісному приміщенні пропонується використовувати порошкові або вуглекислотні вогнегасники, так як вони є універсальними.

Заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном), надійно захищені діелектричними щитками та/або сітками з метою недопущення потрапляння працівника під напругу.

В приміщенні наявна затверджена «План-схема евакуації з кабінету (приміщення)». Пожежна безпека при застосуванні ЕОМ забезпечується:

- системою запобігання пожежі,
- системою протипожежного захисту,
- організаційно-технічними заходами.

Згідно [10] таке приміщення, площею 9 м^2, відноситься до категорії "В" (пожежонебезпечної) та для протипожежного захисту в ньому проектом передбачено

устаткування автоматичною пожежною сигналізацією із застосуванням датчиків-сповіщувачів РІД-1 (сповіщувач димовий ізоляційний) в кількості 1 шт., і застосуванням первинних засобів пожежогасіння. Відповідно до норм первинних засобів пожежогасінні пропонується використовувати:

– ручний вуглекислий вогнегасник ОУ-5 в кількості 1 шт.;
– повсть 1 1 м^2, кошму 2×1,5 м^2 або азбестове полотно 2×2 м^2 в кількості 1 шт.

Виникнення пожежі можливе, якщо на об'єкті є горючі речовини, окислювач і джерела запалювання. Вірогідність пожежної небезпеки приймається значною, якщо ймовірна взаємодія цих трьох чинників. Горючими компонентами є: будівельні матеріали для акустичної і естетичної обробки приміщень, перегородки, підлоги, двері, ізоляція силових, сигнальних кабелів і т.д.

4.3.3 Електробезпека

На робочому місці виконуються наступні вимоги електробезпеки: ПК, периферійні пристрої та устаткування для обслуговування, електропроводи і кабелі за виконанням та ступенем захисту відповідають класу зони за ПУЕ (правила улаштування електроустановок), мають апаратуру захисту від струму короткого замикання та інших аварійних режимів. Лінія електромережі для живлення ПК, периферійних пристроїв і устаткування для обслуговування, виконана як окрема групова три- провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Штепсельні з'єднання та електророзетки крім контактів фазового та нульового робочого провідників мають спеціальні контакти для підключення нульового захисного провідника. Електромережа штепсельних розеток для живлення персональних ПК, укладено по підлозі поруч зі стінами відповідно до затвердженого плану розміщення обладнання та технічних характеристик обладнання. Металеві труби та гнучкі металеві рукави заземлені. Захисне заземлення включає в себе заземлюючих пристроїв і провідник, який з'єднує заземлюючий пристрій з обладнанням, яке заземлюється - заземлюючий провідник.

4.4 Гігієнічні вимоги до параметрів виробничого середовища

4.4.1 Мікроклімат

Мікроклімат робочих приміщень – це клімат внутрішнього середовища цих приміщень, що визначається діючої на організм людини з'єднанням температури, вологості, швидкості переміщення повітря. Оптимальні значення для температури, відносної вологості й рухливості повітря для зазначеного робочого місця відповідають [11] і наведені в табл. 4.4:

Таблиця 4.4 - Норми мікроклімату робочої зони об'єкту

Період року	Категорія робіт	Температура C^0	Відносна вологість %	Швидкість руху повітря, м/с
Холодна	легка-1 а	22 - 24	40 – 60	0,1
Тепла	легка-1 а	23 - 25	40 – 60	0,1

4.4.2 Освітлення

Збільшення освітленості сприяє поліпшенню працездатності навіть в тих випадках, коли процес праці практично не залежить від зорового сприйняття. При поганому освітленні людина швидко втомлюється, працює менш продуктивно, виникає потенційна небезпека помилкових дій і нещасних випадків.

У проекті, що розробляється, передбачається використовувати суміщене освітлення. У світлий час доби використовуватиметься природне освітлення приміщення через віконні отвори, в решту часу використовуватиметься штучне освітлення. Штучне освітлення створюється газорозрядними лампами.

Розрахунок освітлення.

Для виробничих та адміністративних приміщень світловий коефіцієнт приймається не менше -1/8, в побутових – 1/10:

$$S_b = \left(\frac{1}{5} \div \frac{1}{10}\right) \cdot S_n, \qquad (4.1)$$

де S_b – площа віконних прорізів, м^2;

S_n – площа підлоги, м^2.

$S_n = a \cdot b = 3 \cdot 3 = 9$ м^2,

$S_{вік} = 1/10 \cdot 9 = 0{,}9$ м^2.

Приймаємо 2 вікна площею $S = 0,9$ м^2 кожне.

Світильники загального освітлення розташовуються над робочими поверхнями в рівномірно-прямокутному порядку. Для організації освітлення в темний час доби передбачається обладнати приміщення, довжина якого складає 5 м, ширина 5 м, світильниками ЛПО2П, оснащеними лампами типа ЛБ (дві по 80 Вт) з світловим потоком 5400 лм кожна.

Розрахунок штучного освітлення виробляється по коефіцієнтах використання світлового потоку, яким визначається потік, необхідний для створення заданої освітленості при загальному рівномірному освітленні. Розрахунок кількості світильників n виробляється по формулі (4.2):

$$n = \frac{E \cdot S \cdot Z \cdot K}{F \cdot U \cdot M}, \qquad (4.2)$$

де E – нормована освітленість робочої поверхні, визначається нормами – 300 лк;

S – освітлювана площа, м^2; S = 9 м^2;

Z – поправочний коефіцієнт світильника (Z = 1,15 для ламп розжарювання та ДРЛ;

Z = 1,1 для люмінесцентних ламп) приймаємо рівним 1,1;

K – коефіцієнт запасу, що враховує зниження освітленості в процесі експлуатації – 1,5;

U – коефіцієнт використання, залежний від типу світильника, показника індексу приміщення і т.п. – 0,575

M – число люмінесцентних ламп в світильнику – 2;

F – світловий потік лампи – 5400лм (для ЛБ-80).

Підставивши числові значення у формулу (4.2), отримуємо:

$$n = \frac{300 \times 9 \times 1.1 \times 1.5}{5400 \times 0.575 \times 2} = 0,7 \approx 1$$

Приймаємо освітлювальну установку, яка складається з 1 світильника, який складається з двох люмінесцентних ламп загальною потужністю 160 Вт, напругою – 220 В.

4.5 Вентилювання

У приміщенні, де знаходяться ЕОМ, повітрообмін реалізується за допомогою природної організованої вентиляції (вентиляційні шахти), тобто при V приміщення > 40 м^3 на одного працюючого допускається природна вентиляція. Цей метод забезпечує приток потрібної кількості свіжого повітря, що визначається в СНіП.

Також має здійснюватися провітрювання приміщення, в залежності від погодних умов, тривалість повинна бути не менше 10 хв. Найкращий обмін повітря здійснюється при наскрізному провітрюванні.

4.6 Заходи з організації виробничого середовища та попередження виникнення надзвичайних ситуацій

Розрахунок захисного заземлення (забезпечення електробезпеки будівлі).

Загальний опір захисного заземлення визначається за формулою:

$$R_{ççï} = \frac{R_ç \cdot R_n}{R_n \cdot n \cdot \eta_ç + R_ç \cdot \eta_n} \qquad (4.3)$$

де $R_з$ - опір заземлення, якими можуть бути труби, опори, кути і т.п., Ом;

$R_п$ - опір опори, яка з'єднує заземлювачі, Ом;

n - кількість заземлювачів;

$\eta_з$ - коефіцієнт екранування заземлювача; приймається в межах 0,2 ÷ 0,9; $\eta_з$ = 0,7

$\eta_п$ - коефіцієнт екранування сполучної стійки; приймається в межах 0,1 ÷ 0,7; $\eta_п$ = 0,5;

Опір заземлення визначається за формулою:

$$R_ç = \frac{\rho}{2\pi \cdot l}\left(\ln\frac{2 \cdot l}{d} + \frac{1}{2}\ln\frac{4 \cdot t + l}{4 \cdot t - l}\right) \qquad (4.4)$$

де ρ - питомий опір ґрунту, залежить від типу ґрунту, Ом·м;

для піску - 400 ÷ 700 Ом·м; приймаємо ρ = 400 Ом·м;

l - довжина заземлювача, м; для труб - 2-3 м; l = 3 м;

d - діаметр заземлювача, м; для труб - 0,03-0,05 м; d = 0,05 м;

t - відстань від середини забитого в ґрунт заземлювача до рівня землі, м; t = 2 м.

$$R_з = \frac{400}{2 \cdot 3{,}14 \cdot 3}\left(\ln\frac{2 \cdot 3}{0{,}05} + \frac{1}{2}\ln\frac{4 \cdot 2 + 3}{4 \cdot 2 - 3}\right) = 110 \text{, Ом}$$

Опір смуги, що з'єднує заземлювачі, визначається за формулою:

$$R_п = \frac{\rho}{2\pi \cdot L} \cdot \ln\frac{2 \cdot L^2}{b \cdot t_l} \qquad (4.5)$$

де L - довжина смуги, що з'єднує заземлювачі (м) і приблизно дорівнює периметру будівлі: $П_{буд.} = 42 \cdot 2 + 38 \cdot 2 = 160$ м; L = 160 м;

b - ширина смуги, м; b = 0,03 м;

t_1 - глибина заземлення від рівня землі, м; t_1 = 0,5 м.

$$R_п = \frac{400}{2 \cdot 3{,}14 \cdot 160} \cdot \ln\frac{2 \cdot 160^2}{0{,}03 \cdot 0{,}5} = 5{,}99 \text{, Ом}$$

Кількість заземлювачів захисного заземлення визначається за формулою:

$$n = \frac{2 \cdot R_ç}{4 \cdot \eta_ç} = \frac{2 \cdot 110}{4 \cdot 0{,}7} = 79 \text{ øò} \qquad (4.6)$$

де 4 - допустимий загальний опір, Ом;

2 - коефіцієнт сезонності.

Визначаємо загальний опір захисного заземлення:

$$R_{ззп} = \frac{110 \cdot 5{,}99}{5{,}99 \cdot 79 \cdot 0{,}7 + 110 \cdot 0{,}5} = 1{,}7 \text{, Ом}$$

Висновок: дане захисне заземлення буде забезпечувати електробезпеку будівлі, так як виконується умова: $R_{ззп} < 4$ Ом.

4.7 Охорона навколишнього природного середовища

Діяльність за темою магістерської роботи, а саме: Дослідження методів оптимізації продуктивності баз даних впливає на навколишнє природнє середовище і регламентується нормами діючого законодавства: Законом України «Про охорону навколишнього природного середовища», Законом України «Про забезпечення санітарного та епідемічного благополуччя населення», Законом України «Про відходи», Законом України «Про охорону атмосферного повітря», Законом України «Про захист населення і територій від надзвичайних ситуацій техногенного та природного характеру», Водний кодекс України.

Основним екологічним аспектом в процесі діяльності за даними спеціальностями є процеси впливу на атмосферне повітря та процеси поводження з відходами, які утворюються, збираються, розміщуються, передаються на видалення (знешкодження), утилізацію, тощо в ІТ галузі.

В процесі діяльності розробника об’єктної моделі за допомогою ПК виникають процеси поводження з відходами ІТ галузі. Нижче надано перелік відходів, що утворюються в процесі роботи:

Відпрацьовані люмінесцентні лампи - І клас небезпеки

Акумулятор для джерел безперебійного харчування -ІІІ клас небезпеки

Змінні носії інформації - IV клас небезпеки

Макулатура - IV клас небезпеки

Побутові відходи - IV клас небезпеки

Висновки до четвертого розділу

В результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, з якими стикається робітник. Було визначено параметри і певні характеристики приміщення для роботи над запропонованим проектом написаному в дипломній роботі, описано, які заходи потрібно зробити для того, щоб дане приміщення відповідало необхідним нормам і було комфортним і безпечним для робітника. Приведені рекомендації щодо організації робочого місця, а також важливу інформацію щодо пожежної та електробезпеки. Були наведені розміри приміщення та значення температури, вологості й рухливості повітря, необхідна кількість і потужність ламп та інші параметри, значення яких впливає на умови праці робітника, а також – наведені інструкції з охорони праці, техніки безпеки при роботі на комп'ютері.

А також визначені основні екологічні аспекти впливу на навколишнє природнє середовище та зазначені заходи щодо поводження з ними.

Перелік посилань до четвертого розділу

1. НПАОП 0.00-6.03-93 «Порядок опрацювання та затвердження власником нормативних актів про охорону праці, що діють на підприємстві»

2. НПАОП 0.00-4.12-05 «Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці»

3. НАПБ Б.02.005-2003 «Типове положення про інструктажі, спеціальне навчання та перевірку знань з питань пожежної безпеки на підприємствах, в установах та організаціях України»

4. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень»

5. ДСанПіН 3.3.2.007-98 «Правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин»

6. НПАОП 0.00.-1.28-10 «Правил охорони праці під час експлуатації електронно-обчислювальних машин»

7. ГОСТ 12.1.030-81 «ССБТ. Електробезпечність .Захисне заземлення. Занулення»

8. ГОСТ 13109-97 «Електрична енергія. Сумісність технічних засобів віелектромагнітних. Норми якості електроенергопостачання загального призначення »

9. ДБН В.2.5-28:2015 «Природнє і штучне освітлення»

10. НАПБ Б.03.002-2007 «Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою»

11. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень»

12. ДСанПіН 2.2.7.029 «Гігієнічні вимоги щодо поводження з промисловими відходами та визначення їх класу небезпеки для здоров'я населення».

ВИСНОВКИ

Розвиток існуючих систем вимагає розробки та вдосконалення механізмів оптимізації баз даних. Методи оптимізації, описані в роботі, є достатніми для визначення та усунення дефіциту продуктивності. Наявні механізми оптимізації допомагають мінімізувати ризики, пов'язані з тим, що система працює в виробничих середовищах. Загальний підхід до кожної проблеми повинен узгоджуватися з його правилами, але також треба приймати індивідуальні міркування і родити аналіз окремого випадку під час вдосконалення існуючої системи. Єдині процедури оптимізації для всіх проблем можуть призвести до їх неправильної класифікації, а отже, до неправильного методу їх вирішення. Контекст існуючої системи викликає значні обмеження на доступні методи оптимізації бази даних. Ділові ризики, пов'язані з підривом стабільності робочої системи, є важливим чинником остаточного вибору методів. Як правило, оптимізація прагне зменшити обсяг конкретних ресурсів, необхідних для отримання результату, що впливає на час виконання операторів SQL або фрагментів коду.

Подальші дослідження будуть стосуватись автоматизованих механізмів виявлення та швидкого реагування на проблеми продуктивності, основані на певних показаннях, які можуть передбачати проблему в майбутньому. Автоматизовані системи моніторингу в контексті методів оптимізації є відносно новим і маловивченим питанням, але можуть запобігти серйозним проблемам продуктивності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Benjamin Nevarez, High Performance SQL Server, Apress Inc., 2016.
2. Darl Kuhn, Sam R. Alapati and Bill Padfield, Expert Oracle Indexing and Access Paths, Apress Inc., 2016.
3. David Kroenke, David Auer. Database Concepts (3rd Edition). – M.: , 2007.
4. Hugh E Williams. Web Database Applications with PHP & MySQL. – M.: , 2002.
5. Ibrar Ahmed, Gregory Smith, Enrico P., PostgreSQL 10 High Performance, Amazon Digital Services LLC, 2017.
6. Koper, R., Analysis of the database optimization methods in the context of existing systems, Master's thesis, Lodz University of Technology, 2015.
7. Michael Widenius. MySQL Reference Manual. – M.: , 2002.
8. Peter Gulutzan, Trudy Pelzer. SQL Performance Tuning (1st Edition). – M.: 2012.
9. Powell, G., Oracle Performance Tuning for 10gR2, Elsevier Inc., 2007.
10. А.М. Вендров «Современные методы и средства проектирования информационных систем» – М.: Финансы и статистика, 1998. – 176 с.
11. Б. Шварц, П. Зайцев, В. Ткаченко, Дерек Дж. Бэллинг, MySQL. Оптимизация производительности, 3-е издание, Символ-Плюс, 2014.
12. Г. Калянов «Номенклатура CASE-средств и виды проектной деятельности» – М.: ЛОРИ, 1996, – 242с.
13. Илюшечкин В. И., Основы использования и проектирования баз данных, Гриф УМО, 2014.
14. Л.В. Рудикова, Базы данных. Разработка приложений. – СПб.: БХВ-Петербург, 2006.
15. Модели жизненного цикла разработки ИС – belani.narod.ru [Електронний ресурс] – Режим доступу: http://belani.narod.ru/1/S1.htm
16. Орлов С. А., Программная инженерия. Учебник для вузов. 5-е издание обновленное и дополненное. Стандарт третьего поколения. — СПб.: Питер, 2016. — 640 с.: ил, — (Серия «Учебник для вузов»).
17. Ржеуцкая С.Ю., Базы данных. Язык SQL: учебное пособие, ВоГТУ, 2013.

ДОДАТОК А. СТРУКТУРА ГОЛОВНОЇ ТАБЛИЦІ В БАЗІ ДАНИХ

Таблиця А Структура головної таблиці в базі даних

Схема	Таблиця	Атрибут	Тип
tours_db	tour_operators	id	INT(2)
tours_db	tour_operators	name	VARCHAR(50)
tours_db	Countries	id	INT(2)
tours_db	Countries	name	VARCHAR(70)
tours_db	Resorts	id	INT(3)
tours_db	Resorts	name	VARCHAR(150)
tours_db	Resorts	country_id	INT(2)
tours_db	Hotels	id	INT(4)
tours_db	Hotels	name	VARCHAR(250)
tours_db	Hotels	resort_id	INT(3)
tours_db	Hotels	category	INT(1)
tours_db	Hotels	other_info	BLOB(65535)
tours_db	tours_more	id	INT(5)
tours_db	tours_more	cost	INT(4)
tours_db	tours_more	hotel_id	INT(4)
tours_db	tours_more	tour_operator_id	INT(2)
tours_db	tours_more	departure_city_id	INT(1)
tours_db	tours_more	carrier_id	INT(3)
tours_db	tours_more	date_departure	DATE
tours_db	tours_more	duration	INT(1)
tours_db	tours_more	food	VARCHAR(500)
tours_db	tours_more	other_info	BLOB(65535)
tours_db	tours_more	adults_children	VARCHAR(200)
tours_db	tours_more	type_room	VARCHAR(1000)
tours_db	tours_not_available	id	INT(5)
tours_db	tours_not_available	date_adding	DATE
tours_db	tours_not_available	cost	INT(4)
tours_db	tours_not_available	hotel_id	INT(4)
tours_db	tours_not_available	tour_operator_id	INT(2)
tours_db	tours_not_available	departure_city_id	INT(1)
tours_db	tours_not_available	carrier_id	INT(3)
tours_db	tours_not_available	date_departure	DATE
tours_db	tours_not_available	duration	INT(1)
tours_db	tours_not_available	food	VARCHAR(500)
tours_db	tours_not_available	other_info	BLOB(65535)
tours_db	tours_not_available	adults_children	VARCHAR(200)
tours_db	tours_not_available	type_room	VARCHAR(1000)
tours_db	departure_city	id	INT(1)
tours_db	departure_city	name	VARCHAR(90)
tours_db	departure_city	country_id	INT(2)
tours_db	Carriers	id	INT(3)
tours_db	Carriers	name	VARCHAR(120)

ДОДАТОК Б. СЛАЙДИ ПРЕЗЕНТАЦІЇ

Дослідження методів оптимізації продуктивності баз даних

Виконав: Кучмистий О. Г.
Група: КН-17дм

Об'єкт дослідження: процес оптимізації запитів до бази даних, а також процес модифікації структури бази даних.

Предметом дослідження є алгоритми та рекомендації щодо оптимізації запитів, планів запитів, а також модифікації бази даних.

Метою роботи є вивчення методів оптимізації БД та впровадження їх в існуючу програмну систему, яка призначена для організації інформаційного супроводу туристичного агентства.

Завдання

- Зібрати інформацію про найбільш відомі причини уповільнення отримання інформації з бази даних.
- Провести серію експериментів, в ході яких виявити достовірність ефективності знайдених рекомендацій.
- Проаналізувати отримані результати, підбити підсумки.
- Провести необхідні впровадження в існуючу систему на основі проведених досліджень.

3

Основні поняття в процесі оптимізації ПС

- еталонне тестування (benchmarking);
- профілювання (profiling);
- ефективність;
- життєздатність.

4

Аналіз програмних та інструментальних засобів

- HackMySQL - Деніел Ніхтер (Daniel Nichter);
- mysqlsla (MySQL Statement Log Analyzer);
- Maatkit - Берона Шварца (Baron Schwartz);
- mkqueryprofiler;
- ...

5

Як працює SQL сервер

- журнал транзакцій;
- індекси;
  - кластерні індекси
  - некластерні індекси;
- двигун запитів і оптимізатор;
- кешування.

6

Апаратні компоненти продуктивності

- диск I/O;
- мережевий I/O;
- процесори;
- оперативна пам'ять.

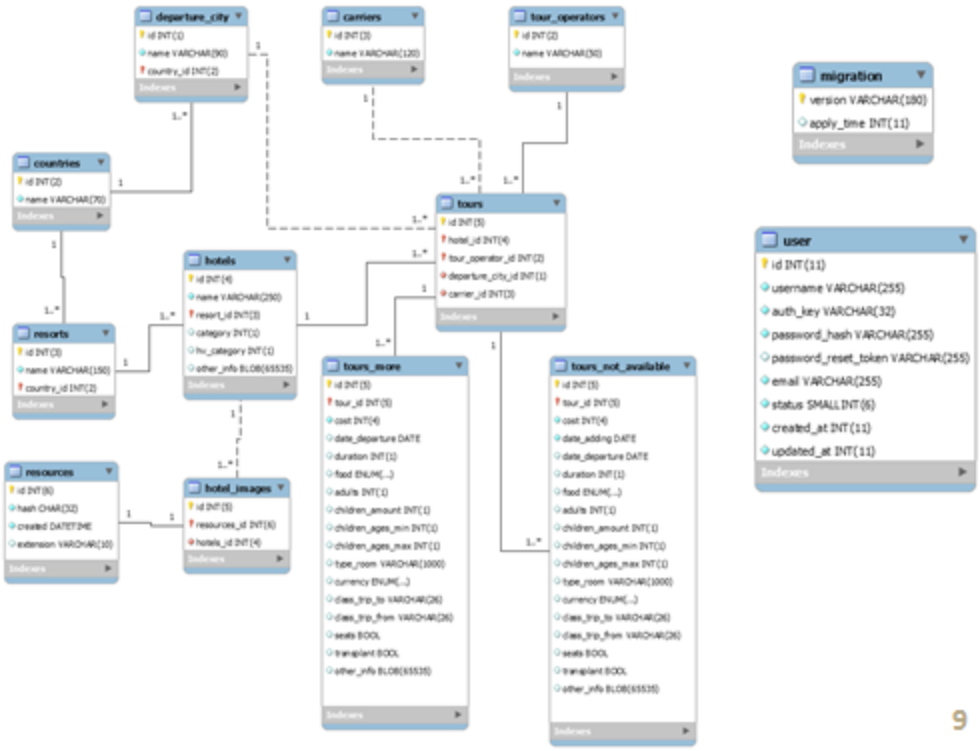
7

Концепції масштабування

- цілісне вдосконалення;
- устаткування сервера;
- параметри програмного забезпечення;
- проблеми проектування;
- рекомендації для клієнтів.

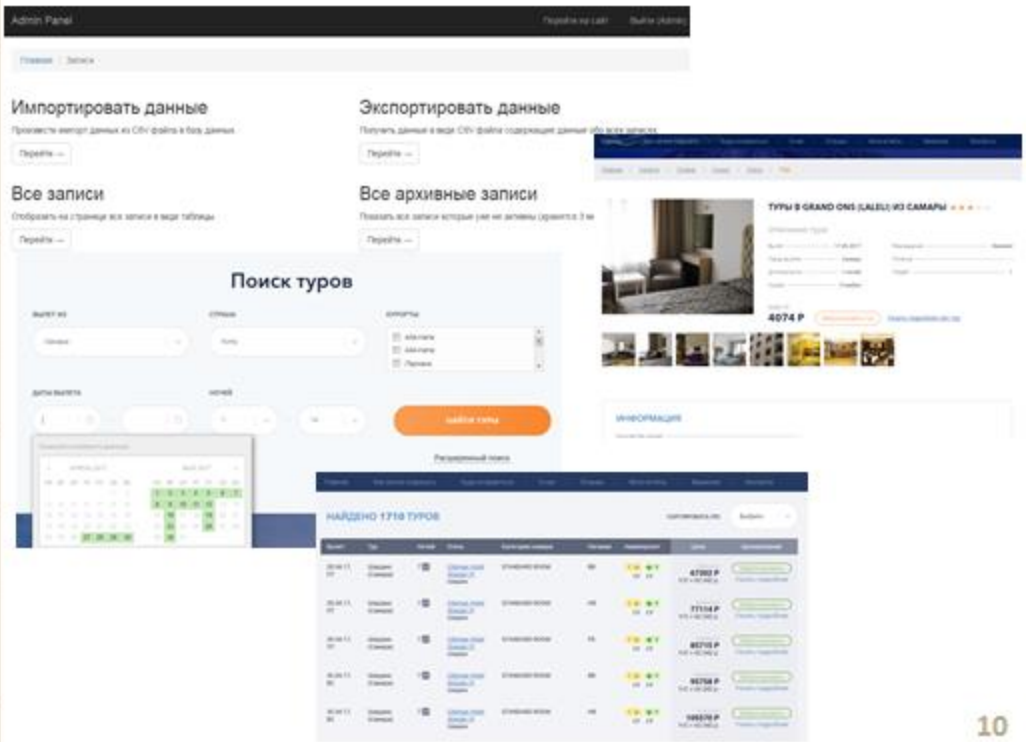
8

Програмна реалізація. Фізична модель БД



9

Програмна реалізація. Результат розробки



10

Висновки:

- єдині процедури оптимізації для всіх проблем можуть призвести до неправильного методу їх вирішення;
- контекст існуючої системи викликає значні обмеження на доступні методи оптимізації бази даних;
- ділові ризики, пов'язані з підривом стабільності робочої системи, є важливим чинником остаточного вибору методів;
- автоматизовані системи моніторингу в контексті методів оптимізації є відносно новим і маловивченим питанням.

11