Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Оптимізація функціонування аналітичних баз даних із використанням OLAP технології
(тема)

Виконав:
студент 2 курсу, групи КТРСм-21-1

Трофимов Н.С.
(прізвище, ініціали)

Спеціальності 151 Автоматизація та компютер'но-інтегровані технології
(код і повна назва спеціальності)

Тип програми Освітньо-професійна

Освітня програма Комп'ютеризовані та робототехнічні системи
(повна назва освітньої програми)

Керівник доц. Невлюдова В. В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри КІТАМ _____ Невлюдов І. Ш.
(підпис) (прізвище, ініціали)

2022 р.

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

«02» грудня 2022 р. *Трофимов Н.С.*

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет Автоматики і комп'ютеризованих технологій
Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки
Рівень вищої освіти другий (магістерський)
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
Тип програми Освітньо-професійна
Освітня програма Комп'ютеризовані та робототехнічні системи
(код і повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри КІТАМ
_____
(підпис)
«_____» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Трофимову Никиті Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Оптимізація функціонування аналітичних баз даних із використанням OLAP технології
Затверджена наказом по університету від 10.10.2022 р. № 740 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 23.12.2022 р.
3. Вихідні дані до роботи Функціонально оптимізована база даних для аналітичних задач. Перелік використовуваних програмних засобів: ОС Microsoft Windows 10 та вище. Технічне забезпечення: ПК з МП Intel i5 та вище
4. Перелік питань, що потрібно опрацювати в роботі
Огляд предметної області та постановка задачі, проаналізувати існуючі рішення кодування даних, визначення системи для того, щоб ефективно виконувати аналітичні запити, визначення реалізації механізму системи, знаходження ефективних інструментів для оптимізації функціонування бази даних, розробка власних алгоритмів для компресії даних та пришвидшення виконання запитів з агригацією

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій
Демонстраційний матеріал, представлений у форматі презентації PowerPoint (*.pptx) – 30 с.

6. Консультанти розділів роботи

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання на виконання	15.09.22	виконано
	кваліфікаційної роботи		
2	Визначення актуальності роботи, мети,	20.09 – 30.09.22	виконано
	предмета, об'єкта,та розробка завдань		
	для досягнення мети		
3	Аналіз літературних джерел	01.10 – 20.10.22	виконано
4	Вибір середовища розробки	20.10 – 15.11.22	виконано
5	Оформлення пояснювальної записки	15.11 – 06.12.22	виконано
6	Охорона праці	07.12 – 10.12.22	виконано
7	Оформлення додатків	11.12 – 13.12.22	виконано
8	Оформлення графічного та презентаційних	14.12 – 16.12.22	виконано
	матеріалів комп'ютерного захисту		
9	Подання роботи на перевірку	17.12-18.12.22	виконано
	Інтернет-сервісом Unicheck		
10	Подання роботи на рецензію	19.12.22	виконано
11	Подання роботи на підпис	21.12.22	виконано
	зав. Кафедри		
12	Подання кваліфікаційної роботи в ЕК	23.12.22	виконано

Дата видачі завдання 15.09.2022 р.

Студент _____ Трофимов Н.С._____
(підпис)

Керівник роботи _____ доц. Невлюдова В.В.
(підпис) (посада, прізвище, (ініціали)

РЕФЕРАТ

Пояснювальна записка: 105 с., 1 табл., 64 рис., 1 дод., 22 джерел.

ВЕЛИКІ БАЗИ ДАНИХ, OLTP, OLAP, COLUMNSTORE INDEX ROWGROUPS, ВЕКТОРИ, ОПТИМІЗАЦІЯ

Мета роботи – оптимізація функціонування аналітичних баз даних, покращення алгоритмів компресії даних та пришвидшення запитів з агригацією.

Об’єкт дослідження – процес оптимізації аналітичних баз даних.

Предмет дослідження – розробка методів оптимізації для аналітичних баз даних.

Методи дослідження – системний аналіз, методи і засоби збору та обробки даних.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– проаналізувати існуючі рішення;

– визначення системи для того, щоб ефективно виконувати аналітичні запити;

– визначення реалізації механізму системи;

– знаходження ефективних інструментів для оптимізації функціонування бази даних;

– розробка власних алгоритмів для компресії даних та пришвидшення виконання запитів з агригацією.

ABSTRACT

Explanatory note: 105 pp., 64 fig., 1 tabl., 1 adj., 22 sources.

LARGE DATABASES, OLTP, OLAP, COLUMNSTORE INDEX ROWGROUPS, VECTORS, OPTIMIZATION

Purpose – to optimize the functioning of analytical databases, improve data compression algorithms and speed up queries with aggregation.

Object of research – the process of optimization of analytical databases.

Subject of research – development of optimization methods for analytical databases.

Research methods – system analysis, methods and means of data collection and processing.

To achieve this goal it is necessary to solve the following tasks:

– analyze existing solutions;

– definition of the system in order to effectively perform analytical queries;

– determining the implementation of the system mechanism;

– finding effective tools to optimize the functioning of the database;

– development of own algorithms for data compression and acceleration of query.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

РБД – реляційні бази даних

АБД – аналітичні бази даних

СУБД – система управління базами даних

КД – компресія даних

OLTP (Online Transaction Processing) – онлайнова обробка транзакцій

OLAP (Online Analytical Processing) – аналітична обробка у реальному часі

MOLAP (Multidimensional OLAP) – Багатовимірна аналітична обробка у реальному часі

ROLAP (Relational OLAP) – реляційна аналітична обробка у реальному часі

HOLAP (Hybrid OLAP) – гібридна аналітична обробка у реальному часі

ВСТУП

Аналітика бізнесу – це завжди було і буде вдалою річчю для того, щоб покращити фінансове становище бізнесу, а також репутацію компанії. Для того, щоб вдало аналізувати бізнес, необхідно зберігати всю основну інформацію про прибутки, про витрати та інші речі, які впливають на існування бізнесу для подальшого аналізу. Для збереження цих даних обирають завжди бази даних. Але для аналітики необхідно обрати не звичайні бази даних, а аналітичні. Тобто перед тим як створювати базу даних, необхідно спочатку визначити потреби клієнта щодо аналітики свого бізнесу. Після чого вже починати створювати архітектуру бази даних та ETL-процесу.

Зазвичай в аналітичних базах даних завжди існує проблема зі зберіганням великої кількості даних та швидкістю виконання запитів при процесі OLAP-системи. Тому існує потреба в оптимізації функцонування бази даних. Спочатку бажано вибрати реалізацію механізму OLAP-системи. Кожна з реалізацій підходить для різних рішень та вимог клієнтів щодо економії міста чи швидкодії системи.

Для того, щоб зменшити розміри таблиць зазвичай використовують Columnstore індекси, для яких виконуються алгоритми компресії даних для різних типів даних та різних ситуацій. Але не завжди існуючі алгоритми можуть буду вдалими для якихось ситуацій, тому було принято рішення розробити власні алгоритми компресії та пришвидшення запитів з агригацією для ситуацій, коли можна покращити чи замінити вже існучі алгоритми.

Пояснювальну записку оформлено згідно з рекомендаціями та вимогами ДСТУ 3008:2015 [1], методичними вказівками [2] – [3].

Об’єкт дослідження – процес оптимізації аналітичних баз даних.

Предмет дослідження – розробка методів оптимізації для аналітичних баз даних.

Методи дослідження – системний аналіз, методи і засоби збору та

обробки даних.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– проаналізувати існуючі рішення;

– визначення системи для того, щоб ефективно виконувати аналітичні запити;

– визначення реалізації механізму системи;

– знаходження ефективних інструментів для оптимізації функціонування бази даних;

– розробка власних алгоритмів для компресії даних та пришвидшення виконання запитів з агригацією.

Результати кваліфікаційної роботи використовувались в ході проведення VI-ої Міжнародної Науково-практичної Конференції «Актуальні проблеми сучасної науки, суспільства та освіти» [4].

1 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ ТЕХНОЛОГІЙ ТА СИСТЕМ, ЯКІ ВИКОРИСТОВУЮТЬСЯ В РЕЛЯЦІЙНИХ ТА АНАЛІТИЧНИХ БАЗАХ ДАНИХ

1.1 Реляційні бази даних

Для того щоб краще розуміти недоліки використання традиційних, реляційних баз даних у випадку, коли стоїть пріоритет швидкого вилучення великої кількості інформації у реальному часі, необхідно розібратись, що саме із себе представляють реляційні бази даних: як фізично зберігаються данні, яким чином виконуються запити до такої системи зберігання, які існують методи вирішення проблем реляційних баз даних (РБД) [5].

1.1.1 Формулювання терміну та історія створення реляційних баз даних

Реляційна база даних – це сукупність інформації, яка організовує дані в заздалегідь визначених відносинах, де дані зберігаються в одній або декількох таблицях (або "відносинах") стовпців і рядків, що дозволяє легко бачити і розуміти, як різні структури даних пов'язані один з одним. Зв'язки – це логічний зв'язок між різними таблицями, встановлений на основі взаємодії між цими таблицями.

Розроблена Е.Ф. Коддом з компанії IBM у 1970-х роках, реляційна модель бази даних дозволяє будь-якій таблиці бути пов'язаною з іншою таблицею за допомогою спільного атрибуту. Замість використання ієрархічних структур для організації даних, Кодд запропонував перехід до використання моделі даних, в якій дані зберігаються, отримують доступ і пов'язані в таблицях без реорганізації таблиць, які їх містять.

Простіше уявити собі РБД як набір файлів електронних таблиць, які

допомагають бізнесу організовувати, управляти та пов'язувати дані. У моделі РБД кожна "електронна таблиця" – це таблиця, яка зберігає інформацію, представлену у вигляді стовпців (атрибутів) і рядків (записів або кортежів).

Атрибути визначають тип даних, а кожен рядок містить значення цього конкретного типу даних. Всі таблиці в реляційній базі даних мають атрибут, відомий як первинний ключ, який є унікальним ідентифікатором рядка. Кожен рядок може бути використаний для створення зв'язку між різними таблицями за допомогою зовнішнього ключа – посилання на первинний ключ іншої існуючої таблиці.

Прикладом РБД може бути звичайна таблиця, де можна побачити зв’язок між рядком та атрибутом (рис. 1.1).

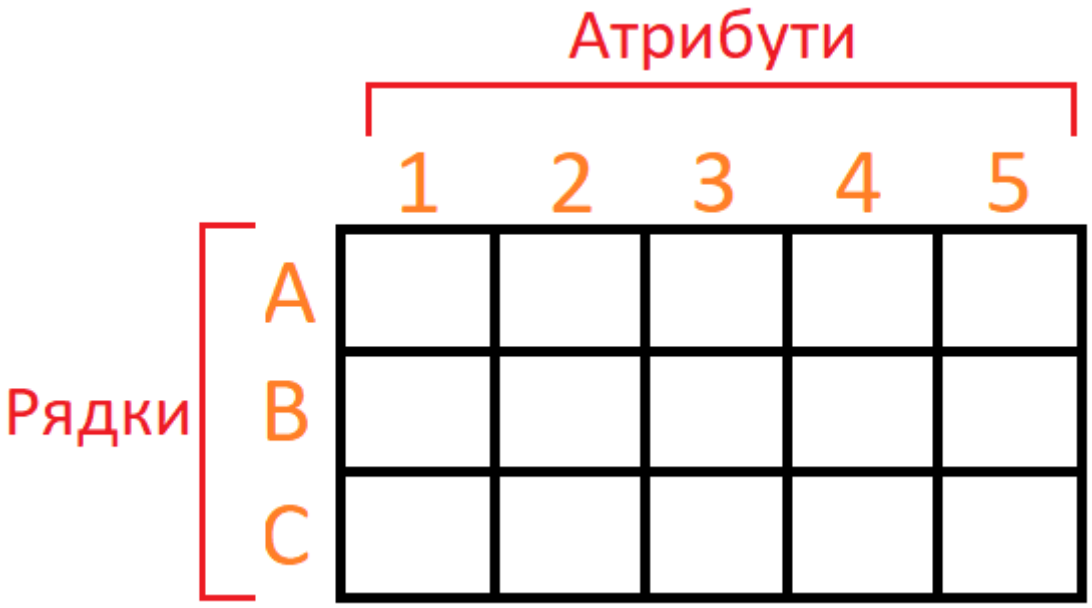

Рисунок 1.1 – Ілюстрація зяв’зку між тюплами та атрибутами

На зображенні (рис 1.1) можна побачити сам принцип та простоту уявлення даних у вигляді зв’язку між стовпчиками та рядками, як у звичайних таблицях. Тобто можна побачити, наприклад, що атрибут «1» має зв’язок зі всіма іншими тюплами {A,B,C}, і такий же принцип для всіх інших атрибутів

таблиці. Важливо розуміти і побачити, що окремо якийсь атрібут чи тюпл не може інсувати. У цьому точно можна переконатись, коли детальніше зрозуміти як фізично зберігаються дані у такій системі.

1.1.2 Фізичне зберігання даних реляційних БД

Самим головним недоліком використання РБД для зберігання великої кількості інформацію задля подальшого аналітичного процесу у режимі реального часу, є саме система чи, коректніше сказати, архітектура фізичного зберігання даних.

Дані зберігаються у так званих pages (пейджах/сторінках). Фундаментальною одиницею зберігання даних в SQL Server є сторінка. Дисковий простір, виділений під файл даних (.mdf або .ndf) в БД, логічно розбивається на сторінки, які нумеруються послідовно від 0 до n. Операції вводу-виводу на диску виконуються на рівні сторінок. Тобто SQL Server читає або записує цілі сторінки даних. Екстенти являють собою набір з восьми фізично суміжних сторінок і використовуються для ефективного управління сторінками. Всі сторінки організовані в екстенти.

У звичайній книзі весь вміст записаний на сторінках. Подібно до книги, SQL Server записує всі рядки даних на сторінки, і всі сторінки даних мають однаковий розмір: 8 КБ. У книзі більшість сторінок містять дані – основний зміст книги – і деякі сторінки містять метадані про зміст, наприклад: зміст і покажчик. Знову ж таки, SQL Server нічим не відрізняється: більшість сторінок містять фактичні рядки даних, які були збережені користувачами; вони називаються сторінками даних і сторінками тексту/зображень (для особливих випадків). Індексні сторінки містять індексні посилання на те, де знаходяться дані. Нарешті, є системні сторінки, які зберігають різні метадані про організацію даних.

Кожна сторінка починається з 96-байтового заголовка, який використовується для зберігання системної інформації про сторінку. Ця

інформація включає номер сторінки, тип сторінки, кількість вільного місця на сторінці та ідентифікатор одиниці розподілу об'єкта, якому належить сторінка.

Треба також мати на увазі, що дані не зберігаються у вигляді пейджів/сторінок. Вони містять серію записів, які не мають фіксованого розміру.

Рядки даних зберігаються на сторінці послідовно, починаючи відразу після заголовка. Кожен запис про зсув рядка зберігає інформацію про те, на якій відстані знаходиться перший байт рядка від початку сторінки. Таблиця зсуву рядків починається в кінці сторінки і кожна таблиця зсуву рядків містить один запис для кожного рядка на сторінці. Таким чином, функція таблиці зсуву рядків полягає в тому, щоб допомогти SQL Server швидко знаходити рядки на сторінці. Записи в таблиці зсуву рядків розташовуються в зворотній послідовності від послідовності рядків на сторінці (рис. 1.2).

Рисунок 1.2 – Архітектура фізичного зберігання даних

Рядки не можуть охоплювати сторінки; однак, частини рядка можуть бути перенесені за межі сторінки, тому рядок може бути дуже великим.

Максимальний обсяг даних і накладних витрат, що містяться в одному рядку на сторінці становлять 8 060 байт. Це не включає дані, що зберігаються на сторінках типу текст/зображення.

Це обмеження послаблюється для таблиць, що містять стовпці типу varchar, nvarchar, varbinary або sql_variant. Коли загальний розмір рядка всіх фіксованих і змінних стовпців у таблиці перевищує обмеження в 8 060 байт, SQL Server динамічно переміщує один або кілька стовпців змінної довжини на сторінки в блоці розподілу ROW_OVERFLOW_DATA, починаючи зі стовпця з найбільшою шириною.

Це робиться щоразу, коли операція вставки або оновлення збільшує загальний розмір рядка понад 8 060 байт. Коли стовпець переміщується на сторінку в блоці розміщення ROW_OVERFLOW_DATA, зберігається 24-байтовий покажчик на вихідну сторінку в блоці розміщення IN_ROW_DATA. Якщо наступна операція зменшує розмір рядка, SQL Server динамічно переміщує стовпці назад на вихідну сторінку даних.

Рядок не може розміщуватися на декількох сторінках і може переповнитися, якщо сумарний розмір полів типу даних змінної довжини перевищує ліміт 8060 байт. Для ілюстрації можна створити таблицю з двома стовпчиками: один varchar (7000), інший varchar (2000). Окремо жоден стовпець не перевищує 8060 байт, але разом вони можуть перевищити, якщо заповнити всю ширину кожного стовпця. SQL Server може динамічно переміщати стовпець змінної довжини varchar (7000) на сторінки в блоці розподілу ROW_OVERFLOW_DATA. При об'єднанні стовпців varchar, nvarchar, varbinary, або sql_variant або стовпців визначеного користувачем типу CLR, які перевищують 8,060 байт на рядок, треба враховувати багато речей, про які буде опис нижче.

Переміщення великих записів на іншу сторінку відбувається динамічно, оскільки записи подовжуються на основі операцій оновлення. Операції оновлення, які вкорочують записи, можуть призвести до того, що записи будуть переміщені назад на початкову сторінку в блоці розміщення

IN_ROW_DATA. Запит і виконання інших операцій вибірки, таких як сортування або об'єднання над великими записами, які містять дані, що переповнюють рядок, уповільнює час обробки, оскільки ці записи обробляються синхронно, а не асинхронно. Тому, при проектуванні таблиці з декількома стовпцями varchar, nvarchar, varbinary, або sql_variant, або CLR користувацького типу, треба враховувати відсоток рядків, які, ймовірно, будуть переповнюватися, і частоту, з якою ці переповнені дані, ймовірно, будуть запитуватися. Якщо існує ймовірність частих запитів до багатьох рядків даних, що переповнюються, краще розглянути можливість нормалізації таблиці таким чином, щоб деякі стовпці були перенесені в іншу таблицю. Потім ці дані можуть бути запитані в асинхронній операції JOIN.

Довжина окремих стовпців все одно повинна бути в межах 8 000 байт для стовпців типу varchar, nvarchar, varbinary або sql_variant, а також стовпців користувацького типу CLR. Тільки їх сукупна довжина може перевищувати 8 060-байтний ліміт рядка таблиці. Сума стовпців інших типів даних, включаючи дані типу char та nchar, повинна не перевищувати 8 060 байт у рядку. Великі об'єктні дані також не підпадають під обмеження в 8 060 байт.

Індексний ключ кластерного індексу не може містити стовпців типу varchar, які мають існуючі дані в блоці розміщення ROW_OVERFLOW_DATA. Якщо кластерний індекс створено на стовпці varchar, а існуючі дані знаходяться в блоці розміщення IN_ROW_DATA, то подальші дії вставки або оновлення стовпця, які б виштовхували дані за межі рядка, будуть невдалими. Стовпці, що містять дані про переповнення рядків, можна включати як ключові або неключові стовпці некластеризованого індексу.

Обмеження розміру запису для таблиць, що використовують розріджені стовпці, становить 8 018 байт. Коли перетворені дані плюс дані існуючого запису перевищують 8 018 байт, повертається помилка «MSSQLSERVER ERROR 576». Коли стовпці перетворюються між розрідженими та нерозрідженими типами, механізм бази даних зберігає копію даних поточного

запису. Це тимчасово подвоює обсяг пам'яті, необхідний для зберігання запису.

Екстенти є основною одиницею управління простором. Екстентом є вісім фізично суміжних сторінок, або 64 КБ. Це означає, що бази даних SQL Server мають 16 екстентів на мегабайт. SQL Server має два типи екстентів:

– єдині екстенти;

– змішані екстенти.

Єдині екстенти належать одному об'єкту, всі вісім сторінок в екстенті можуть бути використані тільки об'єктом-власником. Змішані екстенти поділяють до восьми об'єктів. Кожна з восьми сторінок розширення може належати різним об'єктам (рис. 1.3).

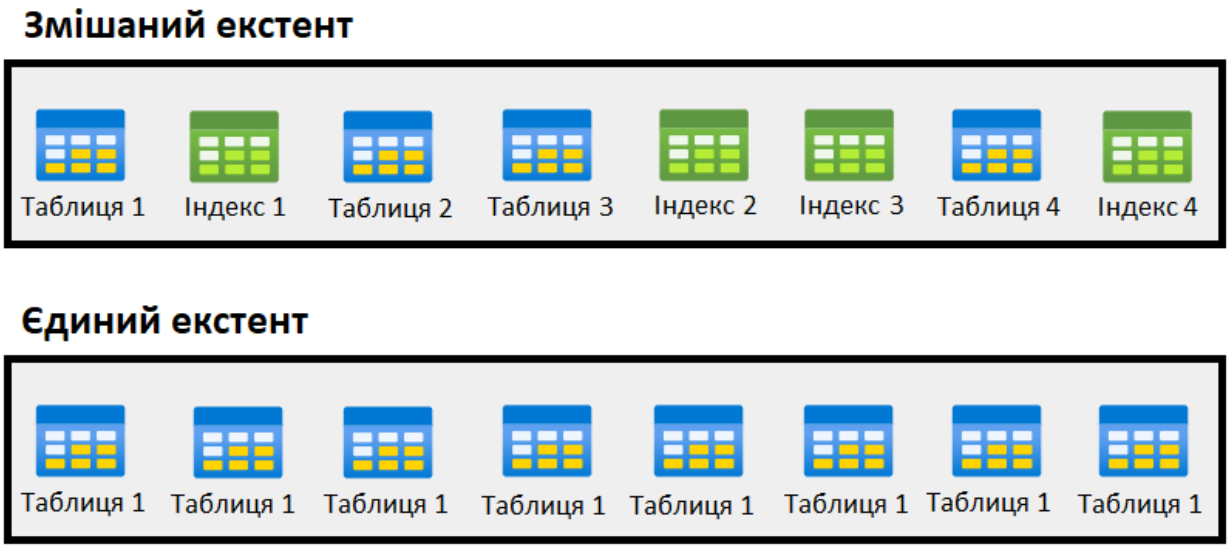

Рисунок 1.3 – Типи екстентів

До версії SQL Server 2014 механізм БД не виділяє цілі розширення для таблиць з невеликими обсягами даних. Нова таблиця або індекс, як правило, виділяє сторінки зі змішаних розширень. Коли таблиця або індекс розростається до восьми сторінок, вона переходить на використання однорідних екстентів для подальшого розподілу. Якщо створюється індекс на існуючій таблиці, яка має достатньо рядків, щоб згенерувати вісім сторінок в

індексі, всі розподіли для індексу виконуються в рівномірних розширеннях.

Починаючи з SQL Server 2016 (13.x), за замовчуванням для більшості розміщень в базі даних користувача і tempdb використовується єдине розширення, за винятком розміщень, що належать до перших восьми сторінок ланцюжка IAM. Розміщення для баз даних master, msdb і model зберігають попередню поведінку.

Структури даних SQL Server, які керують розподілом екстентів і відстежують вільний простір, мають відносно просту структуру. Це має наступні переваги:

а) інформація про вільний простір щільно упакована, тому відносно невелика кількість сторінок містить цю інформацію. Це збільшує швидкість за рахунок зменшення кількості зчитувань з диска, необхідних для отримання інформації про розподіл. Це також збільшує ймовірність того, що сторінки розподілу залишаться в пам'яті і не потребуватимуть додаткових зчитувань;

б) більшість інформації про розподіл не пов'язана між собою. Це спрощує обслуговування інформації про розподіл. Кожен розподіл або перерозподіл сторінок може бути виконаний швидко. Це зменшує конфлікт між паралельними завданнями, які повинні розподіляти або перерозподіляти сторінки.

1.2 OLTP-система

Оперативна обробка транзакцій (OLTP) – це управління даними про транзакції за допомогою комп'ютерних систем. Системи OLTP записують операції обміну даними в організації, що виконуються щодня і підтримують запитування цих даних, щоб на їхній основі робити висновки.

Дані про транзакції – це відомості, отримані в результаті відстеження взаємодій, пов'язаних з діяльністю організації. Як правило, це бізнес-транзакції, наприклад: отримані від клієнтів платежі, відправлені постачальникам платежі, надходження продуктів на склад або їхнє

переміщення зі складу, оформлені замовлення або надані послуги. Події транзакцій, які самі по собі є транзакціями, зазвичай містять вимір часу, деякі числові значення і посилання на інші дані. Транзакції зазвичай мають бути атомарними та узгодженими. Атомарність означає, що транзакція завжди повністю завершується успішно або невдало як одна елементарна операція і ніколи не залишається в стані часткової завершеності. Якщо не вдається виконати транзакцію, система БД повинна відкотити всі кроки, які вже виконані в рамках цієї транзакції. У традиційній реляційній систем управління базами даних (СУБД), якщо не вдалося завершити транзакцію, відкат відбувається автоматично. Узгодженість означає, що після виконання транзакції дані завжди залишаються в допустимому стані [6].

У транзакційних БД сувора узгодженість транзакцій може підтримуватися за допомогою різних стратегій блокування, наприклад: песимістичного блокування. Це забезпечує сувору узгодженість усіх даних у контексті підприємства для всіх користувачів і процесів.

Рівень сховища даних у трирівневій архітектурі найчастіше використовується для розгортань із використанням даних про транзакції. Як правило, трирівнева архітектура складається з рівня подання, рівня бізнес-логіки та рівня сховища даних. Відповідна архітектура розгортання – це n-рівнева архітектура, у якій може бути кілька середніх рівнів для бізнес-логіки (табл. 1.1).

Таблиця 1.1 – Характеристика даних о транзакціях

Вимоги	Опис
Нормалізація	Висока степінь нормалізації
Схема	Схема при записі (суворе дотримання)

Продовження таблиці 1.1

Робоче навантаження	Велика кількість операцій запису, середня кількість операцій читання
Розмір даних	Невеликий і середній розмір
Моделювання	Реляційний
Форма подання даних	Таблиця
Гнучкість запитів	Висока гнучкість
Стратегія блокування	Песимістична чи оптимістична
Використання транзакцій	Так
Цілісність	Високий ступінь цілісності даних
Можливість оновлення	Так
Можливість додавання	Так

OLTP можна використати так, щоб забезпечити ефективне опрацювання та зберігання бізнес-транзакцій, а також швидко та узгоджено надавати доступ до них для клієнтських додатків. Використовуйте цю архітектуру, якщо будь-які відчутні затримки під час обробки негативно вплинуть на повсякденну роботу організації [7].

Системи OLTP призначені для ефективного опрацювання та зберігання транзакцій, а також для запитування даних про транзакції. Ефективне опрацювання та зберігання окремих транзакцій у системі OLTP частково досягається шляхом нормалізації даних: поділу даних на більш дрібні та менш надлишкові блоки. Це забезпечує ефективність, оскільки система OLTP може незалежно обробляти велику кількість транзакцій і дає змогу уникнути додаткового опрацювання, необхідного для підтримання цілісності даних за наявності надлишкових даних.

Отже, вже стає зрозуміло, що звичайні (відомі) системи та архітектура БД не підходить для функціонування великого датасету (набору даних), тому

що в аналітичних реаліях дані потрібно отримувати швидко, в реальному часі для того, щоб якісно проводити аналітичні роботи чи ще щось, що потребує швидкого екстракту даних за якихось умов із таблиці або таблиць. І так як бізнес робить аналітику даних пріоритетом, як ніколи раніше. Так, згідно з дослідженням ринку бізнес-аналітики Dresner Advisory 2021 року, організації лише у сфері роздрібної/оптової торгівлі, фінансових послуг та технологій планують збільшити свої річні бюджети на бізнес-аналітику на 50 % і більше. В цьому буде корисне використання великих даних для аналізу – це, безумовно, шлях, яким хочуть піти конкурентоспроможні компанії, щоб поліпшити процес прийняття рішень. Але отримання цінності з обсягу даних, який зростає з кожним днем, вимагає, щоб компанії спочатку були досвідченими в управлінні даними, щоб вони могли виконувати якісний аналіз даних. Але для цих цілей не підходить традиційне використання РБД чи баз даних які підтримуються OLTP системою, які більш підходять для частого оброблення даних: зміни, видалення, тощо. А ось сховища даних дуже добре підходить для вирішення таких бізнес-задач, які потребують швидкого отримання агрегаційної інформації чи якісь історичні данні з величезної БД. Сховище даних зберігає та організовує різні типи даних: історичні, операційні, дані обробки транзакцій та метадані з різних бізнес-процесів для аналітичного використання, покращуючи доступність даних та підвищуючи здатність бізнесу приймати оптимальні рішення [8].

Також дуже важливо мати на увазі, що аналітичних баз даних (АБД) зазвичай важать дуже багато, бо можуть існувати таблиці, які мають по 5, 10, 100+ мільйонів рядків, на які потрібно виділяти дуже багато місця на серверах. І якщо нехтувати такими інструментами як columnstore індекс, то це точно вплине на працездатність всієї системи, пов’язаної з цією БД.

Гарним прикладом буде наступні зображення (рис. 1.4 – рис. 1.5) OLTP-система дуже вигідна і хороша, коли нам потрібно вилучити якийсь рядок зі зберігання даних (купа або кластерний індекс), але коли нам потрібно вилучити інформацію тільки за якоюсь колонкою, то для цього потрібно

зчитувати абсолютно всі пейджі і знаходити там шматочки цієї потрібної нам колонки.

Рисунок 1.4 – Вилучення рядку із сторінки/пейджі

В таких випадках (рис.1.4) звичайні системи добре підходять, коли нам потрібно зчитувати якісь рядки, саме в цих випадках OLTP-системи дуже чудове себе відчувають, але у прикладі нижче (рис.1.5), наводиться весь недолік цієї системи зберігання даних. Тобто коли у нас багато даних (мільйони рядків), тоді потрібно зчитати всі пейджи і вилучити звідти ось ці дані за якоюсь однією колонкою, що в підсумку відбуватиметься дуже довго.

Рисунок 1.5 – Вилучення даних колонки із сторінки/пейджі

Цю проблему, звісно, можна вирішити не кластерним індексом за цією колонкою, але тоді буде теж неприємна ситуація для продуктивності та ресурсів системи, яка зберігає базу даних. БД зростатиме в розмірі, оскільки ці створені індекси – це копія даних, яка зберігається окремо від таблиці.

1.3 Висновки до першого розділу

У першому розділі були розглянуті існуючі системи та технології РБД та АБД. Було розглянута архітектура зберігання даних без наявності індексів у РБД: як саме зберігаються дані на пейджах, як пейджі поділяються на групи та скільки місця займає один пейдж у кілобайтах. Було виявлено, що пейдж – це мінімальна одиниця, яку процесор може зчитати за один раз, тобто менше ніж 8 Кб процесор не може зчитати при роботі з БД. Розглянуто недоліки зберігання даних у форматі кучі (heap) для великих баз даних.

Була оглянута OLTP-система, яка підходить для транзакційної моделі БД та добре проявляє себе з використанням наступних DML-операцій:

– INSERT;

– DELETE;

– UPDATE;

– MERGE.

2 РІШЕННЯ С ПРИВОДУ ОБРАННЯ МЕХІНЗМІВ OLAP-СИСТЕМИ. АРХІТЕКТУРА COLUMNSTORE ІНДЕКСІВ

АБД – це сховище, доступне лише для читання, яке збирає історичні дані, пов'язані з ключовими показниками ефективності діяльності та метриками, такими як продажі, продуктивність та запаси. Для організацій вона створює легкодоступну систему для будь-якого відповідного працівника або зацікавленої сторони для пошуку відповідних даних, виконання запитів та створення звітів на основі існуючих даних. Хоча вона не працює так само як БД у режимі реального часу, вона постійно оновлюється, оскільки нові дані збираються з відповідних потоків даних організації [9].

АБД створюються для бізнес-аналітики та аналізу великих даних і зазвичай функціонують як частина великих сховищ даних. Вони популярні, оскільки пропонують більш швидкий час виконання запитів, простіше обслуговування та легше масштабування завдяки своїй менш мінливій природі. АБД відрізняються від транзакційних (або OLTP) баз даних, які обробляють транзакції та інші операційні програми.

Важливо пам'ятати, що термін "аналітична база даних" може відноситися до різних стилів баз даних. До них відносяться: стовпчасті БД, які організовують дані в стовпці для зменшення кількості точок даних, що підлягають обробці; додатки сховищ даних, які включають інструменти баз даних на одній платформі; БД в пам'яті, які використовують системну пам'ять для прискорення обробки; БД MPP, які використовують кілька кластерів серверів, що працюють одночасно; і БД аналітичної обробки в режимі онлайн, які зберігають куби даних, що можуть бути проаналізовані на основі декількох параметрів.

Одним з основних випадків використання АБД є створення більш швидкої системи запитів для організацій.

Зростаюче значення даних, а також величезний обсяг даних, які

виробляються організаціями щодня, означає, що простий аналіз даних, які надходять до сховища, може бути неефективним і марнотратним.

Натомість, структуровані БД в рамках існуючих сховищ або як окремі додатки роблять дані більш доступними і полегшують взаємодію з ними для будь-якого члена компанії.

Можливо, найпоширенішим використанням АБД є виконання більш широкої, більш всеосяжної аналітики, ніж можуть забезпечити транзакційні БД. У той час як транзакційні БД зосереджені на повсякденній діяльності та операціях, використання аналітичної БД для запитів дозволяє задавати питання "що, якщо" і робити прогнозний аналіз більш ефективно, пропонуючи при цьому більш широкі можливості для пошуку корисної інформації з історичних даних.

Найважливіше те, що АБД – це зручний спосіб надати кожному учаснику організації легкий доступ до даних для самообслуговування аналітики та конкретних запитів.

Тобто побудова для бізнес-аналітики БД, яка б задовольняла умовам оптимізованого зберігання інформації, агригацій, тощо, також необхідність створення архітектури, яка б дозволяла як найшвидше виконувати запити до БД, а також знаходження інструментів для поліпшення роботи не тільки клієнтів, бізнес аналітиків чи тестувальників, а ще й інженерів є основною задачею оптимізації.

2.1 OLAP-система

Механізм OLAP є на сьогодні одним із популярних методів аналізу даних [10]. Є два основні підходи до вирішення цього завдання. Перший з них називається Multidimensional OLAP (MOLAP). MOLAP – реалізація механізму за допомогою багатовимірної БД на стороні сервера. Другий Relational OLAP (ROLAP) – побудова кубів "на льоту" на основі SQL-запитів до реляційної СУБД. Також існує третій підхід, який має назву Hybrid OLAP (HOLAP)

Кожен із цих підходів має свої плюси і мінуси.

Але перед тим як почати розбір цих підходів, необхідно розібрати загальну схему роботи OLAP-системи (рис. 2.1).

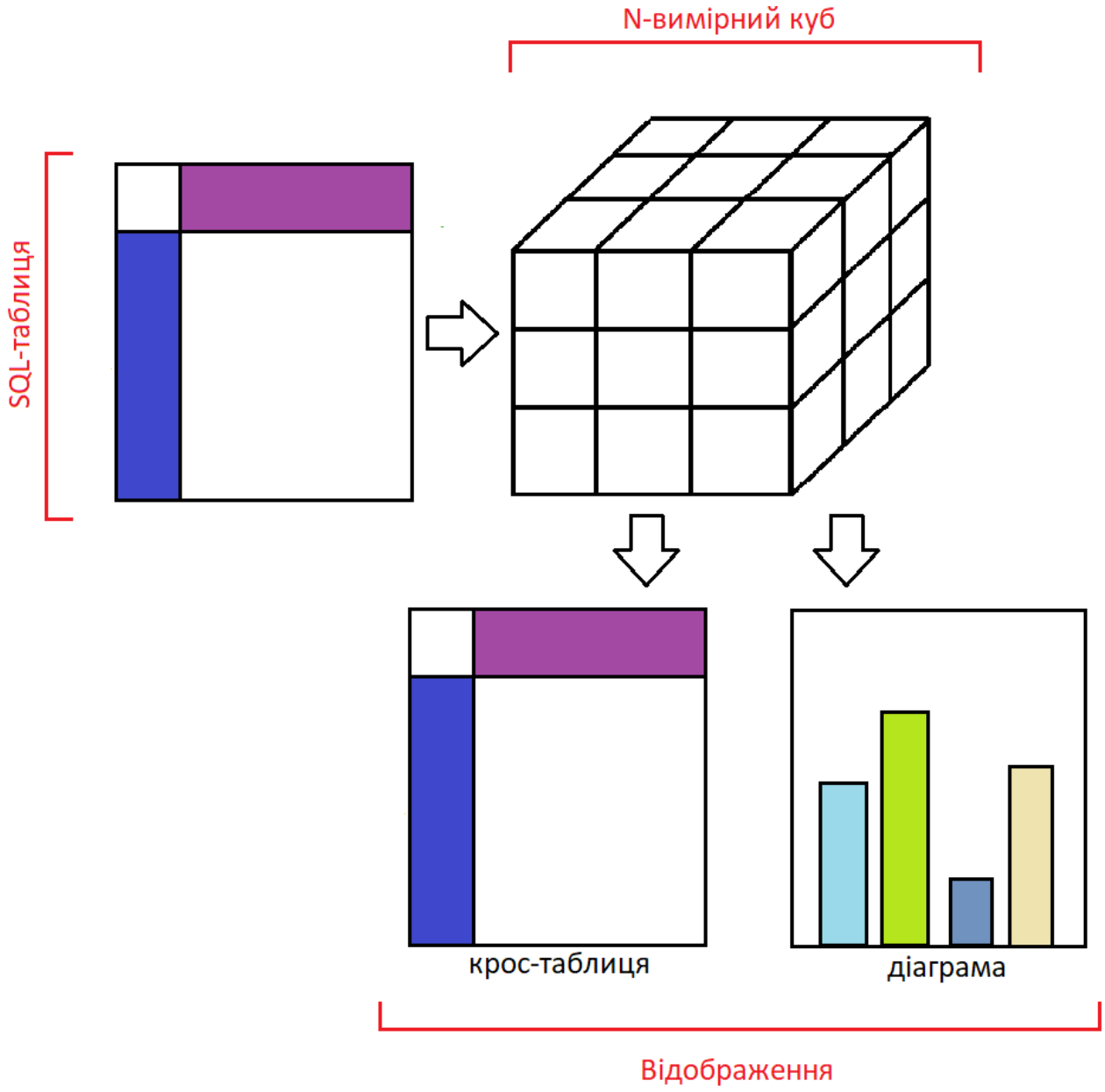

Рисунок 2.1 – Загальна схема функціонування системи OLAP

2.1.1 Реалізація механізму за допомогою MOLAP

MOLAP (Multidimensional OLAP) – багатовимірне концептуальне

представлення. Являє собою множинну систему, що складається з декількох незалежних вимірів уздовж яких можуть бути проаналізовані певні сукупності даних. Одночасний аналіз за кількома вимірами визначається як багатовимірний аналіз.

Описувана система найбільш наочна і зручна в обігу, оскільки дає змогу відображати і розглядати будь-які взаємозв'язки в найскладніших багатокомпонентних системах. Детальні й агреговані дані містяться в багатовимірній базі. Зберігання даних у багатовимірних структурах дає змогу маніпулювати даними як багатовимірним масивом, завдяки чому швидкість обчислення агрегатних значень однакова для будь-якого з вимірів. MOLAP передбачає створення явного, фізично збереженого багатовимірного куба (або декількох кубів) з виконанням аналітичних запитів тільки над ними, без звернення до реляційної БД. У цьому випадку досягається найбільша продуктивність, проте в цьому випадку багатовимірна БД виявляється надлишковою, оскільки багатовимірні дані повністю містять детальні реляційні дані [11].

З огляду на все, розглянуте раніше, використання системи MOLAP доцільне тільки за таких умов:

– невеликий обсяг вихідних даних для аналізу (не більше кількох гігабайтів), тобто рівень агрегації даних досить високий;

– набір інформаційних вимірів стабільний (оскільки будь-яка зміна їхньої структури майже завжди вимагає повної перебудови гіперкуба);

– час відповіді системи на нерегламентовані запити є найбільш важливим параметром.

Типовим прикладом схема реалізації системи MOLAP можна подивитись на зображенні (рис. 2.2). Де видно, що існує багатовимірна система, по якій обирається точка зіткнення для підрахунку агрегаційного запиту.

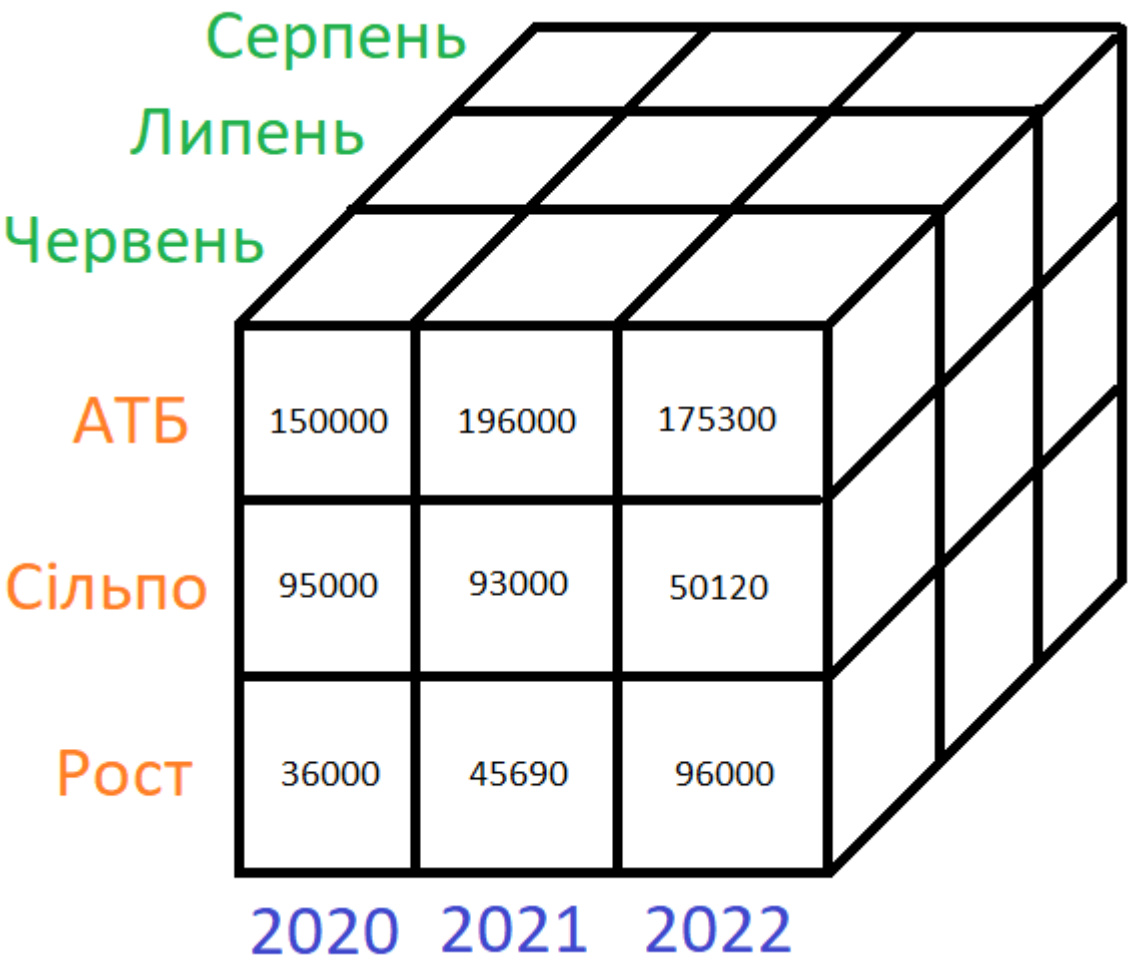

Рисунок 2.2 – Схема реалізації системи MOLAP

До переваг системи MOLAP відносяться інші властивості:

– висока продуктивність. Пошук і вибірка даних здійснюються набагато швидше, ніж у реляційних базах даних;

– структура та інтерфейси найкращим чином відповідають структурі аналітичних запитів;

– багатовимірні СУБД легко справляються з інтеграцією в інформаційну модель різноманітних додаткових функцій.

До недоліків системи MOLAP відносяться такі властивості:

а) MOLAP можуть працювати тільки зі своїми власними багатовимірними БД і ґрунтуються на патентованих ліцензійних рішеннях для

багатовимірних СУБД, що позначається на ціні. Такі технології забезпечують повний цикл OLAP-оброблення і або містять, крім серверного модуля, власний інтегрований клієнтський інтерфейс, або використовують для зв'язку з користувачем зовнішні програми роботи з електронними таблицями;

б) низькі показники ефективності використання зовнішньої пам'яті, гірші, порівняно з реляційними, БД механізми транзакцій;

в) відсутні єдині стандарти на інтерфейс, мови опису та управління даними;

г) не підтримують реплікацію даних, часто використовувану як механізм завантаження.

2.1.2 Реалізація механізму за допомогою ROLAP

ROLAP (реляційна OLAP) – OLAP-системи, які мають прямий доступ до наявних баз даних або використовують дані, вивантажені у власні локальні таблиці.

Аналітичні запити в ROLAP будуються над віртуальним багатовимірним представленням даних, і їх виконання відбувається на рівні РБД, тобто виконуються SQL-запити над реляційною системою. Основними складовими архітектури БД – є таблиця фактів (fact table) і таблиці вимірів (dimension tables). Таблиця фактів – є основною таблицею БД. У ній зазвичай містяться відомості про об'єкти або події, сукупність яких буде піддана аналізу. Таблиці вимірювань містять постійні або рідко змінювані дані. Вони містять щонайменше одне описове поле і цілочисельне ключове поле для однозначної ідентифікації вимірюваної величини. Таблиця вимірювань обов'язково має перебувати у відношенні "один до багатьох" із таблицею фактів. Якщо кожен вимір міститься в одній таблиці вимірів, то таку схему називають "зірка". Якщо ж хоча б один із вимірів міститься в кількох взаємопов'язаних таблицях, то така схема побудови називається "сніжинка".

Якщо багатовимірна модель реалізується у вигляді РБД, необхідно її

представляти як довгі та "вузькі" таблиці фактів і порівняно невеликі та "широкі" таблиці вимірювань.

Таблиці фактів містять числові значення комірок гіперкуба, а решта таблиць визначають багатовимірну сукупність вимірів, що їх містить. Частину інформації можна отримувати за допомогою динамічної агрегації даних, розподілених за нормалізованими структурами, що відрізняються за своєю архітектурою від "зірки". Але в цьому разі запити, що включають агрегацію за високонормалізованої структури БД можуть виконуватися досить повільно. Подання багатовимірної інформації за допомогою зіркоподібних реляційних моделей усуває проблему оптимізації зберігання розріджених матриць, яка гостро стоїть перед багатовимірними СУБД, у яких проблема розрідженості вирішується спеціальним вибором схеми. Хоча для зберігання кожного осередку використовується цілий запис, що включає, крім безпосередньо значень, вторинні ключі – посилання на таблиці вимірів.

ROLAP-системи мають свої переваги та недоліки порівняно з багатовимірними системами. До переваг відносять такі властивості:

а) реляційні СУБД можуть працювати з дуже великими БД і мають розвинені функції адміністрування. При використанні ROLAP розмір сховища не є настільки важливим параметром, як у випадку з MOLAP;

б) у разі оперативного аналітичного опрацювання вмісту сховища даних інструменти ROLAP дають змогу здійснювати аналіз безпосередньо над сховищем, адже зазвичай корпоративні сховища даних реалізуються за допомогою реляційних СУБД;

в) за умови мінливої розмірності задачі, коли зміни в структуру вимірювань вносять доволі часто, ROLAP системи з динамічним представленням розмірності стають найкращим рішенням, адже в них такі маніпуляції не вимагають фізичної реорганізації БД;

г) системи ROLAP можуть функціонувати на набагато менш потужних клієнтських станціях, оскільки основне обчислювальне навантаження припадає на сервер, де виконують складні аналітичні SQL-запити, які формує

система;

ґ) реляційні СУБД забезпечують значно вищий рівень захисту даних і хороші можливості розмежування прав доступу.

До недоліків системи ROLAP відносяться такі властивості:

а) обмежені можливості розрахунку значень функціонального типу;

б) менша продуктивність, ніж у MOLAP. Для забезпечення порівнянної з MOLAP продуктивності реляційні системи вимагають ретельного опрацювання схеми БД і спеціального налаштування індексів. Але в результаті такої роботи продуктивність добре налаштованих реляційних систем при використанні схеми "зірка" можна порівняти з продуктивністю систем на основі багатовимірних БД.

2.1.3 Реалізація механізму за допомогою HOLAP

HOLAP являє собою гібридний OLAP. Він може керувати компромісом між масштабованістю ROLAP і реалізацією запитів MOLAP, деякі комерційні OLAP-сервери залежать від методу HOLAP. У цьому випадку користувач визначає, яку частину даних зберігати в MOLAP, а яку в ROLAP. Наприклад, як правило, низькорівневі дані зберігаються з використанням РБД, тоді як високорівневі дані, включаючи агрегації, зберігаються в незалежній MOLAP.

HOLAP – це суміш ROLAP та MOLAP, які є різними реалізаціями OLAP. HOLAP дозволяє зберігати елементи даних у сховищі MOLAP, а інші елементи даних – у сховищі ROLAP, забезпечуючи компроміс переваг кожного з них. Ступінь контролю, який має розробник куба над цим поділом, змінюється від продукту до продукту.

Оскільки гібридні OLAP дозволяють використовувати декілька наборів з двох OLAP, вони, як правило, зберігають дані як в РБД, так і в багатовимірній БД. В результаті, рішення про доступ до однієї з двох баз даних базується на тому, яка з них найкраще підходить для бажаного типу обробки або програмного забезпечення.

Це підтримує набагато більшу гнучкість при управлінні даними. Для обробки великих обсягів даних дані зберігаються в реляційній базі даних, в той час як для теоретичної обробки дані зберігаються в багатовимірній базі даних. Цей метод корисний в наступних ситуаціях, які полягають в наступному:

а) якщо є потреба у використанні існуючих узагальнених та організованих джерел даних;

б) якщо існують випадки вузьких місць у продуктивності, коли доступ до інформації здійснюється з сервера;

в) якщо є обсяги інформації, з якими не може впоратися одна багатовимірна БД.

Наша задача полягає в тому, щоб можна було зекономити на вартості розмірності серверів і не страждати від повільного отримання необхідної інформації. Тому за основу візьмемо механізм ROLAP, який може бути дуже гнучким в плані оптимізації структури даних і планів запиту.

2.2 Columnstore індекси

Індекси columnstore – це стандарт зберігання і запитування великих обсягів даних у таблицях фактів. При цьому використовується формат зберігання даних у стовпцях і виконується відповідна обробка запитів, що дає змогу практично в 10 разів підвищити продуктивність запитів до сховища даних порівняно з традиційним сховищем, у якому дані зберігаються в рядках. Також, можна домогтися 10-кратного стиснення даних щодо нестиснутих даних [12].

Починаючи з версії SQL Server 2016 (13.x); з пакетом оновлення 1 (SP1), індекси columnstore дозволяють виконувати операційну аналітику – продуктивний аналіз транзакційного робочого навантаження в реальному часі [13].

2.2.1 Вектори Columnstore індексів

Як і звичайні індекси в базах даних, columnstore індекси мають такий же самий принцип зберігання. Тобто якщо це некластерний індекс, то робиться копія даних і зберігається окреми від основної таблиці, а якщо індекс кластерний, то робиться повний ребілдінг сховища даних у таблиці з використанням необхідних для побудови алгоритмів і методів [14].

Взагалі в columnstore індексах використовується метод колонковий формат зберігання, тобто дані зберігаються не у вигляді кучі чи звичайного кластерного індексу, такий формат ще називають Rowstore, а у вигляді окремо створених векторів, кожний з яких зберігає інформацію лише для одної колонки таблиці (рис. 2.3).

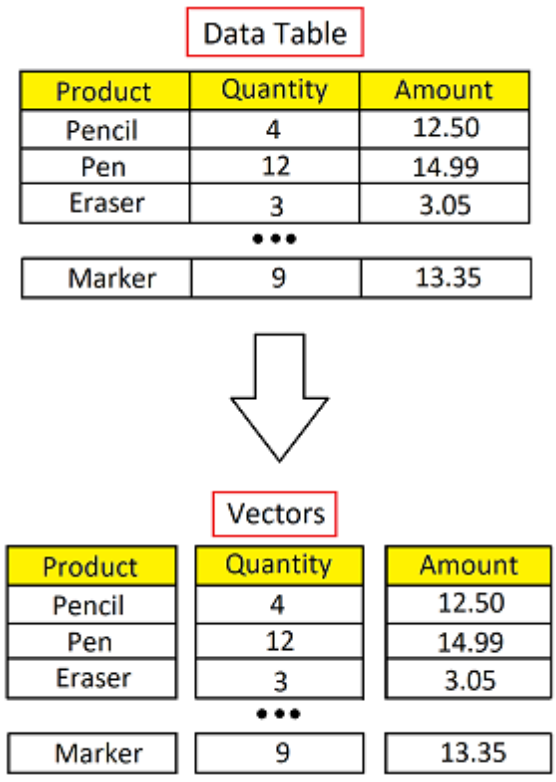

Рисунок 2.3 – Вектори columnstore індексу

Цей підхід до збереження даних дозволяє виконувати швидше запити,

які основані на якихось колонках. Тобто, якщо нам буде необхідно отримати лише всі дані по Product, то для цього нам не потрібно буде зчитувати всі пейджі/сторінки і там шукати необхідні дані, які стосуються лише колонки Product, як це зроблено у rowstore-сховищах, а лише зчитати один вектор і відобразити бажану інформацію користувачу [15].

2.2.2 Визначення терміну та застосування Rowgroups в columnstore індексах

Rowgroup – це група рядків, що стискаються у форматі columnstore одночасно. Rowgroup зазвичай містить максимальне можливе число рядків – 1 048 576 рядків.

Щоб домогтися високої продуктивності і високого рівня стиснення, індекс columnstore розділяє таблиці на групи rowgroup, кожна з яких потім стискається на рівні стовпців. Кількість рядків у групі рядків має бути досить великою, щоб підвищити швидкість стиснення, і досить малою для використання переваг використання операцій у пам'яті [16].

Група рядків, з якої було видалено всі дані, переходить зі стану COMPRESSED у стан TOMBSTONE, після чого вона видаляється фоновим процесом, який називається завданням перенесення кортежів.

Тобто при створенні індексу на таблицю дані розбиваються на групи (rowgroups) по приблизно по мільйону рядків, але ця цифра може бути і меншою, залежно від того, якого типу дані, скільки байт вони займають, унікальність рядків по колонці.

В кожній групі існують всі стовпчики із таблиці. Приклад розподілу таблиці на групи можна подивитись на малюнку нижче (рис. 2.4). Кольорові сегменти у кожній групі – це стовпець, який має однакову кількість рядків відносно інших стовпців однієї групи [17].

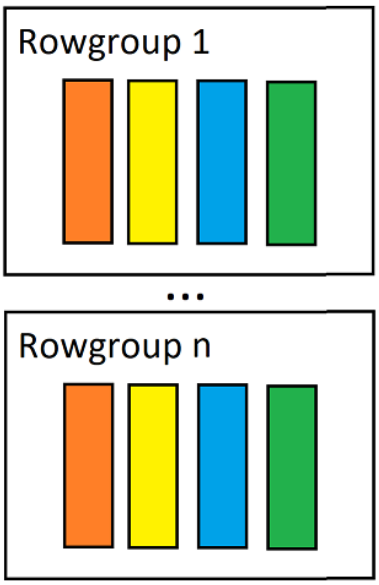

Рисунок 2.4 – Розподіл таблиці на групи (rowgroup)

Ці групи, які зберігають дані у вигляді колонок називають compressed regroup. У таких групах для кожної колонки застосовується кодування векторів для того, щоб вони займали менше міста. Але також в columnstore індексах існує інший тип груп, які називаються Delta store rowgroups. Вони поділяються на Open rowgroup та Closed rowgroup. Ці групи не зберігають дані у вигляді колонок, а у вигляді звичайних rowstore індексів, тобто у вигляді бінарного дерева. Приклад Delta store групи наведений нижче (рис.2.5).

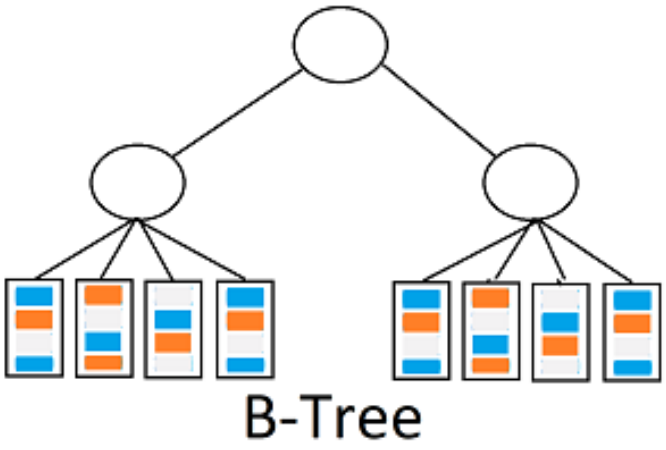

Рисунок 2.5 – Delta store rowgroup

Причиною існування таких груп у columnstore індексах є збереження ресурсів при вставці нових рядків даних до таблиці.

Наприклад, коли при вставці нових даних було додано менше одного мільйона рядків, то оптимізатор СУБД приймає рішення залишити його у вигляді rowgroup для того, щоб не створювати зайву не заповнену скомпресовану групу.

Тобто коли відбувається заливка нових даних, то спочатку вони потрапляють у Delta сховища, які також поділяються на Open та Closed rowgroups.

Якщо кількість рядків у цій групі менше одного мільйона, то ця група відноситься до Open rowgroup, потім коли відбувається нова заливка, то спочатку заповнюються відкриті групи, які мають рядків менше мільйона, щоб досягти максимальної кількості рядків для групи. Після чого група починає відноситись до Closed rowgroup.

Closed групи мають максимальну кількість рядків та в них не можна більше заливати нові дані.

Це означає, що вони вже готові до наступного етапу оптимізації – це компресії даних та перетворення архітектури збереження даних на стовпчиковий [18].

Після того, як група почала відноситись до compressed rowgroup, відбувається перенесення Closed групи до групи Tombstone, яка потім видаляється з часом автоматично чи можна зробити це власноруч. Цикл життя груп можна подивитись на малюнку нижче (рис. 2.6).

Кожен раз, коли заповнюється відкрита та закрита роугрупи, то відбувається перенос роугрупи до наступного етапу – TOMBESTONE. Ця роугрупа не видна користувачам БД, до неї нема доступу. При роботі з таблицями вектори з цієї роугрпи не зчитуються та ніяк не впливають на інші, відкриті та вже закомпресовані роугрупи [19].

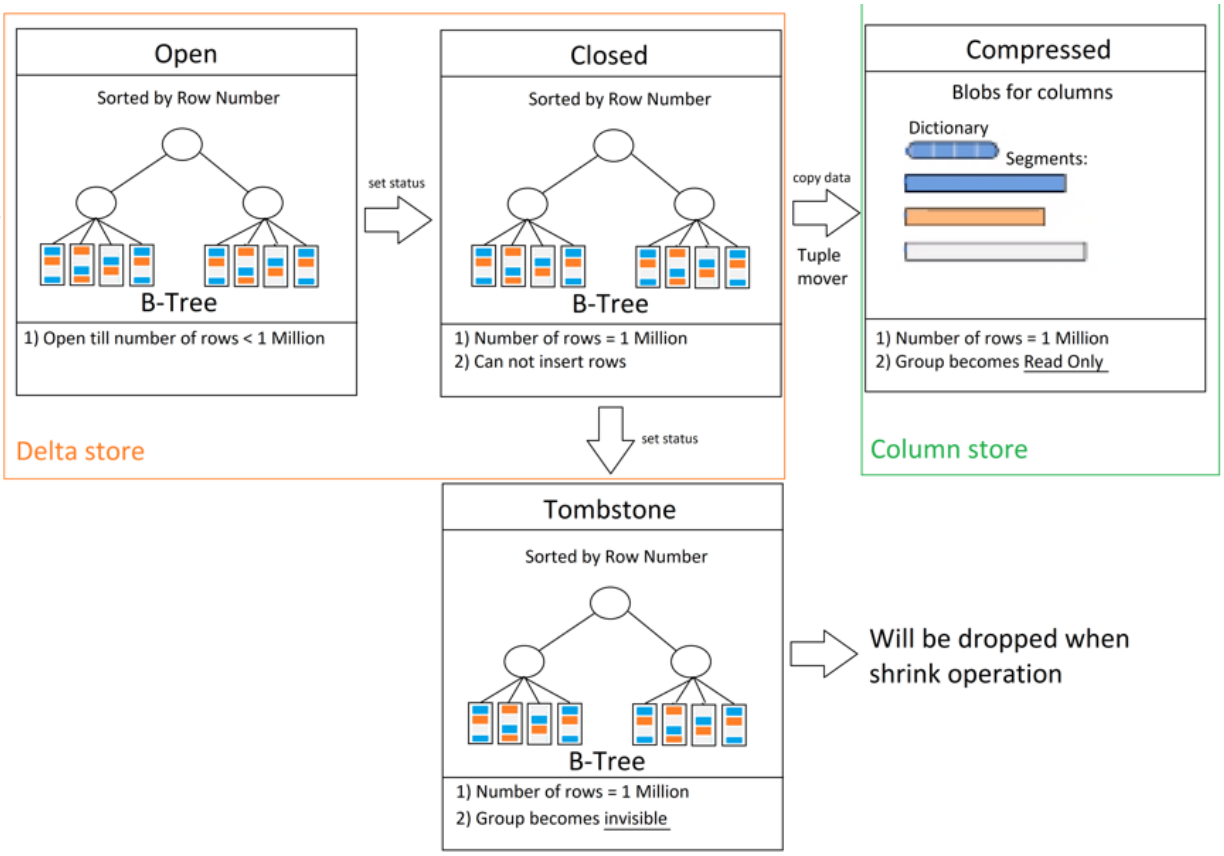

Рисунок 2.6 – Цикл життя груп columnstore індексу

2.3 Висновки до другого розділу

Було розглянуто OLAP-системи, які пристосовані виключно для аналітичних задач. В таких системах маніпуляція з даними дуже обмежена: не можливо записати нову інформації, оновити старі дані чи видалити. Існує лише можливість зчитування, тому ця система повністю налаштована на те, щоб швидко зчитувати необхідну інформацію для аналітики.

Було виявлено 3 механізми OLAP: ROLAP, MOLAP, HOLAP. Визначено найкращу ситуацію для кожної реалізації механізму. Якщо використовувати якусь із цих реалізацій механізму, то при умові, що потрібно найвища швидкодія отримання даних і немає обмежень на зберігання даних, то підійде MOLAP. Якщо ж питання ваги БД є важливим і можна поступитися швидкодією, то ROLAP буде найкращим вибором. Реалізація механізму

HOLAP підійде для середнього варіанту.

Були розглянуті основи архітектури Columnstore індексів. Виявлено, що векторизація даних – дуже зручний метод зберігання даних в випадку, якщо рядків у таблиці дуже багато.

3 МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ, СТВОРЕННЯ АЛГОРИТМІВ ДЛЯ ПРИШВИДШЕННЯ ВИКОНАННЯ ЗАПИТІВ ТА РОЗРОБКИ НОВОЇ КОМПРЕСІЇ ДАНИХ В АБД

3.1 Математична модель опису алгоритмів компресії

Основною перевагою columnstore індексів є саме компресія даних (КД) за допомогою алгоритмів кодування даних. Кожен з цих алгоритмів обирається відносно того, який тип даних у стовпчика [20]. Всього існує два основних методу кодування, які залежать від типу даних:

а) кодування значення (value base encoding);

б) словникове кодування (dictionary encoding).

Кодування значення працює тільки з цифровими типами даних, бо метод полягає в математичних перетвореннях даних. Алгоритм кодування можна побачити на малюнку нижче (рис. 3.1).

Base Value Encoding (for integer)

Value	Binary Value	Size (bits)	Value * 10^{-2}	Value * 10^{-2} – 5	Binary Value	Size (bits)
1700	110 1010 0100	11	17	12	1100	4
289000	100 0110 1000 1110 1000	19	2890	2885	1011 0100 0101	12
500	1 1111 0100	9	5	0	0	0
10000	10 0111 0001 0000	14	100	95	101 1111	7
1000	11 1110 1000	10	10	5	0101	3
2000000	1 1110 1000 0100 1000 0000	21	20000	19995	100 1110 0001 1011	15
Total:		84				41
Original			Calculation		Encoded	

Рисунок 3.1 – Кодування значення (integer) для компресії у columnstore індексах

Для початку у колонці із групи аналізується можливість ділення всіх

значень на якесь число, щоб всі числа стали меншими, але цілими. Коефіцієнт на який ділять всі значення вектору називається мантісою. Це значення обирається кожен раз по різному. Основною її задачею є ділення всіх значень на 10 у якійсь степені так, щоб всі значення із вектору залишились цілими. Після того, як отримали закодовані за допомогою ділення всіх значень на мантісу, обирається найменше число і віднімають це значення для кожного рядку. Такий коефіцієнт називається базою. До кожного значення з вектору застосовується наступна математична формула для находження частково закодованого значення з використанням мантіси:

$$Em = x * 10^k, \tag{3.1}$$

де k – мантіса;

x – не закодоване значення.

Далі необхідно знайти значення бази за наступною формулою:

$$b = X_{min}, \tag{3.2}$$

де X_{min} – мінімальне значення з вектору.

Наступним кроком іде застосування бази до частково закодованого значення за наступною формулою:

$$E = Em - b, \tag{3.3}$$

Результат являє собою повністю закодований вектор, який став менше важити, ніж від первісного вектору. Це кодування в теорії може бути використане і для не цифрових значень, тобто спочатку використати Hash-кодування, яке буду переварювати дані значення у цифровий вигляд, а вже потім використати Value-кодування для ще більшого зменшення значення.

Також існує такий же метод але вже для цифрових значень з плаваючою комою (рис.3.2).

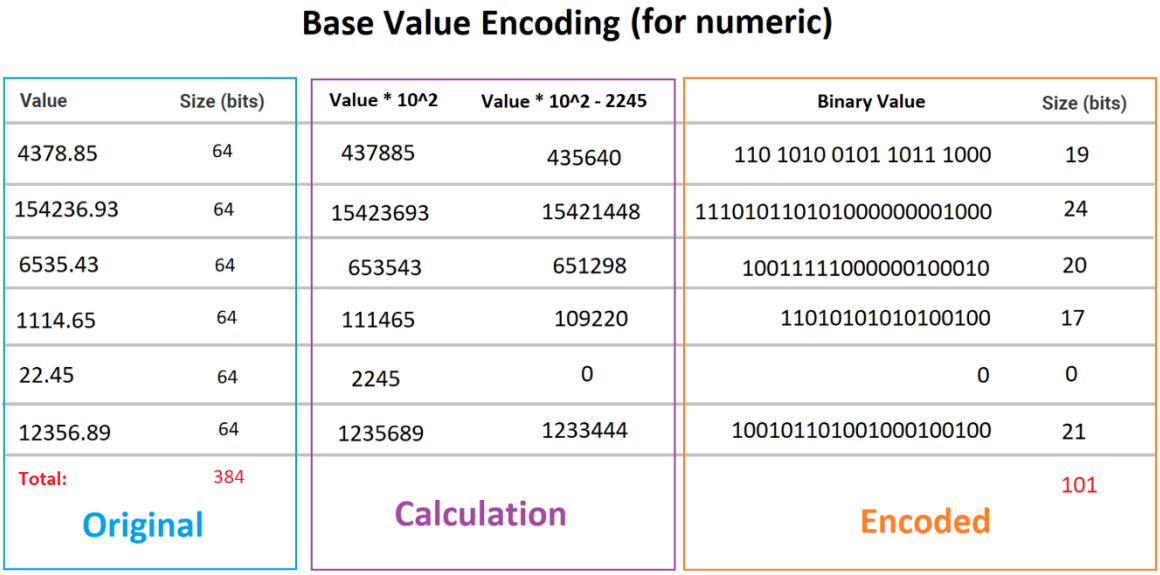

Рисунок 3.2 – Кодування значення (numeric) для компресії у columnstore індексах

Також ефективним кодуванням є хеш-кодування, яке використовує хеш-функції для перетворення одного більшого значення на більше менше в даному випадку (рис. 3.3).

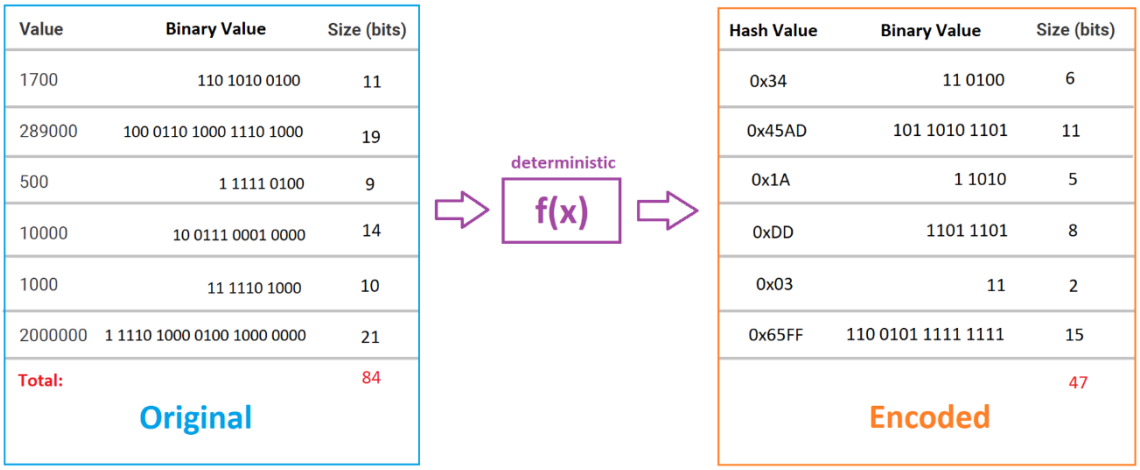

Рисунок 3.3 – Хеш-кодування у columnstore індексах

Таке кодування може використовувати різні функції для перетворення,

але основна мета цих функцій – зменшити розміри значень у векторі. Зрозуміти до кінця що коїться всередині функції дуже складно, але точно можна сказати, що відбувається аналіз даних вектору:

а) яка середня розмірність значень;

б) скільки дублікатів;

в) яке максимальне по розмірам значення;

г) яке мінімальне по розмірам значення.

Це все робиться для того, щоб обрати самий оптимальний алгоритм перетворення значень. Щоб мінімізувати випадки створення одного і того же значення при різних вхідних даних. Якщо таке відбувається, то потребується виділення додаткових ресурсів на обробки даної проблеми.

Також слід зазначити що Hash-функція має властивість детерміністичності, а це означає, що функція буде виконувати один і той же алгоритм для кожного вхідного значення.

Взагалі існують два основні методи Hash-кодування:

а) Open Addression Method (OAM);

б) Chain Method (CM).

Кожен з цих методів був створений для того, щоб боротись з проблемою створення одного і того ж hash-значення для різних вхідних даних, але кожен це робить по своєму.

У OAM-методі для кожного з вхідних значень повинно бути унікальне закодоване значення (рис.3.4).

Для досягнення такого результату використовується динамічна формула:

$$\mathrm{H} = f(x) + i * f_2(x), \tag{3.4}$$

де $f(x)$ – перша hash-функція;

$f_2(x)$ – друга hash-функція;

i – динамічне значення, яке залежить від ітерації.

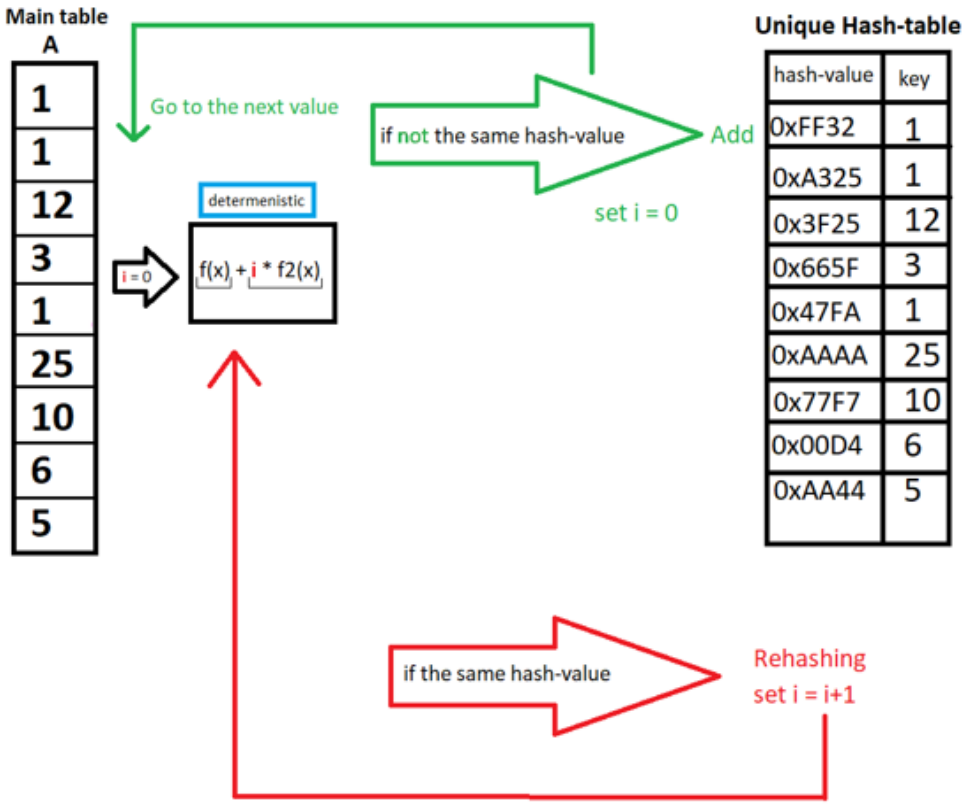

Рисунок 3.4 – Open Addression метод для Hash-кодування

Суть полягає в тому, щоб брати по черзі значення з вектору, надсилати їх до комплексу функцій з формули (3.4) для початку зі значенням i = 0 і якщо таке значення ще нема у списку, то тоді вона додається у вигляді хеш-значення та ключа яке являє собою оригінал.

Якщо таке значення вже є у списку, то тоді значення «i» збільшується на одиницю і оригінал знову подається до комплексу функцій з формули (3.4), але вже з новим значенням i = 1. Такі дії продовжують до поки хеш-значення зі списку не будуть унікальними.

В цьому методі є свої переваги і недоліки. До переваг відноситься повне вирішення проблеми з однаковими хеш-значеннями для різних вхідних даних, а до недоліків – неекономність цього методу, бо перевірка проходить методом перебору всіх елементів списку. Тому це може бути дуже довгий процес.

У СМ-методі може бути однакове хеш-значення для різних вхідних

даних, бо тут застосована ланцюгова методика для зберігання однакових значень (рис. 3.5).

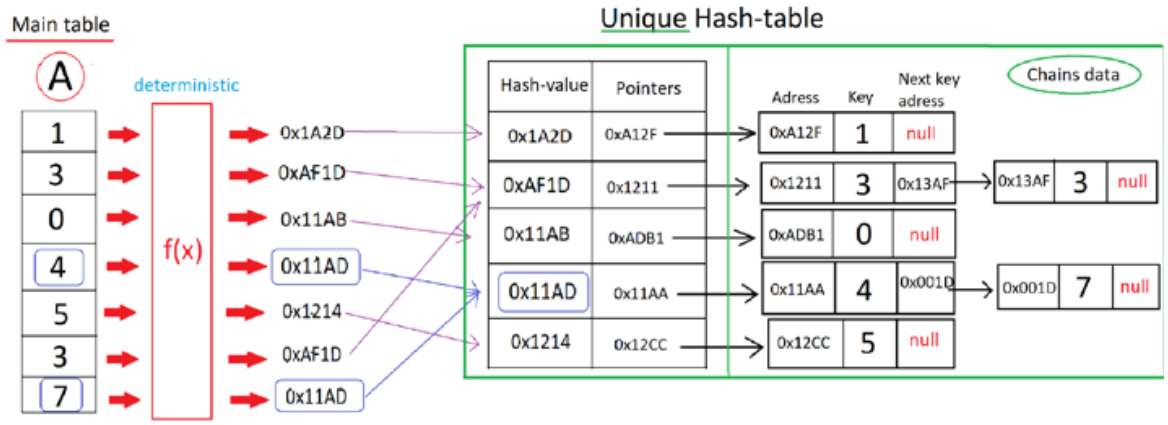

Рисунок 3.5 – Chain метод для Hash-кодування

Суть полягає у тому, щоб теж брати по черзі вхідні дані з вектору для подальшого занесення у звичайну хеш-функцію, і потім йде перевірка хеш-значення: чи існує таке вже у списку.

Якщо такого значення ще нема, то додається до списку, а також вказівник до початку ланцюгу, в якому буде міститись значення оригіналу та вказівник на наступне значення з таким же хеш-значенням, якщо наступного нема, то вказується NULL.

Якщо таке хеш-значення є, то оригінал додається в кінець ланцюгу зі своїм вказівником та вказівником на наступне значення, якщо воно є.

3.2 Розгляд існуючих алгоритмів компресії у Columnstore індексах

Другим основним кодуванням у columnstore індексах є словникове кодування для значень з типом даних VARCHAR, CHAR та інші типи даних (рис. 3.6).

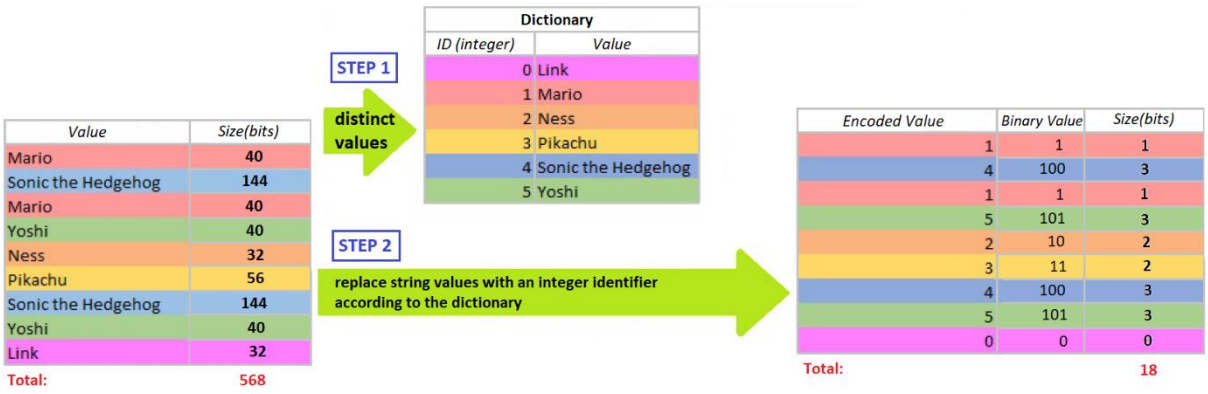

Рисунок 3.6 – Словникове кодування у columnstore індексах

Дуже простий метод для розуміння, який полягає в тому, щоб створити словник, базуючись на даних вектору, який має всі унікальні значення із цього стовпчика, у вигляді ID та самого значення [21]. Після чого сегмент кодується, записуючі замість самого значення ID тим самим зменшуючи вагу у рази. Але треба мати на увазі, що цей метод буде ефективний при наявності маленької кількості унікальних значень стовпця вектору групи (рис. 3.7).

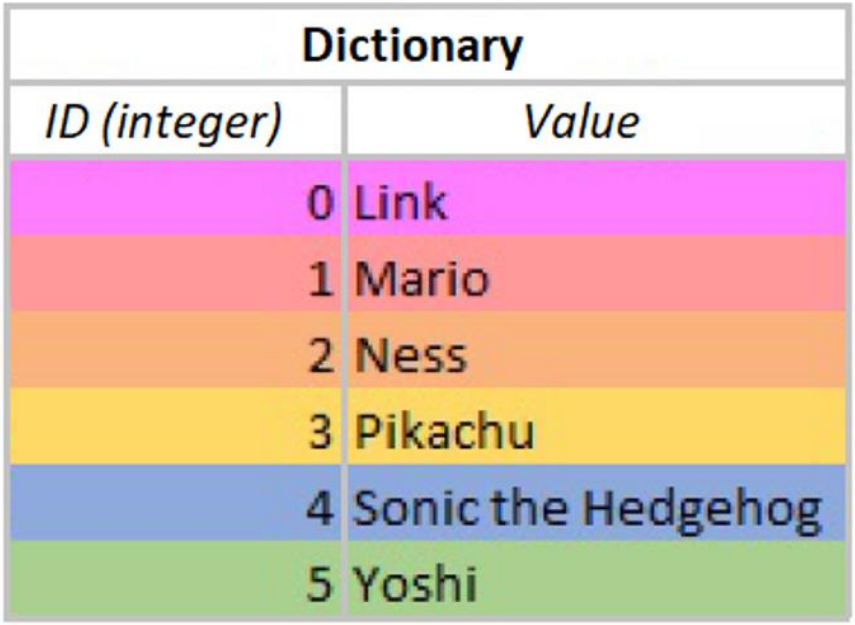

Рисунок 3.7 – Словник кодування

Словники в кодуваннях columnstore індексах є дуже ефективні та прості у розумінні та у використанні алгоритмів. Їх суть полягає у тому, що на вхід подається по порядку значення з вектору, перевіряються: чи є вже це значення у словнику. Якщо його нема, то це значення записується у кінець словника та надається йому номер ID, які потім будуть використовуватися замість оригінальних значень у векторах. Якщо це значення вже є у словнику, то нічого не додається у словник і береться наступне значення з вектору. І таким чином буде відбуватись до поки не закінчяться значення у векторі (рис. 3.8).

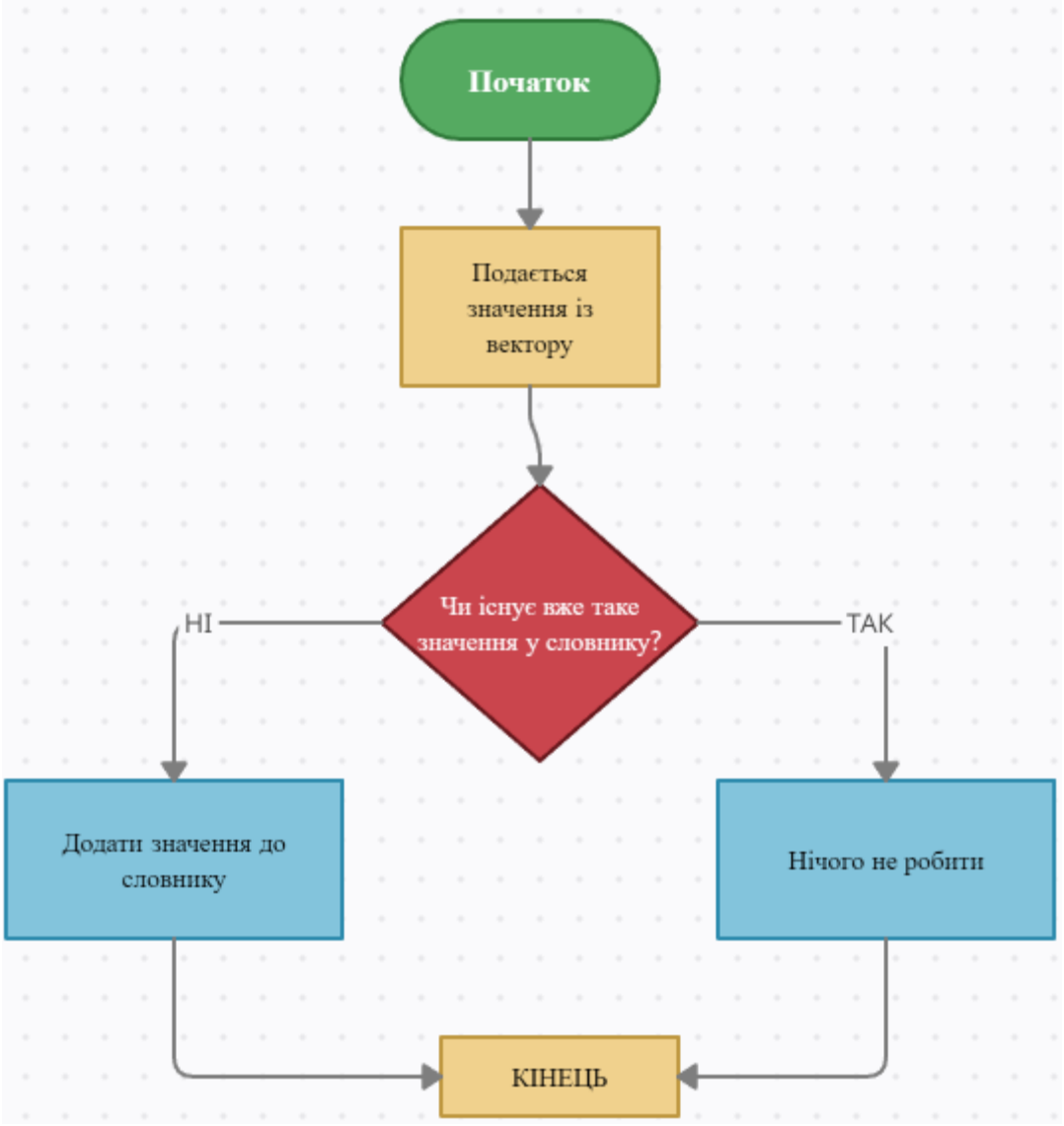

Рисунок 3.8 – Алгоритм заповнення словника кодування

Таким чином з словниковим кодування отримується дуже хороша КД, на зображенні наглядно видно, що компресія зменшила загальну вагу вектору у 30 разів, але також необхідно враховувати, що зберігаються не лише закодовані значення у векторах, а ще й сам словник. Словник може існувати лише один для одного вектору.

Наступний алгоритм є так званим загальним алгоритмом, який може бути задіяним для всіх векторів і не має вимог щодо типу даних. Таким кодуванням є RLE-кодування.

Уявимо, що ми маємо таку таблицю з вже закодованими значеннями за допомогою словника (рис. 3.9) чи іншими типами кодування, які залежать від типів даних.

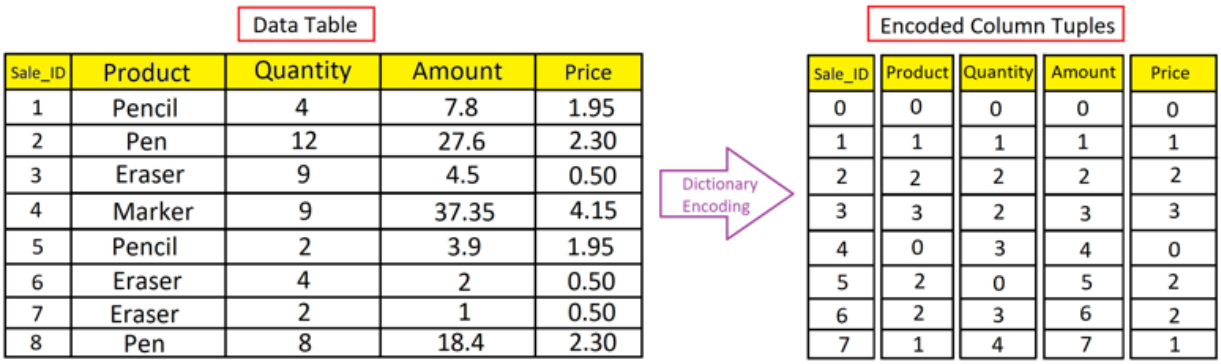

Рисунок 3.9 – RLE-кодування: приклад таблиці з закодованим прикладом

У першу чергу необхідно зчитати статистику по колонкам з таблиці для того, щоб просортувати вектори по кількості дублікатів. Тобто необхідно визначити у якого з векторів найбільше всього дублікатів. Це буде перший в черзі по сортуванню вектор (рис. 3.10). Наступні вектои вже будуть сортуватись у рамках сортування попереднього вектору. І так до самого останнього вектору.

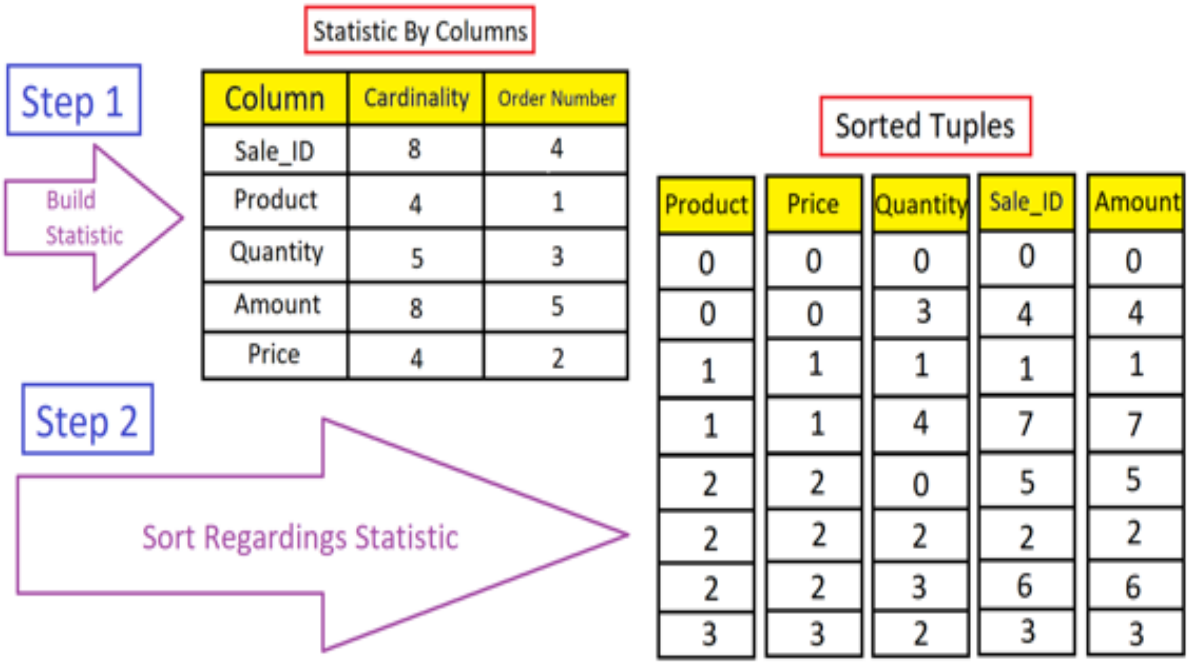

Рисунок 3.10 – RLE-кодування: сортування за допомогою статистики

На рисунку (рис.3.10) можна побачити, що перша колонка – це та колонка, для якої значення Cardinality найменше. Це значення дорівнює кількості унікальних значень у векторі.

Просортувавши перший вектор, йде сортування наступного вектору, але вже у рамках однакових значень першого вектору. Таким чином, вектори, які були просортовані перші впливають на сортування векторів, які йдуть наступними.

Після того, як всі вектори були просортовані, відбувається просте перетворення одного вектору у два, якщо це можливо.

Суть полягає у тому, що беруться однакові значення вектору, які йдуть один за одним, послідовно та записуються одним значенням, а в іншому векторі записується кількість таких значень (рис. 3.11).

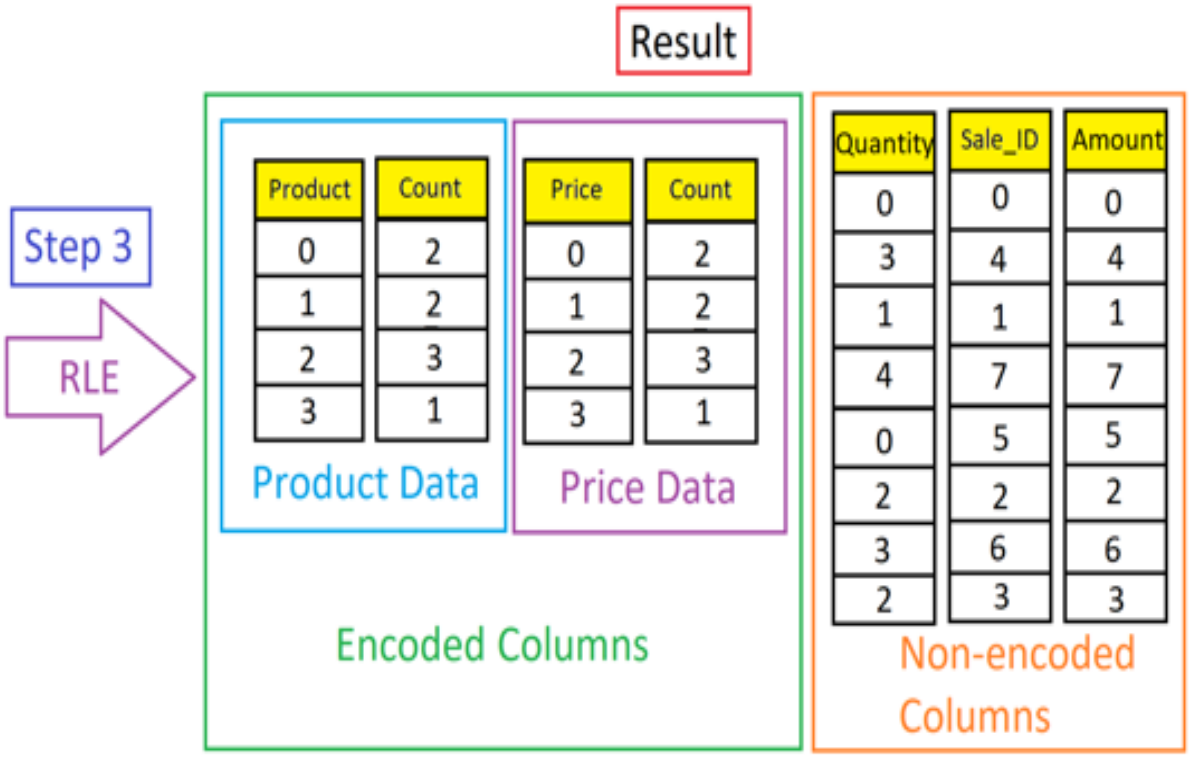

Рисунок 3.11 – RLE-кодування: результат

Отже, можна побачити, що в даному випадку були закодовані лише 2 вектори, бо вони мали однакові значення, які йшли послідовно, інші не мали, тому залишились у незмінному стані.

Другий вектор не створюється для незакодованих випадків, бо в цьому нема ніякого сенсу.

Можна зробити висновок, що це кодування дуже ефективне, якщо вектори мають дуже багато дублікатів і навпаки, якщо дублікатів зовсім нема, то це кодування не буде задіяно, бо результат буде не змінний.

Існує також алгоритм кодування, який розробив Хафман, називається цей алгоритм – словникове кодування Хафмана.

Суть полягає у наступному: уявимо що в нас є словник по якомусь вектору (рис. 3.12).

Dictionary		
ID (integer)	*Value*	*Size(bits)*
0	Link	*32*
1	Mario	*40*
2	Ness	*32*
3	Pikachu	*56*
4	Sonic the Hedgehog	*144*
5	Yoshi	*40*
Total:		**344**

Рисунок 3.12 – Кодування Хафмана: словник

Базуючись на даних словника створюється унікальна табличка, яка містить унікальний символ з словника та кількість таких символів. Туди записуються абсолютно всі символи, які були використані у словнику, навіть коми та дужки (рис. 3.13). Зазвичай, цей метод кодування використовується майже завжди, коли звичайний словниковий метод не підходить по різним причинам.

Також кодування Хафмана дає змогу використовувати його для інших методів кодування задля покращення алгоритмів компресії даних.

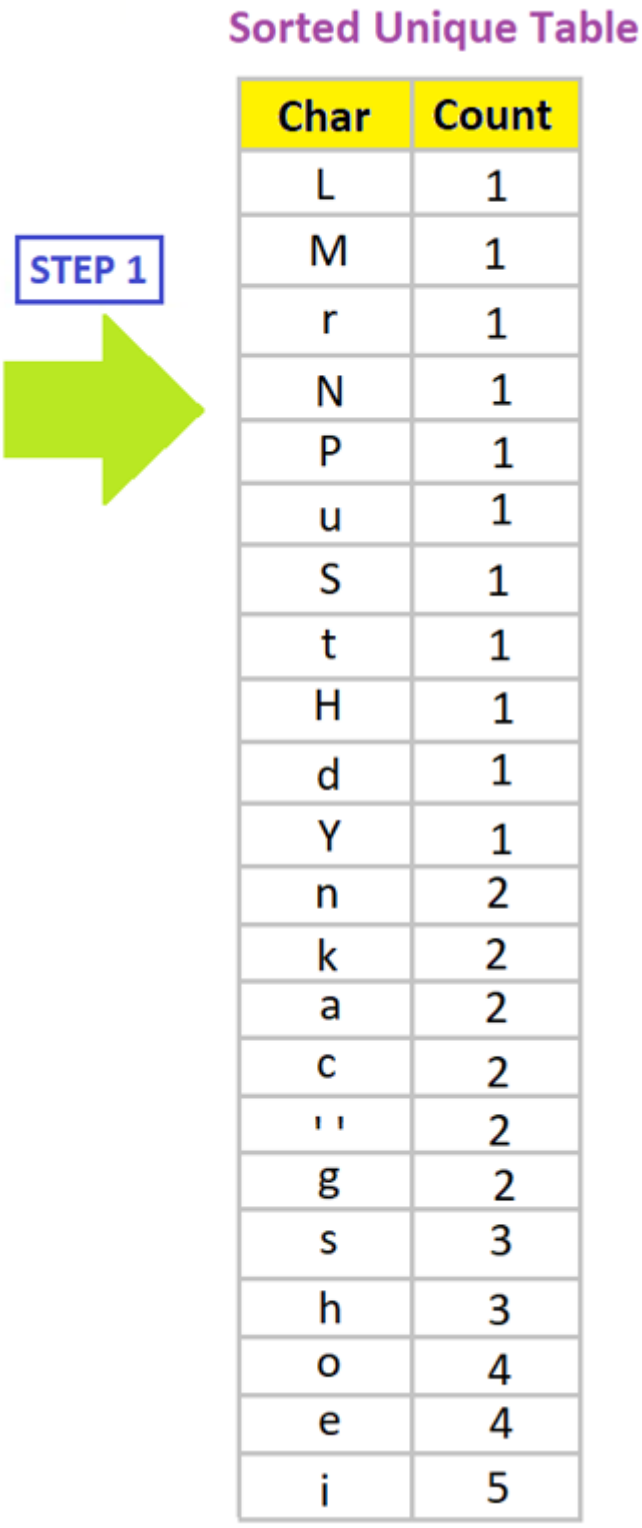

Рисунок 3.13 – Кодування Хафмана: унікальна таблиця

Кількість однакових символів має значення, так як чим більше однакових символів тим менше повинно бути закодоване значення. Алгоритм

кодування полягає у створенні бінарного дерева, де найвища гілка – це символ, який повторюється найчастіше у словнику (рис. 3.14).

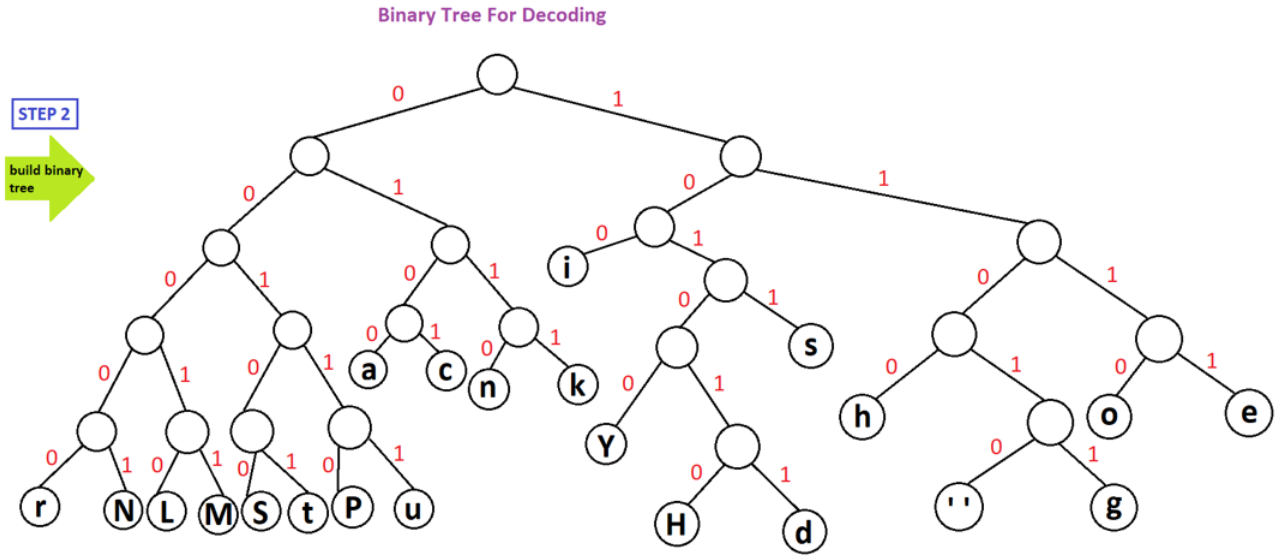

Рисунок 3.14 – Кодування Хафмана: бінарне дерево

Отже суть полягає в тому, що якщо гілка йде вправо, то записується одиниця, якщо вліво – нуль. В такому випадку, наприклад, символ «S» буде дорівнювати набору нулиів та одиниць 1011.

Символ, який зустрічайться частише у словнику, повинен розташовуватися вище всіх. Якщо існують символи, які мають однакову кількість дублікатів, то в дереві Хафмана буде відбуватися змога поставити їх на один рівень.

Повний список закодованих символів можна побачит на рисунку (рис. 3.15).

Coding Table

Char	Count	Encoded Value
L	1	00010
M	1	00011
r	1	00000
N	1	00001
P	1	00110
u	1	00111
S	1	00100
t	1	00101
H	1	101010
d	1	101011
Y	1	10100
n	2	0110
k	2	0111
a	2	0100
c	2	0101
' '	2	11010
g	2	11011
s	3	1011
h	3	1100
o	4	1110
e	4	1111
i	5	100

Рисунок 3.15 – Кодування Хафмана: таблиця закодованих символів

Після створення таблиці закодованих символів починається процес заміни символів у словнику на закодовані значення (рис.3.16).

Dictionary		
ID (integer)	*Encoded Value*	*Size(bits)*
0	00010 100 0110 0111	16
1	00011 0100 00000 100 1110	21
2	00001 1111 1011 1011	17
3	00110 100 0111 0100 0101 1100 0011	28
4	00100 1110 0110 100 0101 11010 00101 1100 1111 11010 101010 1111 101011 11011 1111 1100 1110 11011	81
5	10100 1110 1011 1100 100	20
Total:		183

Рисунок 3.16 – Кодування Хафмана: закодований словник

Можна побачити з рисунку (рис. 3.16), що словник був зменшений у розмірах майже в два рази (було 344 біта, а стало 183). Але треба зазначити, що також зберігаються дерево та таблиця з рисунку (рис. 3.15) для подальшого декодування цих значень при запиті системи до цього вектору.

Фінальним штрихом в кодуванні векторів завжди йде Bit Packing. Це такий алгоритм, який зводить всі закодовані значення у вектори до бітового формату (нулів та одиниць). Але в колумнстор індексах йде не звичайне переведення значень у біти, а ще є мета і тут зекономити місто на серверах. Уявимо, що в нас є вже закодований вектор (рис. 3.17).

Value	Binary	Number Of Bits	Prefix For Value Length
65006	00000000000000001111110111101110	15	01111
161	00000000000000000000000010100001	7	00111
61231	00000000000000001110111100101111	15	01111
44241422	00000010101000110001001000001110	25	11001
4423	00000000000000000001000101000111	12	00001
241	00000000000000000000000011110001	7	00111
2311	00000000000000000000100100000111	11	00011
5111	00000000000000000001001111110111	12	00001
931	00000000000000000000001110100011	9	01001
1031	00000000000000000000010000000111	10	01010
103	00000000000000000000000001100111	6	00110
62042	00000000000000001111001001011010	15	01111
4324222	00000000010000011111101101111110	22	10110
123411576	00000111010110110001110001111000	26	11010
214704342	00001100110011000010000011010110	27	11011
Total:	480 bits	219	

Рисунок 3.17 – Bit Packing: знаходження кількості залишкових бінарних значень

Дані зберігаються у 32-бітному форматі, тобто яке б велике, або мале значення не було у закодованому векторі, якщо це значення не перебільшує 4294967296. Але для економії місця, так як любе значення буде мати 32 бінарних символи, то всі нулики до першої одиниці можна видалити, а також першу одиницю після цих нуликів. Після чого записати кількість залишку бітованих значень, а також закодувати цю кількість теж в бітові значення для того, щоб зчитування системою було можливим (рис. 3.18).

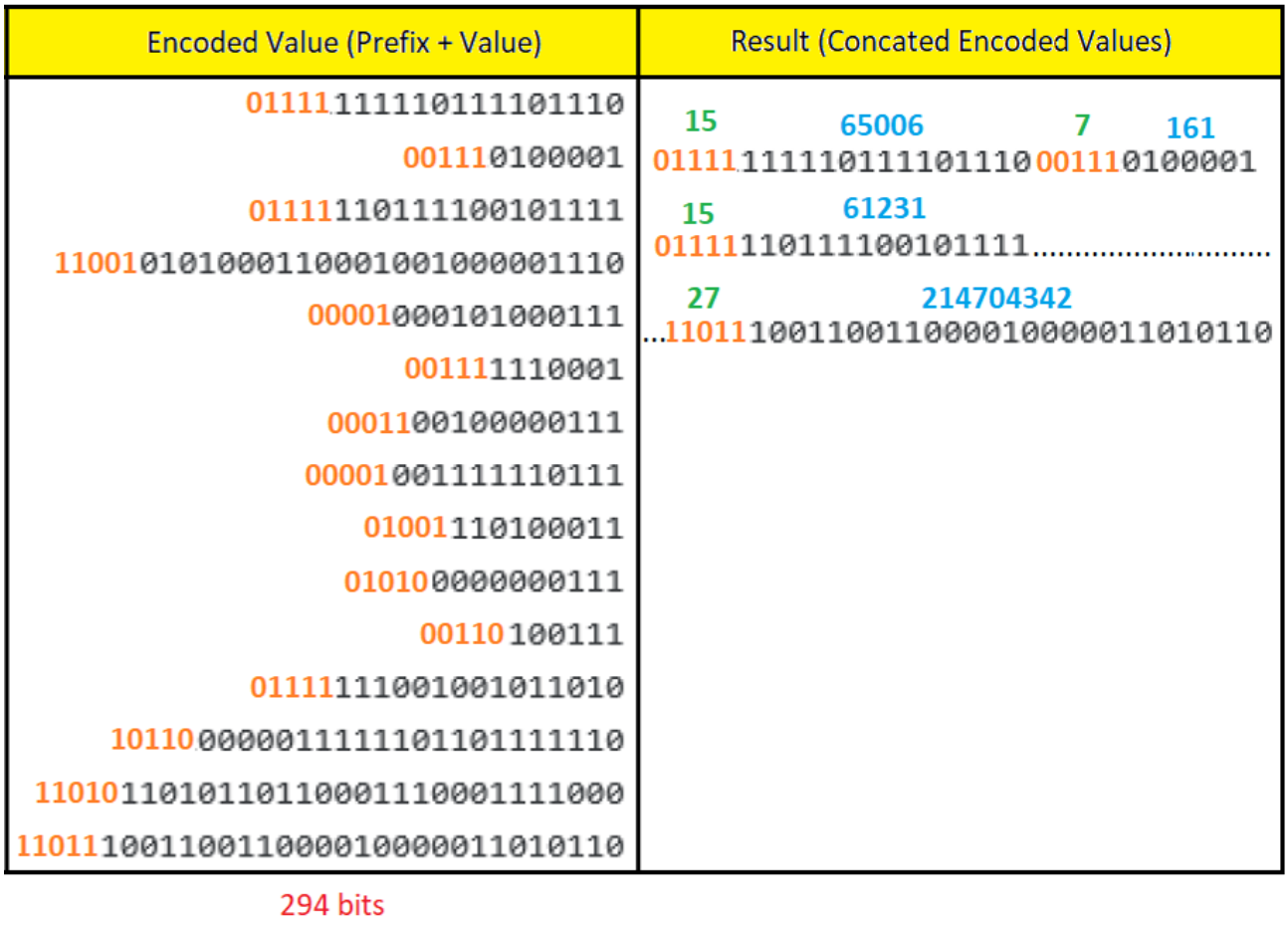

Рисунок 3.18 – Bit Packing: закодований бінарний результат

Отже можна побачити (рис. 3.18), що замість того, щоб зберігати велику кількість нуликів, можна замінити закодованим фіксованим значенням (5 біт), бо максимальна кількість залишкових бітових значень може бути 31, бо ми також видаляємо першу одиницю у значенні. Тому результат виглядає, як

стрічка, яка складає собою початок та саме залишкове бінарне значення. Початок – це кількість бітів, які потрібно зчитати для отримання одного значення, а залишкове бінарне значення – це отримане бінарне значення при видаленні всіх нулів, які йдуть зліва та крайньої одиниці.

3.3 Розробка алгоритму для компресії даних в АБД

Для такої компресії було застосовані методи з окремих кодувань. Суть кодування полягає в тому, щоб закодувати цифрове значення, яке погано кодується кодуванням значення (рис. 2.1). Тобто коли майже нема нулів в кінці чи база не сильно зменшує загальні значення.

Отже, уявимо, що в нас є вектор, який зберігає цифрові значення (рис. 3.19).

Data Presentation

Value	Binary	Hexidecimal + delimiter
65006	0000000000000001111110111101110	7DEE,
161	0000000000000000000000010100001	21,
61231	0000000000000001110111100101111	6F2F,
44241422	0000001010100011000100100001110	A3120E,
4423	0000000000000000001000101000111	147,
241	0000000000000000000000011110001	71,
2311	0000000000000000001001000000111	107,
5111	0000000000000000001001111110111	3F7,
123411576	0000011101011011000111100011111000	35B1C78,
214704342	0000110011001100001000001101011O	4CC20D6
Total:	480 bits /167 bits	

Рисунок 3.19 – Цифрове кодування: презентація даних

Перше, що потрібно зробити, це застосувати метод з кодування Bit Packing, тобто перевести все до бінарного вигляду, а потім скоротити ведучі нулі та першу одиницю. Після чого необхідно перевести залишки бінарного значення у шістнадцятирічну систему. Також, як і у Bit Packing необхідно, щоб був якийсь розділ між залишковими бінарними значеннями і тут використовується не кількість залишкових бінарних значень, а звичайна кома, яка потім закодується деревом Хафмана (рис. 3.20).

Sorted Unique Table

Char	Count
,	9
1	6
7	5
2	4
E	3
F	3
3	3
0	3
C	3
D	2
6	2
4	2
A	1
5	1
B	1
8	1

Рисунок 3.20 – Цифрове кодування: побудова унікальної таблиці

Зазвичай алгоритм Хафмана використовується для кодування словників, але тут було обране рішення застосувати цей алгоритм для того, щоб зменшити цифрові значення, які були обрізані за допомогою Bit Packing і яке використовує кому для розділу між значеннями (рис. 3.21).

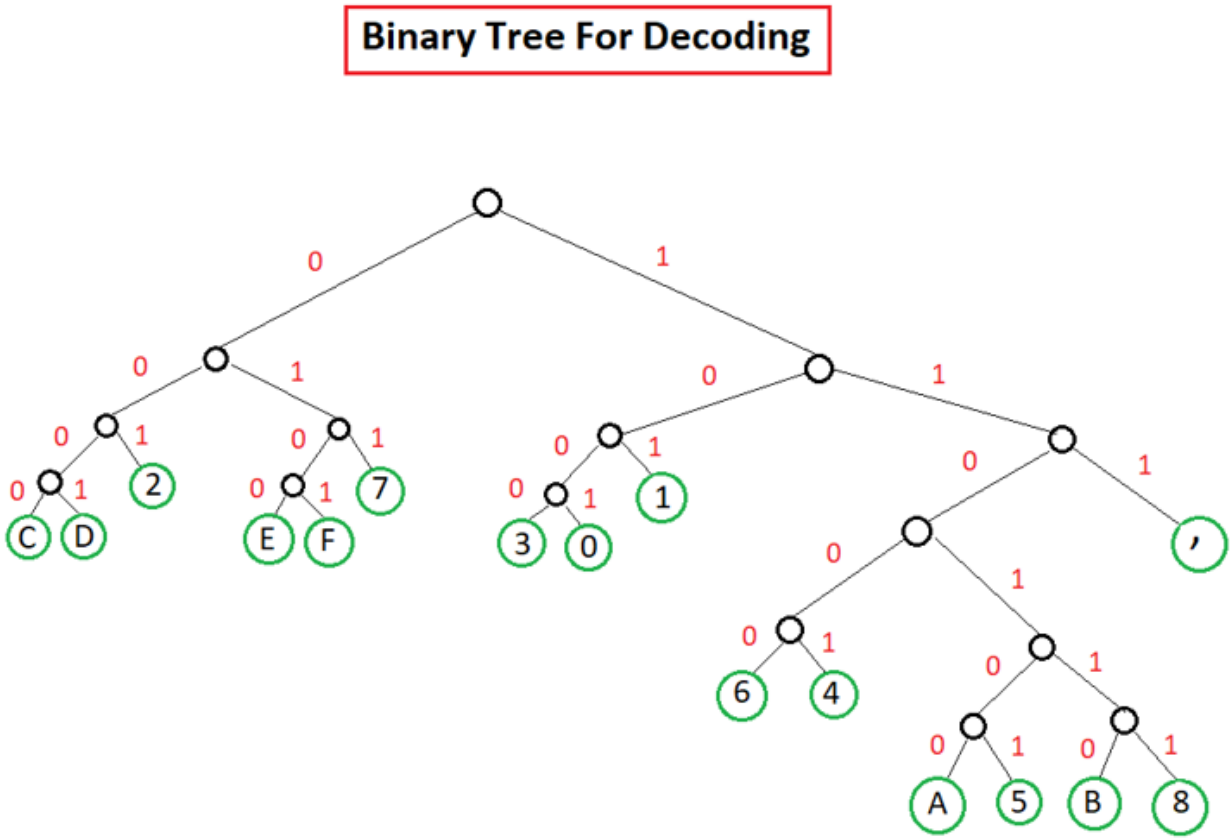

Рисунок 3.21 – Цифрове кодування: дерево Хафмана

Дерево будується таким же чином, як і в звичайному алгоритмі Хафмана. Після чого будується таблиця закодованих значень (рис. 3.22).

Coding Table

Char	Encoded Value
,	111
1	101
7	011
2	001
E	0100
F	0101
3	1000
0	1001
C	0000
D	0001
6	11000
4	11001
A	110100
5	110101
B	110110
8	110111

Рисунок 3.22 – Цифрове кодування: таблиця закодованих значень

Маючи цю таблицю (рис. 3.22), треба підставити закодовані значення замість векторних значень, щоб отримати зменшену версію таких же даних (рис. 3.23).

Encoded Result

Value	Hexidecimal + delimiter	Encoded Value	Size (bits)
65006	7DEE,	011 0001 0100 0100 111	18
161	21,	001 101 111	9
61231	6F2F,	11000 0101 001 0101 111	19
44241422	A3120E,	110100 1000 101 001 1001 0100 111	27
4423	147,	101 11001 011 111	14
241	71,	011 101 111	9
2311	107,	101 1001 011 111	13
5111	3F7,	1000 0101 011 111	14
123411576	35B1C78,	1000 110101 110110 101 0000 011 110111 111	35
214704342	4CC20D6	11001 0000 0000 001 1001 0001 11000	29
Total:			187

Рисунок 3.23 – Цифрове кодування: підставлення закодованих значень

Після того як підставили закодовані значення, можна побачити, що компресія виявилось вдалою, бо результатом компресії є зменшення загальної ваги у бітах майже в 3 рази. А кінцевий результат є ланцюг, який конкатинує всі закодовані значення разом з роздільником.

Result (concatenated encoded values)

7DEE , 21 ,
011 0001 0100 0100 111 001 101 111.....

4CC20D6
....11001 0000 0000 001 1001 000111000

Рисунок 2.23 – Цифрове кодування: результат

3.4 Розробка алгоритму для пришвидшення виконання запитів з агрегацією

Одне з найважливіших задач у аналітиці – це швидка обробка даних.

Час – це дуже важливий ресурс в аналітиці, тим паче, коли актуальність аналітики у реальному часі є пріоритетом.

Для того, щоб зрозуміти алгоритм пришвидшення агрегуючих запитів, необхідно спочатку зрозуміти сам процес матеріалізації колумнстор індексів, якщо вони були застосовані до таблиць.

Отже, уявимо, що в нас є таблиця з трьома колонками (Product, Quantity, Amount), на яку було застосовано колумнстор індекс. Ця таблиця була повністю скомпресована. При запиті на цю таблицю на вході ми будемо мати вектори у бінарному вигляді (рис. 3.24).

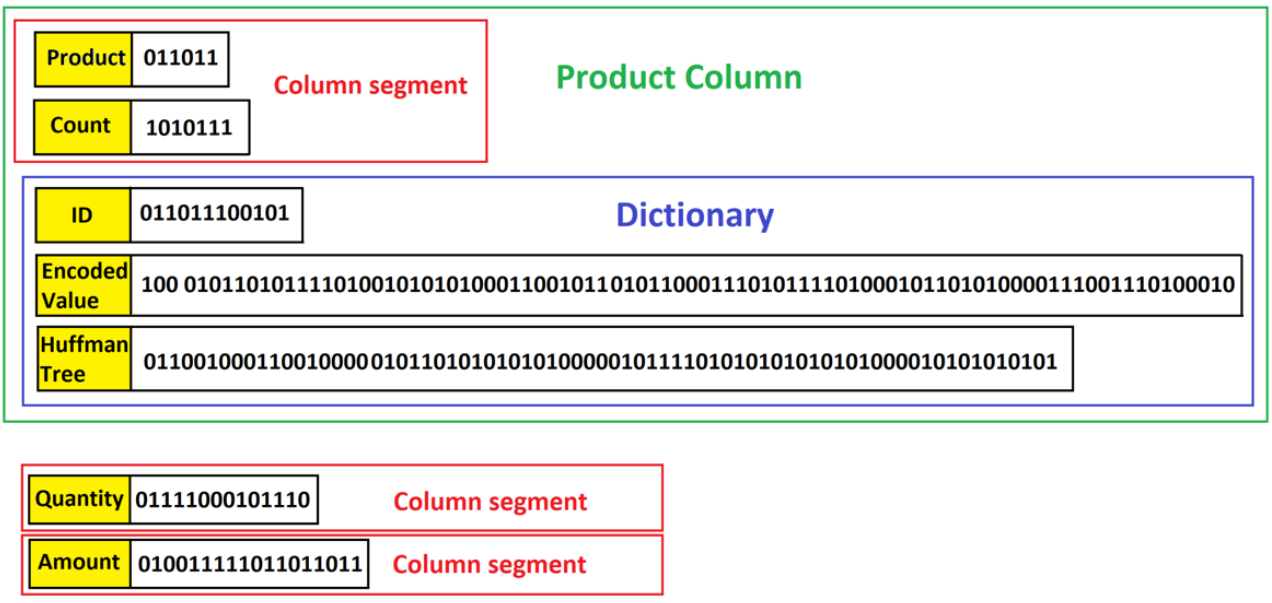

Рисунок 3.24 – Звичайна матеріалізація: уявлення колонок у бінарному стані

Після чого йде Bit Unpacking, тобто приведення даних до закодованого вигляду разом з словниками та іншими деталями, які були застосовані при кодуванні (рис. 3.25).

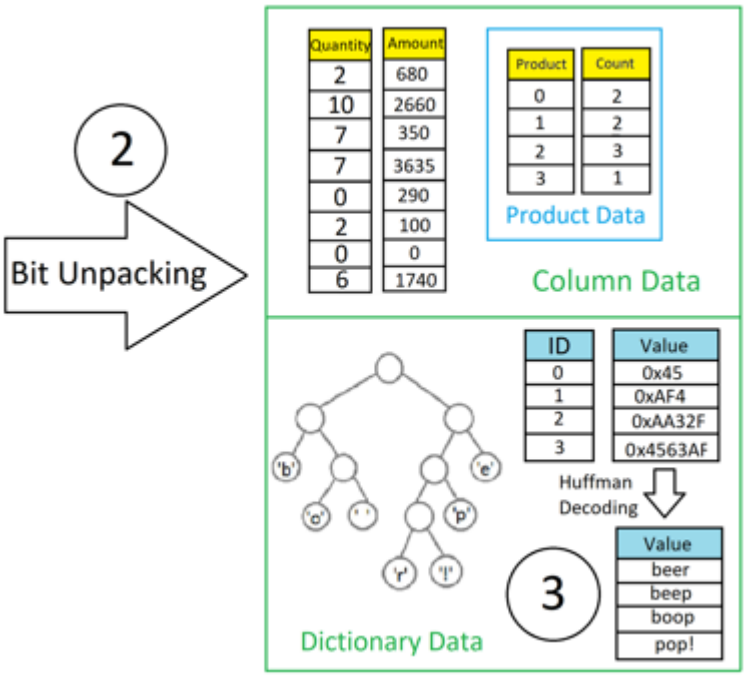

Рисунок 3.25 – Звичайна матеріалізація: приведення даних з бітового стану до десятичного

Можна побачити на рисунку (рис. 3.25), що були застосовані різні кодування від кодування значення до загального кодування RLE. Все це зберігалось у бітованому стані. Тепер задача системи полягає в тому, щоб декодувати всі ці вектори у нормальний (первинний вигляд) (рис. 3.26).

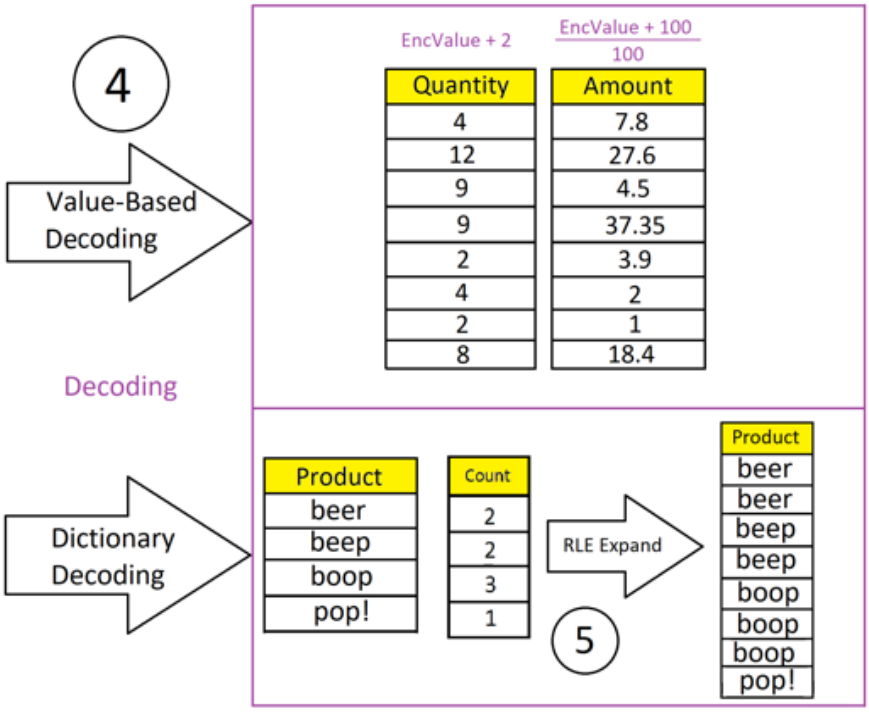

Рисунок 3.26 – Звичайна матеріалізація: декодування векторів

Після того, як повністю всі дані декодуються у всіх векторах, відбувається просте з'єднання цих векторів і приведення до табличного вигляду для подальшого відображення результату кінцевому користувачу (рис. 3.27).

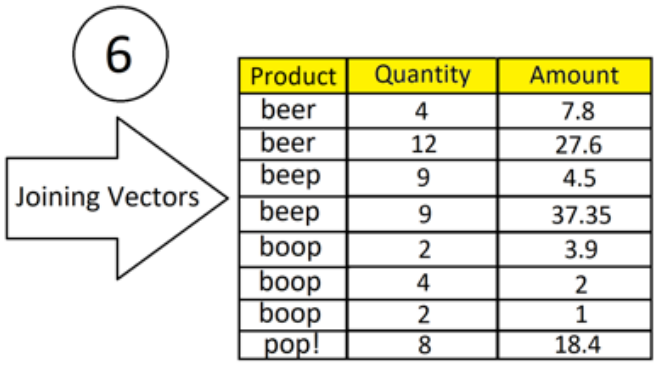

Рисунок 3.27 – Звичайна матеріалізація: з'єднання векторів і приведення до табличного вигляду

Отже, можна побачити, що при застосуванні любих запитів до таблиці відбувається повна матеріалізація даних для подальшої маніпуляції над даними (така природа колумнстор індексів).

Але алгоритм може виправити цю проблему способом застосування маніпуляцій на рівні закодованих значень, щоб зменшити навантаження на процесор при рахуванні агрегуючих функцій, тим самим зменшивши час виконання запиту [22].

Уявимо тепер, що є запит на таблицю, яка має 4 колонки з різними типами даних. У запиті будуть використані агрегуючі функції (рис. 3.28).

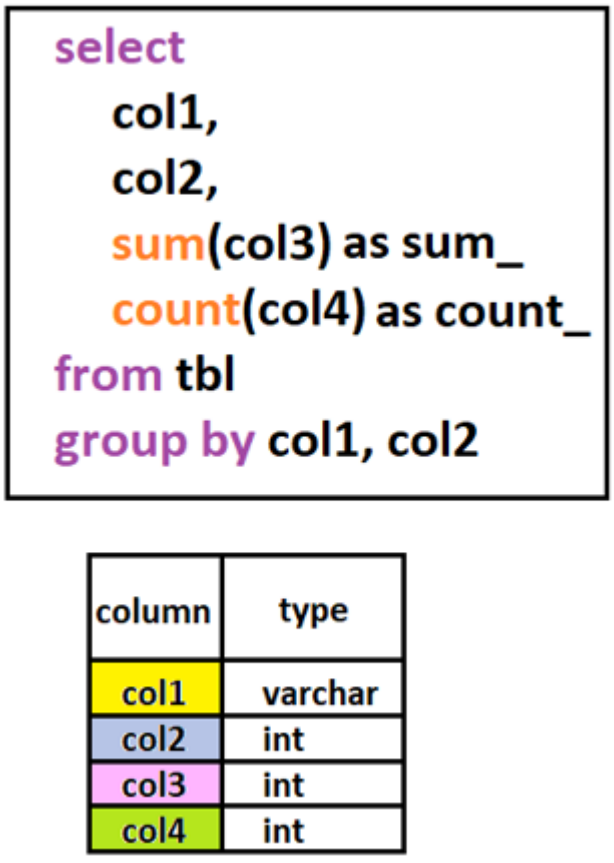

Рисунок 3.28 – Пришвидшення агрегації: Запит до таблиці з агрегуючими функціями

Отже, потрібно виконати відображення даних у табличному вигляді, де дві колонки будуть просто згруповані, а інші дві – проагреговані. На цій таблиці застосований колумнстор індекс. Далі необхідно зчитати метадані з системної таблиці про роугрупи для цього індексу: виявити всі закриті, відкриті та закомпресовані роугрупи (рис. 3.29).

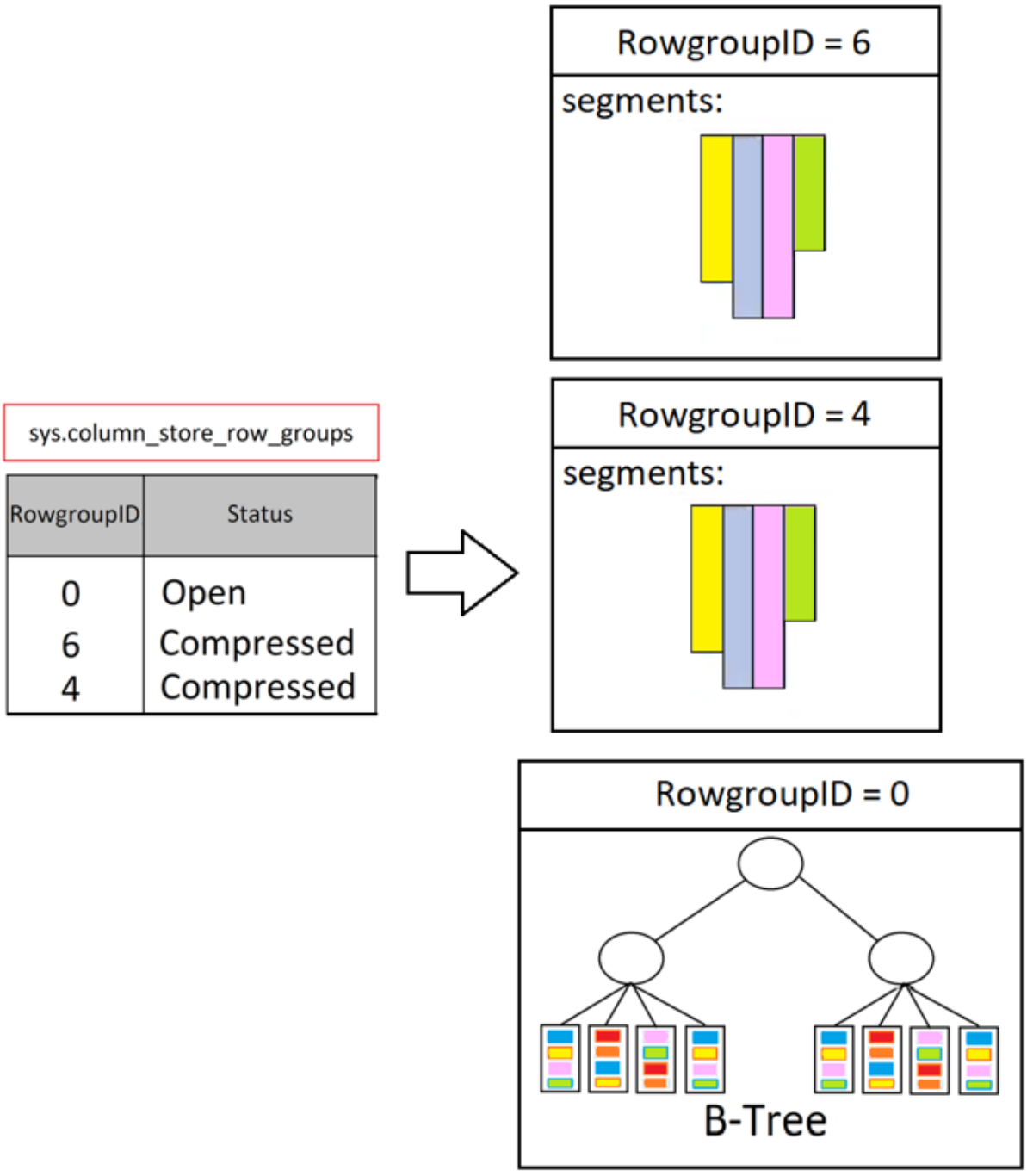

Рисунок 3.29 – Пришвидшення агрегації: виявлення роугруп

Наприклад, у нас є одна відкрита роугрупа та дві закриті, це означає, що для відкритої роугрупи не було застосовано алгоритмів кодування – це звичайне табличне уявлення даних, яке зберігається як звичайний індекс. Отже, для відкритої роугрупи не буде застосовано алгоритм пришвидшеної агрегації, ця роугрупа буде проагрегована окремо. Наступним кроком необхідно зробити Bit Unpacking, щоб привести дані векторів у десятичний вигляд замість бінарного (рис. 3.30).

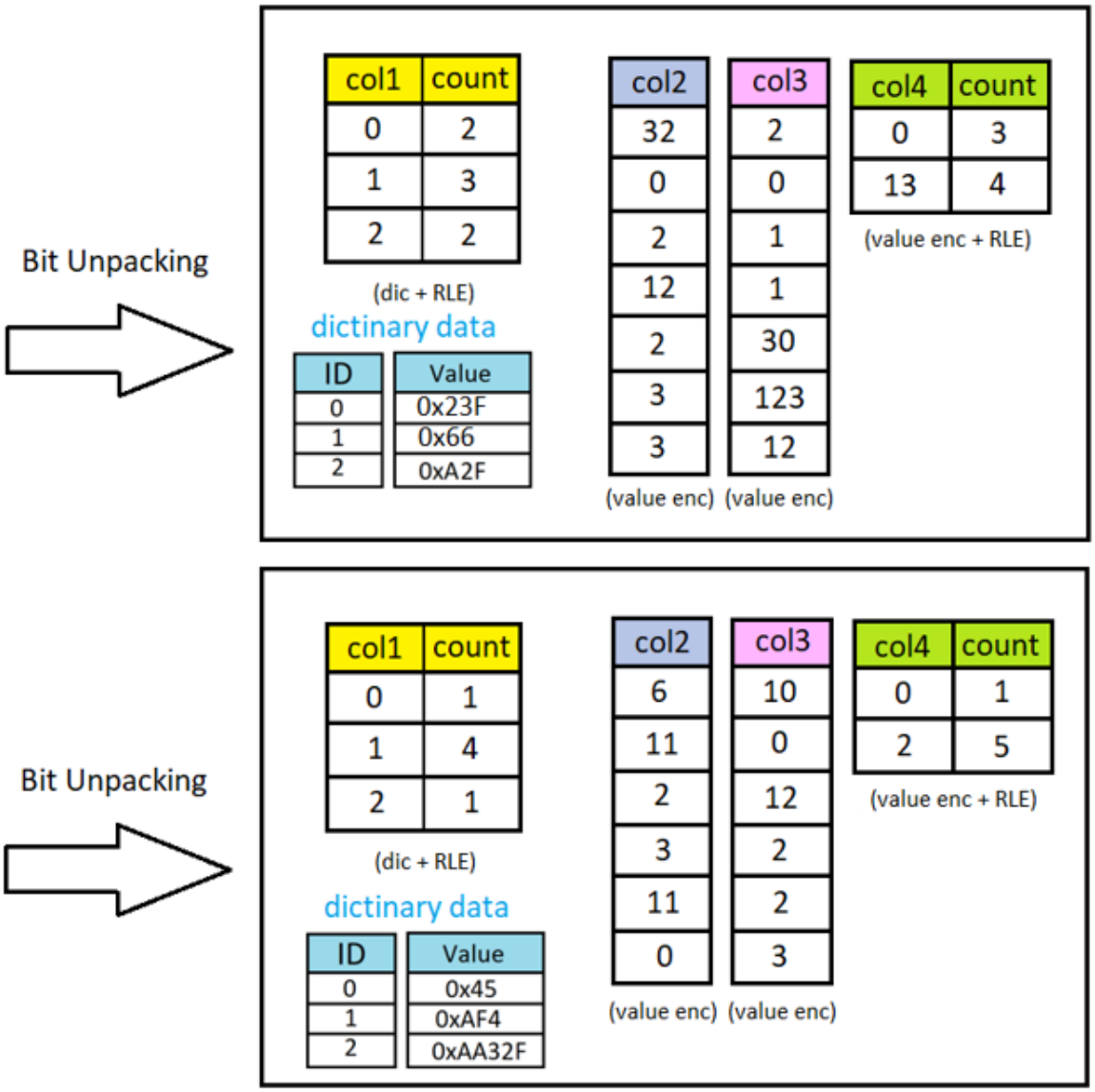

Рисунок 3.30 – Пришвидшення агрегації: Bit Unpacking

Далі необхідно зробити часткову матеріалізацію, бо якщо було застосовано кодування значенням, то буде не точно проведена агрегація над такими закодованими значеннями. У кожному такому кодуванні існує база, яка віднімається від мінімального значення так, щоб замість нього став нуль. А отже при агрегації над таким закодованим значенням буде неточний результат (рис. 3.31).

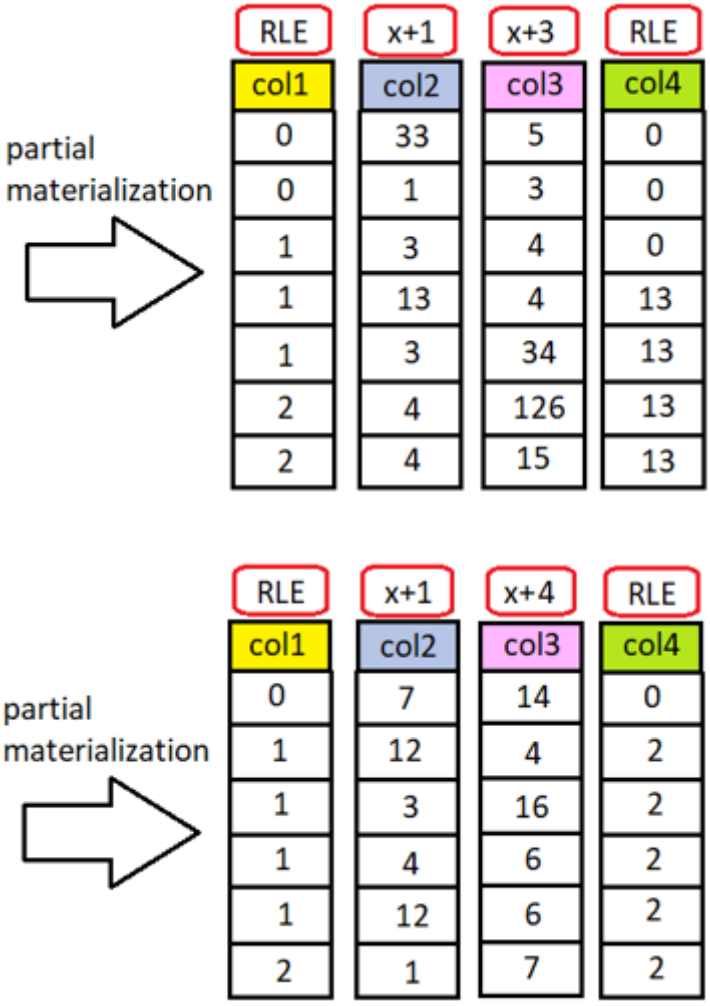

Рисунок 3.31 – Пришвидшення агрегації: часткова матеріалізація векторів

Далі відбувається агрегація закодованих значень, але так як в нас у запиті використовуються групування, то це буде робитись ще й з використанням хеш-функцій. До цих функцій будуть надходити дані з

векторів, які не виконують агрегаційні функції (рис. 3.32).

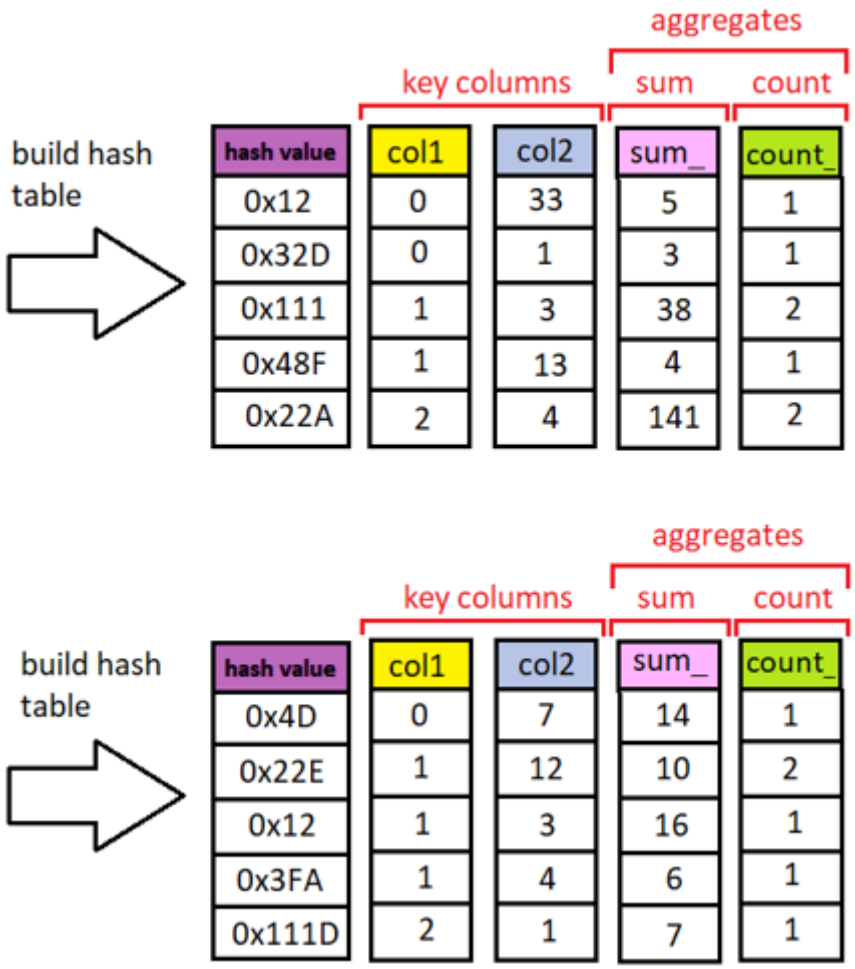

Рисунок 3.32 – Пришвидшення агрегації: агрегація закодованих даних

Після того, як дані вектори були повністю проагриговані, то наступним кроком вже буде повна матеріалізація, тобто повне декодування даних для кожного з векторів окремо (рис. 3.33).

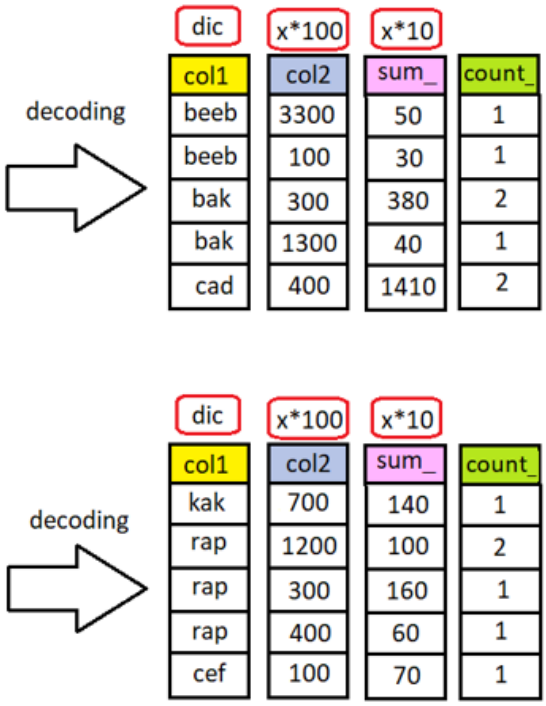

Рисунок 3.33 – Пришвидшення агрегації: повна матеріалізація окремих векторів

Потім просто поєднуються всі вектори и закомпресовані роугрупи до єдиного табличного вигляду (рис. 3.34).

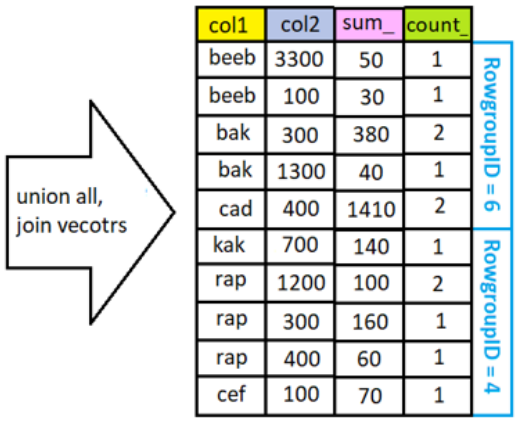

Рисунок 3.34 – Пришвидшення агрегації: поєднання векторів і закомпресованих роугруп

Також необхідно з’єднати з компресованими роугруми і відкриті (рис. 3.35).

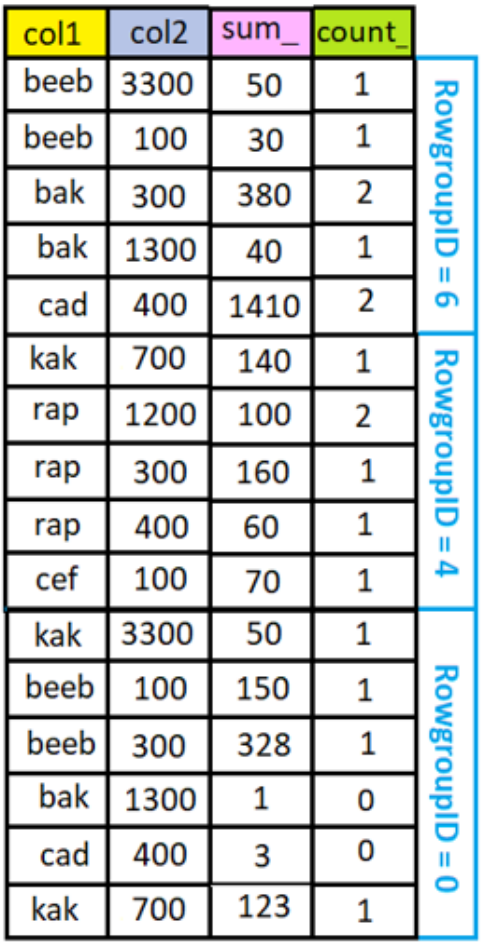

Рисунок 3.35 – Пришвидшення агрегації: з’єднання всіх роугруп

Далі, щоб групування і агрегація задіялось повністю для всіх рядків необхідно застосувати ще раз хеш-функцію, на яку подаються знову дві неагргуючі колонки (рис. 3.36). Це відбувається лише у випадку, коли існує відкрита чи закрита роугрупи. В інших випадках ці зайві дії не відбуваються, а одразу створюється табличний вигляд для кінцевого користувача (рис. 3.37).

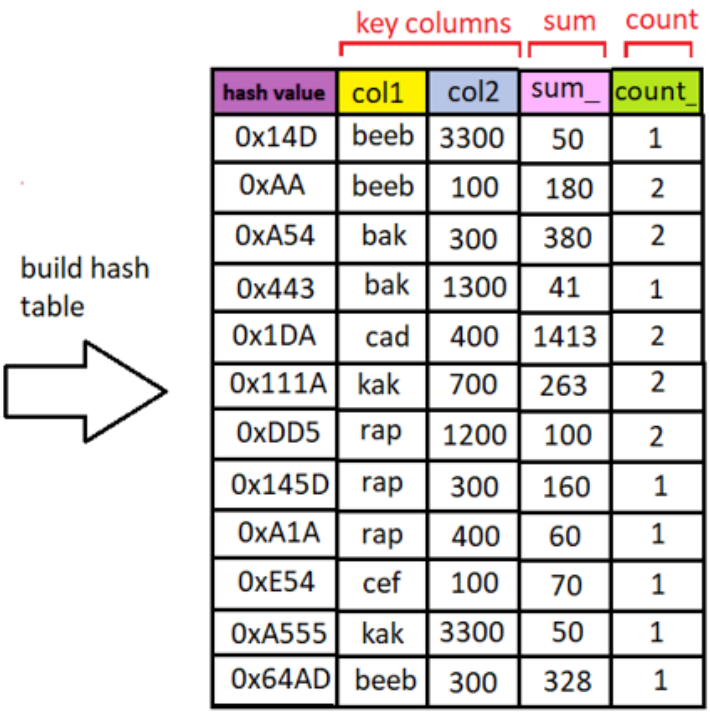

Рисунок 3.36 – Пришвидшення агрегації: групування всіх результатів з кожної роугрупи

col1	col2	sum_	count_
beeb	3300	50	1
beeb	100	180	2
bak	300	380	2
bak	1300	41	1
cad	400	1413	2
kak	700	263	2
rap	1200	100	2
rap	300	160	1
rap	400	60	1
cef	100	70	1
kak	3300	50	1
beeb	300	328	1

Рисунок 3.37 – Пришвидшення агрегації: кінцевий результат

Використання такого методу дуже добре пришвидшить виконання запитів, які застосовують у звичайних виразах агрегаційні функції.

3.5 Висновки до третього розділу

Було розглянуто математичні моделі реалізації алгоритму компресії, в яких використовуються переважно математичні маніпуляції для зменшення загальної ваги вектору у роугрупі. Також було розглянуто існуючі алгоритми компресії даних для різних типів даних та випадків.

Розроблений власний алгоритм для компресії даних у випадку, коли використовується цифрові значення у векторі і якщо мантіса та база не значно зменшили загальну вагу вектору у кодуванні значенням. Розроблений власний алгоритм для пришвидшення виконання запитів з використанням агригаційних функцій.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ФУНКЦІОНУВАННЯ АНАЛІТИЧНИХ БАЗ ДАНИХ

4.1 Архітектура бази даних та її властивості

Існує БД, яка зберігає всю інформацію про покупки різних продуктових товарів. Зазвичай БД для аналітичних задач мають архітектуру зірки, яка являє собою одна таблиця фактів і таблиці вимірів для цієї таблиці фактів (рис. 4.1).

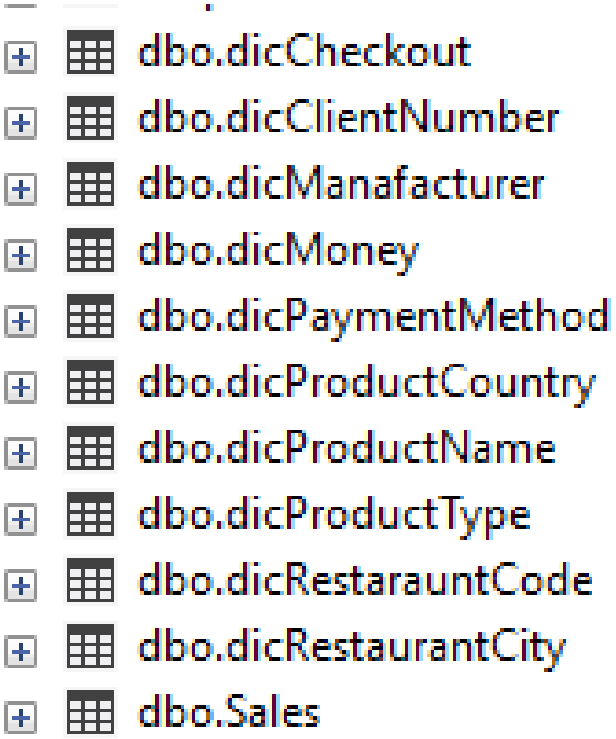

Рисунок 4.1 – База даних магазину

На зображенні (рис. 4.1) бачимо, що є 10 таблиць вимірів та одна таблиця фактів «Sales». Ця БД показує лише дані про продажі товарів, ніяких складів для товарів чи додаткової інформації про виробника.

Зазвичай створюють колумнстор індекси на таку велику таблицю, для того, щоб швидше робилися запити, бо на даний час не було створено індексу (рис.4.2, рис.4.3).

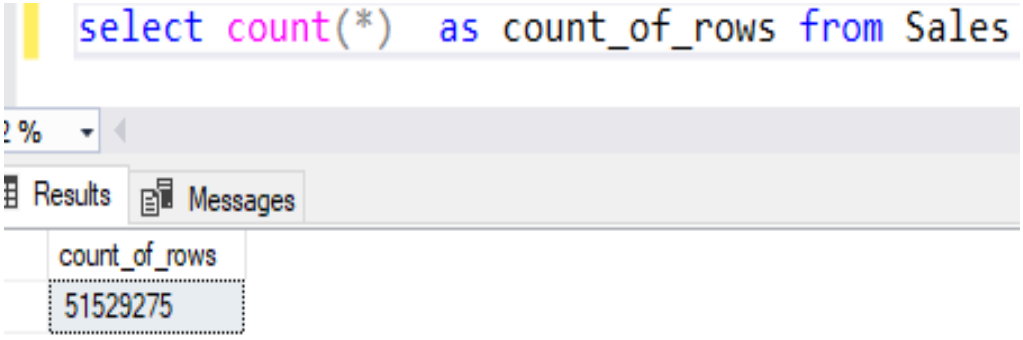

Рисунок 4.2 – Запит на відображення всіх рядків з таблиці фактів

Рисунок 4.3 – Кількість затраченого часу на виконання запиту для відображення всіх рядків таблиці фактів без використання індексу

Отже можна побачити з рисунків (рис. 4.2 та рис. 4.3), що кількість рядків у факт-таблиці сягає 51 529 275 мільйонів рядків, а затрачений час на виконання запиту для відображення всіх рядків факт-таблиці без використання індексу дорівнює 2 хвилини та 11 секунд. Так, це звичайно ще залежить від комп’ютера, який виконує ці запити. Дуже важливу роль грає процесор та оперативна пам’ять. Для виконання запитів було використано процесор Intel Core i5-8100 та 2 плати оперативної пам’яті по 4 гігабайти DD3.

Таблиця фактів містить наступну частину основної інформації (рис. 4.4).

ID	ClientNumber	ClaimBonus	Checkout	Amount	Discount_%	PaymentMethod	ProductType	ProductName	ProductPrice	ProductCountry	Manafacturer	RestaurantCity	RestaurantCode	Order_ID	TxnDatetime
826	MC0002	5,39	Checkout 8	134,75	23	Cash	Fish	Beef	175,00	Poland	Roshen	Odesa	KR48	83479	2022-12-04 15:29:29.807
827	MC0006	5,508	Checkout 7	137,70	10	Card	Dairy	Nuts	153,00	Poland	Roshen	Kharkiv	KR61	744066	2022-12-04 15:29:29.807
828	MC0008	5,6776	Checkout 2	141,94	6	Card	Meat	Pork	151,00	France	Dobra	Odesa	KR49	182272	2022-12-04 15:29:29.810
829	MC0006	10,2664	Checkout 2	256,66	18	Card	Fish	Juce	313,00	Poland	Carboona	Sevostopol	KR49	22578	2022-12-04 15:29:29.810
830	MC0014	6,912	Checkout 7	172,80	20	Cash	Fish	Milk	216,00	France	Roshen	Odesa	KR48	199745	2022-12-04 15:29:29.810
831	MC0005	7,568	Checkout 6	189,20	14	Cash	Vegetables	Bread	220,00	Sweden	Milka	Kyiv	KR56	941112	2022-12-04 15:29:29.810
832	MC0014	8,7048	Checkout 2	217,62	22	Cash	Fruit	Banana	279,00	Germany	Kulik Systems	Sevostopol	KA46	742148	2022-12-04 15:29:29.810
833	VP005	3,2144	Checkout 7	80,36	18	Card	Fruit	Bread	98,00	Poland	Carboona	Kharkiv	KR50	449192	2022-12-04 15:29:29.810
834	VP002	7,76	Checkout 6	194,00	3	Card	Meat	Bread	200,00	France	Roshen	Donetsk	KR47	548308	2022-12-04 15:29:29.810
835	P0005	2,0592	Checkout 6	51,48	1	Cash	Chocolate	Juce	52,00	Poland	Shiza	Sevostopol	KR55	451231	2022-12-04 15:29:29.810
836	P0001	9,7636	Checkout 1	244,09	23	Card	Vegetables	Nuts	317,00	Germany	Milka	Kyiv	KR55	243765	2022-12-04 15:29:29.813
837	P0005	6,014	Checkout 2	150,35	3	Cash	Dairy	Beef	155,00	Poland	Dobra	Odesa	KA45	163438	2022-12-04 15:29:29.813
838	MC0001	6,916	Checkout 1	172,90	9	Cash	Dairy	Pork	190,00	Turkey	Kulik Systems	Donetsk	KR60	886842	2022-12-04 15:29:29.813
839	MC0008	11,894	Checkout 4	297,35	5	Card	Dairy	Milk	313,00	Turkey	Shiza	Donetsk	KR56	722929	2022-12-04 15:29:29.813
840	P0004	4,374	Checkout 1	109,35	19	Cash	Vegetables	Chicken	135,00	Poland	Dobra	Odesa	KR58	951189	2022-12-04 15:29:29.813
841	MC0003	11,264	Checkout 1	281,60	12	Cash	Vegetables	Tomato	320,00	Germany	Dobra	Dnipro	KR49	212090	2022-12-04 15:29:29.813
842	P0006	3,1684	Checkout 4	79,21	11	Cash	Dairy	Apple	89,00	France	Shiza	Sevostopol	KR51	778633	2022-12-04 15:29:29.817
843	MC0004	6,3516	Checkout 8	158,79	21	Cash	Chocolate	Apple	201,00	Germany	Shiza	Dnipro	KR62	777041	2022-12-04 15:29:29.817
844	MC0002	5,2668	Checkout 8	131,67	1	Cash	Bread	Milk	133,00	Norway	Milka	Kyiv	KR48	992966	2022-12-04 15:29:29.817
845	P0007	3,2032	Checkout 7	80,08	12	Cash	Fish	Perch	91,00	Ukraine	Milka	Kharkiv	KR49	174958	2022-12-04 15:29:29.817
846	MC0013	8,9744	Checkout 2	224,36	21	Card	Chocolate	Milk	284,00	Norway	Carboona	Lviv	KR47	537987	2022-12-04 15:29:29.817
847	MC0004	3,1248	Checkout 3	78,12	16	Cash	Dairy	Chicken	93,00	Poland	Milka	Kharkiv	KR53	522986	2022-12-04 15:29:29.817
848	MC0010	4,2036	Checkout 1	105,09	7	Card	Bread	Bread	113,00	Sweden	Kulik Systems	Donetsk	KA45	199184	2022-12-04 15:29:29.820
849	MC0008	3,192	Checkout 3	79,80	5	Cash	Fish	Banana	84,00	Ukraine	Dobra	Dnipro	KR60	326868	2022-12-04 15:29:29.820
850	MC0005	2,4288	Checkout 2	60,72	12	Card	Bread	Milk	69,00	Germany	Dobra	Sevostopol	KR60	784887	2022-12-04 15:29:29.820
851	MC0011	8,1432	Checkout 2	203,58	13	Cash	Fruit	Nuts	234,00	Germany	Shiza	Dnipro	KR53	668465	2022-12-04 15:29:29.820
852	P0005	7,9692	Checkout 3	199,23	13	Card	Dairy	Bread	229,00	Germany	Kulik Systems	Dnipro	KR57	222733	2022-12-04 15:29:29.820
853	P0005	9,306	Checkout 1	232,65	1	Cash	Dairy	Pork	235,00	Germany	Regina	Donetsk	KR57	739874	2022-12-04 15:29:29.820
854	MC0013	8,0464	Checkout 8	201,16	6	Cash	Meat	Snikers	214,00	Poland	Shiza	Odesa	KR56	646153	2022-12-04 15:29:29.820
855	MC0001	4,232	Checkout 3	105,80	8	Card	Fruit	Tomato	115,00	Ukraine	Roshen	Lviv	KR48	625293	2022-12-04 15:29:29.820
856	MC0005	11,196	Checkout 8	279,90	10	Card	Dairy	Snikers	311,00	Sweden	Milka	Odesa	KR50	212164	2022-12-04 15:29:29.823
857	P0007	10,7824	Checkout 3	269,56	8	Cash	Bread	Perch	293,00	Poland	Kulik Systems	Odesa	KR57	823220	2022-12-04 15:29:29.823
858	VP005	1,2528	Checkout 2	31,32	13	Cash	Fruit	Perch	36,00	Turkey	Dobra	Kharkiv	KR51	151950	2022-12-04 15:29:29.823
859	MC0012	0,9072	Checkout 6	22,68	16	Cash	Vegetables	Apple	27,00	France	Regina	Sevostopol	KR51	700434	2022-12-04 15:29:29.823
860	P0003	9,6096	Checkout 3	240,24	23	Cash	Meat	Salmon	312,00	Poland	Kulik Systems	Dnipro	KR52	858184	2022-12-04 15:29:29.823
861	MC0003	5,6916	Checkout 8	142,29	7	Card	Dairy	Perch	153,00	Ukraine	Roshen	Sevostopol	KR63	162769	2022-12-04 15:29:29.827
862	MC0004	2,94	Checkout 7	73,50	25	Cash	Fish	Juce	98,00	Sweden	Regina	Kyiv	KR51	682176	2022-12-04 15:29:29.827
863	P0005	4,4352	Checkout 1	110,88	1	Card	Meat	Banana	112,00	France	Milka	Lviv	KR58	780509	2022-12-04 15:29:29.827

Рисунок 4.4 – Частина основної інформації факт-таблиці

Також таблиця фактів містить додаткову інформації, яка являє собою ключі до таблиць вимірів (рис. 4.5).

Checkout_ID	ClientNumber_ID	Amount_ID	Manafacturer_ID	ProductPrice_ID	PaymentMethod_ID	ProductCountry_ID	ProductName_ID	ProductType_ID	RestaurantCode_ID	RestaurantCity_ID
8	2	25	2	25	2	2	11	5	2	6
7	6	25	2	25	1	2	2	1	15	1
2	8	25	3	25	1	6	12	4	3	6
2	6	26	5	26	1	2	9	5	3	4
7	14	25	2	25	2	6	8	5	2	6
6	5	25	7	25	2	4	7	6	10	2
2	14	25	6	26	2	3	3	7	20	4
7	19	24	5	24	1	2	7	7	4	1
6	16	25	2	25	1	6	7	4	1	5
6	25	24	4	24	2	2	9	2	9	4
1	21	25	7	26	1	3	2	6	9	2
2	25	25	3	25	2	2	11	1	18	6
1	1	25	6	25	2	7	12	1	14	5
4	8	26	4	26	1	7	8	1	10	5
1	24	25	3	25	2	2	10	6	12	6
1	3	26	3	26	2	3	6	6	3	7
4	26	24	4	24	2	6	4	1	5	4
8	4	25	4	25	2	3	4	2	16	7
8	2	25	7	25	2	5	8	3	2	2
7	27	24	7	24	2	1	14	5	3	1
2	13	25	5	26	1	5	8	2	1	3
3	4	24	7	24	2	2	10	1	7	1
1	10	25	6	25	1	4	7	3	18	5
3	8	24	3	24	2	1	3	5	14	7
2	5	24	3	24	1	3	8	3	14	4
2	11	25	4	25	2	3	2	7	7	7
3	25	25	6	25	1	3	7	1	11	7
1	25	25	1	25	2	3	12	1	11	5
8	13	25	4	25	2	2	1	4	10	6
3	1	25	2	25	1	1	6	7	2	3
8	5	26	7	26	1	4	1	1	4	6
3	27	26	6	26	2	2	14	3	11	6
2	19	23	3	23	2	7	14	7	5	1
6	12	22	1	23	2	6	4	6	5	4
3	23	25	6	26	2	2	13	4	6	7
8	3	25	2	25	1	1	14	1	17	4
7	4	24	1	24	2	4	9	5	5	2
1	25	25	7	25	1	6	3	4	12	3

Рисунок 4.5 – Частина інформації з таблиці про ключі до таблиць вимірів

Таблиці виміри виглядають як звичайні словники, де кожний рядок є унікальним (рис. 4.6 – рис. 4.7).

```
select * from dicMoney
```

132 %

Results | Messages

	Group1_ID	Group2_ID	Group3_ID	ID	Code	Description	Group1	Group2	Group3	MinValue	MaxValue
1	0	0	0	0	None	NULL	None	None	None	NULL	NULL
2	1	1	1	1	Zero	NULL	Zero	Zero	Zero	0,00	0,00
3	2	2	2	2	Non Zero	NULL	Non Zero	Non Zero	Non Zero	NULL	NULL
4	2	3	3	3	Credit	NULL	Non Zero	Credit	Credit	NULL	NULL
5	2	3	4	4	< $250	NULL	Non Zero	Credit	< $250	NULL	NULL
6	2	3	4	5	$0 - $10	NULL	Non Zero	Credit	< $250	-10,00	-0,01
7	2	3	4	6	$10 - $25	NULL	Non Zero	Credit	< $250	-25,00	-10,01
8	2	3	4	7	$25 - $50	NULL	Non Zero	Credit	< $250	-50,00	-25,01
9	2	3	4	8	$50 - $100	NULL	Non Zero	Credit	< $250	-100,00	-50,01
10	2	3	4	9	$100 - $250	NULL	Non Zero	Credit	< $250	-250,00	-100,01
11	2	3	5	10	$250 - $500	NULL	Non Zero	Credit	$250 - $500	-500,00	-250,01
12	2	3	6	11	$500 - $1000	NULL	Non Zero	Credit	$500 - $1000	-1000,00	-500,01
13	2	3	7	12	$1000 - $2500	NULL	Non Zero	Credit	$1000 - $2500	-2500,00	-1000,01
14	2	3	8	13	$2500 - $5000	NULL	Non Zero	Credit	$2500 - $5000	-5000,00	-2500,01
15	2	3	9	14	$5000 - $10...	NULL	Non Zero	Credit	$5000 - $10...	-10000,00	-5000,01
16	2	3	10	15	$10000 - $2...	NULL	Non Zero	Credit	$10000 - $2...	-25000,00	-10000,01
17	2	3	11	16	$25000 - $5...	NULL	Non Zero	Credit	$25000 - $5...	-50000,00	-25000,01
18	2	3	12	17	$50000 - $1...	NULL	Non Zero	Credit	$50000 - $1...	-100000,00	-50000,01
19	2	3	13	18	> $100000	NULL	Non Zero	Credit	> $100000	-922337203685477,5808	-100000,01
20	2	4	14	19	Debit	NULL	Non Zero	Debit	Debit	NULL	NULL
21	2	4	15	20	< $250	NULL	Non Zero	Debit	< $250	NULL	NULL
22	2	4	15	21	$0 - $10	NULL	Non Zero	Debit	< $250	0,01	10,00
23	2	4	15	22	$10 - $25	NULL	Non Zero	Debit	< $250	10,01	25,00
24	2	4	15	23	$25 - $50	NULL	Non Zero	Debit	< $250	25,01	50,00
25	2	4	15	24	$50 - $100	NULL	Non Zero	Debit	< $250	50,01	100,00
26	2	4	15	25	$100 - $250	NULL	Non Zero	Debit	< $250	100,01	250,00
27	2	4	16	26	$250 - $500	NULL	Non Zero	Debit	$250 - $500	250,01	500,00
28	2	4	17	27	$500 - $1000	NULL	Non Zero	Debit	$500 - $1000	500,01	1000,00
29	2	4	18	28	$1000 - $2500	NULL	Non Zero	Debit	$1000 - $2500	1000,01	2500,00
30	2	4	19	29	$2500 - $5000	NULL	Non Zero	Debit	$2500 - $5000	2500,01	5000,00
31	2	4	20	30	$5000 - $10...	NULL	Non Zero	Debit	$5000 - $10...	5000,01	10000,00
32	2	4	21	31	$10000 - $2...	NULL	Non Zero	Debit	$10000 - $2...	10000,01	25000,00
33	2	4	22	32	$25000 - $5...	NULL	Non Zero	Debit	$25000 - $5...	25000,01	50000,00
34	2	4	23	33	$50000 - $1...	NULL	Non Zero	Debit	$50000 - $1...	50000,01	100000,00
35	2	4	24	34	> $100000	NULL	Non Zero	Debit	> $100000	100000,01	922337203685477,5807

Рисунок 4.6 – Таблиця вимір dicMoney

```
select * from dicManafacturer
```

132 %

Results | Messages

	ID	Code	Description
1	0	None	None
2	1	Regina	Regina manafacturer
3	2	Roshen	Roshen manafacturer
4	3	Dobra	Dobra manafacturer
5	4	Shiza	Shiza manafacturer
6	5	Carboona	Carboona manafacturer
7	6	Kulik Systems	Kulik Systems manafacturer
8	7	Milka	Milka manafacturer

Рисунок 4.7 – Таблиця вимір dicManufacrurer

Ці таблиці будуть використовуватись в OLAP-технології, як окремі виміри, які вже потім будуть використовуватись в кубі.

Вага факт-таблиці до того, як був задіяний columnstore index до таблиці фактів дорівнює 7 957,297 Мб. Це доволі не маленький розмір (рис. 4.8).

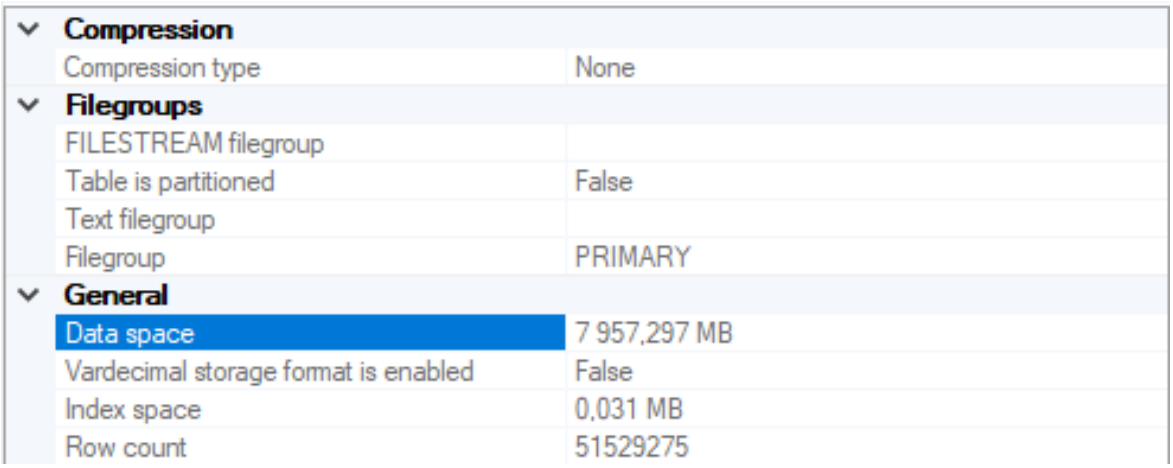

Рисунок 4.8 – Характеристики таблиці фактів

Побудова індексу займає не мало часу, бо потрібно, в першу чергу зчитати велику кількість пейджів, потім створити роугрупи та скомпресувати ці роугрупи, повністю замінивши архітектуру збереження даних цієї таблиці (рис. 4.9).

Рисунок 4.9 – Час створення колумнстор індексу на таблицю фактів

Після того, як був створений індекс на таблицю фактів, розміри самої таблиці дуже сильно змінились (майже в 7 разів). Теперішня вага таблиці становить 806,984 Мб (рис. 4.10).

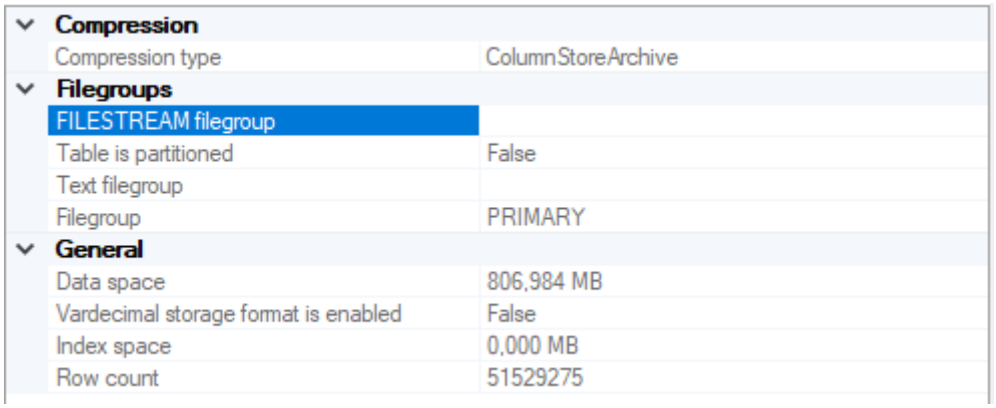

Рисунок 4.10 – Характеристики таблиці фактів з індексом

4.2 Експерименти по оптимізації КД та пришвидшенню запитів за допомогою розроблених методів та алгоритмів

Ми можемо побачити, що для колонки Order_ID було використано кодування значенням, а дані, які зберігаються там не дуже добре підходять під цей алгоритм. Можна побачити, що в усіх випадках роугруп не була задіяна мантіса, а в деяких випадках навіть база не задіяна (рис. 4.11).

partition_id	hobt_id	column_id	segment_id	version	encoding_type	row_count	has_nulls	base_id	magnitude	primary_dictionary_id	secondary_dictionary_id	min_data_id	max_data_id	null_value	on_disk_size
72057594046906368	72057594046906368	1	3	-2147483647	1	1048576	0	0	1	-1	-1	3	47531523	-1	3613791
72057594046906368	72057594046906368	1	4	-2147483647	1	1048576	0	4	1	-1	-1	7	47534031	-1	3622401
72057594046906368	72057594046906368	1	5	-2147483647	1	1048576	0	16	1	-1	-1	19	47533525	-1	3618003
72057594046906368	72057594046906368	1	6	-2147483647	1	1048576	0	5	1	-1	-1	8	28113328	-1	3619685
72057594046906368	72057594046906368	1	7	-2147483647	1	1048576	0	18286	1	-1	-1	18289	40805749	-1	3694719
72057594046906368	72057594046906368	1	8	-2147483647	1	1048576	0	18279	1	-1	-1	18282	40713200	-1	3694031
72057594046906368	72057594046906368	1	9	-2147483647	1	1048576	0	18284	1	-1	-1	18287	40806825	-1	3683409
72057594046906368	72057594046906368	1	10	-2147483647	1	1048576	0	18276	1	-1	-1	18279	40805688	-1	3678107
72057594046906368	72057594046906368	1	11	-2147483647	1	1048576	0	18366	1	-1	-1	18369	40807060	-1	3675623
72057594046906368	72057594046906368	1	12	-2147483647	1	1048576	0	18278	1	-1	-1	18281	40806843	-1	3687973
72057594046906368	72057594046906368	1	13	-2147483647	1	1048576	0	14834371	1	-1	-1	14834374	47532624	-1	3687637
72057594046906368	72057594046906368	1	14	-2147483647	1	1048576	0	4543	1	-1	-1	4546	51526514	-1	3685833
72057594046906368	72057594046906368	1	15	-2147483647	1	1048576	0	1561	1	-1	-1	1564	51527391	-1	3696693
72057594046906368	72057594046906368	1	16	-2147483647	1	1048576	0	255	1	-1	-1	258	51529157	-1	3697457
72057594046906368	72057594046906368	1	17	-2147483647	1	1048576	0	468	1	-1	-1	471	51524874	-1	3685909
72057594046906368	72057594046906368	1	18	-2147483647	1	1048576	0	3037	1	-1	-1	3040	51529011	-1	3671533
72057594046906368	72057594046906368	1	19	-2147483647	1	1048576	0	36	1	-1	-1	39	51529246	-1	3615727
72057594046906368	72057594046906368	1	20	-2147483647	1	1048576	0	32	1	-1	-1	35	51529227	-1	3641555
72057594046906368	72057594046906368	1	21	-2147483647	1	1048576	0	47	1	-1	-1	50	51529239	-1	3652505

Рисунок 4.11 – Дані про компресію колонки Order_ID

Для таких випадків було б доцільно використовувати саме новий алгоритм, який дозволить зробити сильнішу компресію таких типів даних. Зазвичай, ситуація, коли буде задіяна мантіса, дуже рідкісна, тому майже завжди новий алгоритм буде краще ніж кодування значенням, а інколи й хеш-

кодування.

На заображенні (рис. 4.12) запит для аналітичних задач, а саме обрати кількість виробників і витрачену суму для кожного клієнта з окремими ордером.

```
select
    ClientNumber,
    Order_ID,
    sum(ProductPrice),
    count(Manafacturer)
from Sales
group by ClientNumber, Order_ID
```

Results | Messages

	ClientNumber	Order_ID	(No column name)	(No column name)
	P0005	623889	307,00	1
	VP004	106124	366,00	3
	MC0011	942975	229,00	1
	VP002	346608	375,00	2
	VP005	870452	792,00	3
	P0001	323227	477,00	2
	P0003	4053	24,00	1
	MC0014	488711	900,00	4
	P0003	833809	433,00	3
0	VP005	621671	139,00	1
1	MC0013	180847	204,00	2
2	MC0009	804833	269,00	1
3	VP006	614537	366,00	2
4	VP002	677635	26,00	1
5	VP004	127436	466,00	3
6	P0008	330407	329,00	2
7	P0002	612478	113,00	2
8	MC0001	494072	772,00	5
9	MC0013	128974	275,00	2
0	P0008	367770	184,00	1
1	MC0006	112535	239,00	1
2	VP006	595244	1311,00	6
3	MC0002	728103	290,00	2
4	P0001	489203	182,00	2
5	MC0001	756798	623,00	3
6	MC0012	332930	70,00	1
7	VP006	798671	318,00	2
8	P0001	704626	964,00	3
9	MC0006	850422	75,00	1

Рисунок 4.12 – Звичайний запит для аналітичних задач

На малюнку (рис. 4.13) представлений план запиту: обрати кількість виробників і витрачену суму для кожного клієнта з окремими ордером. Це відображено схема того, як буде СУБД буде виконувати цей запит.

Рисунок 4.13 – Загальний план запиту

З приведених даних, можна побачити, що спочатку йде зчитування даних з пейджів (колумнстор індексу), потім повна матеріалізація даних роугруп, а вже потім розрахунок агрегаційних функцій.

Час виконання цього запиту можна побачити на малюнку (рис. 4.14).

Рисунок 4.14 – Час виконання запиту для аналітичних задач

Після зчитування стрілка показує, що до наступного вузлу (ноду) було переслано всі 51 529 300 мільйонів записів (рис. 4.15).

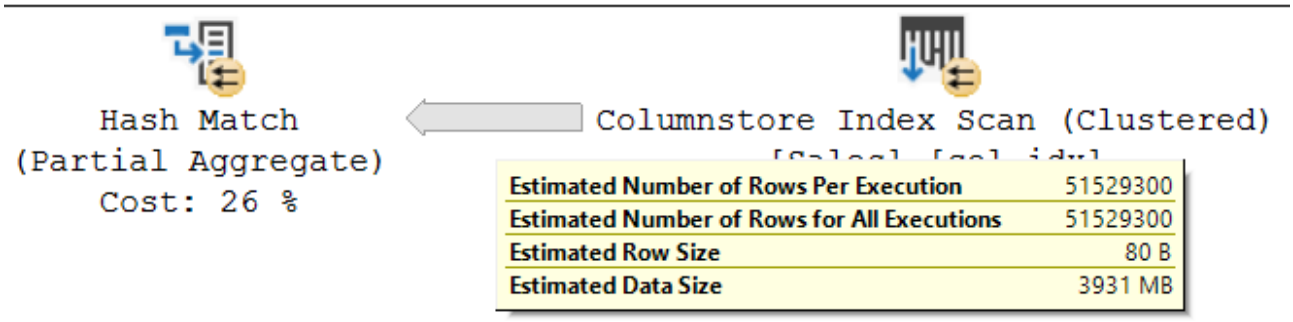

Рисунок 4.15 – Детальна статистика плану запиту

Але цього можна уникнути, використавши розроблений мною алгоритм швидкої агрегації, бо це дуже сильно зменшить навантаження на процесор та

оперативну пам’ять. Замість того, щоб оперувати над вже матеріалізованими, великими даними, процесор може це робити з закодованими відносно малими даними.

Якщо б алгоритм було задіяно, то вже не було би потреби використовувати агрегаційні вузли і стрілка показувала б не 51 мільйон пересланих рядків, а 23 554 139, як і у кінцевому результаті.

4.3 Охорона праці

Розробка аналітичних БД – це дуже важкий процес як для системи, яка зберігає та оброблює інформацію, так і для працівників, які створюють та підтримують БД у функціональному стані. Сервери, які зберігають дані – це дуже цінний ресурс для компанії. Саме там зберігається вся інформація, яка необхідна для аналітичних задач різних клієнтів. Якщо ця інформація буде витрачена або зіпсована, то наслідками можуть бути: великі грошові збитки та втрата репутації, що в свою чергу – є основою для любого бізнесу [23].

Щоб не відбувалися важкі наслідки від втрати, або зіпсування клієнтських даних, персоналу, який підтримує функціонування БД та керівникам, необхідно слідувати наступним правилам:

– при роботі над зміною архітектури БД чи логіки процедури та калькуляції необхідно спочатку зробити власний бекап всієї БД, щоб після того, якщо відбудеться втрата інформації чи не бажаний результат, персонал міг швидко відновити БД;

– створити три рівні БД для розробки та тестування, для імпорту нових клієнтських даних та для самих клієнтів. Клієнтська БД – це самий важливий рівень. Дуже не рекомендовано, щоб щось трапилось з цією БД, коли клієнт буде нею користуватись;

– при внесенні нових клієнтських побажань щодо функціоналу БД, спочатку необхідно зробити зміни БД для розробки та тестування, вже після чого переносити на наступні рівні;

– ніколи не змінювати клієнтські файли, які надсилаються для імпорту до БД, щоб потім не було непорозумінь з клієнтами;

– завжди перевіряти та налаштовувати ETL-процес до та після його виконання;

– ніколи не використовувати більше 20% ядер процесору на сервері, щоб не навантажувати систему для інших задач. Це можна зробити лише зі згоди керівника та мати важливу причину використовувати більше 20%. Це може трапитись, коли персонал не встигає доробити якусь задачу перед запуском ETL-процесу;

– ніколи не змінювати налаштування клієнтів на БД чи на сайті, де клієнти зазвичай працюють над аналітикою свого власного бізнесу;

– вчасно попереджати керівників, якщо трапився якийсь збій у системі функціонування БД.

Окрім бізнес-правил необхідно ще не порушувати технічні правила щодо налаштування серверів та підтримки оптимального оточення для них:

– слідкувати за температурою приміщення, де зберігаються фізично сервери, щоб температура не перевищувала задану норму;

– слідкувати, щоб у приміщенні волога також не перевищувала задану норму. Забезпечити надійний захист від зовнішнього середовища;

– регулярно фізично перевіряти та оновляти сервери, щоб у відповідний момент вони не згоріли чи не створили умови для появлення багів.

4.4 Висновки до четвертого розділу

Була розроблена аналітична база даних, розглянута її архітектура та початкові властивості. Ця база даних розроблена виключно для аналітичних задач, а це означає, що створюється ця база даних для того, щоб при перенесені її до OLAP-системи, було простіше порахувати агрегації. Між цим були проведені експерименти щодо швидкодії запитів без Columnstore індексу.

Також були проведені експерименти по оптимізації КД та

пришвидшенню запитів за допомогою розроблених методів та алгоритмів. Було знайдено випадки, коли існуюча компресія не дуже вдало зменшила загальну вагу вектору. Отже було виявлено, що розроблений алгоритм компресії був би краще в цих випадках.

Був проведений експеримент з використанням у запиті агрегаційних функцій, де відбувалась повна матеріалізація даних перед тим, як виконати агрегації. Цю ситуацію можна покращити розробленим алгоритмом для пришвидшення виконання запитів з використанням агрегаційних функцій.

ВИСНОВКИ

Було досліджено оптимальніший підхід до вибору системи та реалізації її механізму для великих баз даних, які використовують зазвичай для бізнес-аналітики. Ці задачі полягають у отриманні зазвичай агрегаційних даних з виборкою по вимірам (діменшинам). Для звичайних баз даних, які мають архітектуру зберігання даних як у звичайних РБД, ці задачі будуть дуже важкими та неекономними в ресурсному плані. Тому було прийнято рішення використовувати іншу систему, таку як OLAP, яка перебудовує модель даних таким чином, щоб було простіше виконувати аналітичні запити. Було виявлено три реалізації механізму OLAP, які мають свої переваги та недоліки. Але проаналізувавши всі три види, було обрано саме ROLAP, так як з використанням цього механізму з'являється можливість для оптимізації процесу функціонування БД та її економного зберігання. Також були розібрані інструменти SQL Server, які дозволяють добре оптимізувати БД. Таким чином, використовуючи систему OLAP з реалізацією механізму ROLAP, можна створити на великі таблиці columnstore індекси, для того, щоб дуже ефективно зменшити вагу самої БД та сильно підвищити продуктивність функціонування великих баз даних для аналітичних бізнес-задач.

Були розглянуті існуючі алгоритми компресії даних, які використовуються в різних випадках та для різних типів даних. Шляхом експериментів було знайдено випадки, коли існуючі алгоритми компресії не дуже вдало справлялись з зменшенням загального розміру вектору, то було запропоновано використання власно-розробленого алгоритму КД. Також, було виявлено, що при використанні агрегацій у запитах вектори у роугрпах повністю матеріалізуються та вже потім агрегуються. Тому було запропоновану використати власно-розроблений алгоритм для пришвидшення виконання запитів з використанням агрегаційних функцій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008: 2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. К.: ДП “УкрНДНЦ”. 2016. 30 с.

2. Дипломне проектування для студентів усіх форм навчання спеціальностей 151 «Автоматизація там комп’ютерно-інтегровані технології»/ упоряд. І.Ш. Невлюдов, А.О. Андрусевич, О.В. Токарєва, Г.В. Пономарьова. Київ, 2018. 320.

3. Методичні вказівки з підготовки та захисту кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності 151 Автоматизація та комп’ютерно-інтегровані технології, освітньо-професійних програм: «Автоматизоване управління технологічними процесами», «Комп’ютерно інтегровані технологічні процеси і виробництва», «Комп’ютеризовані та робототехнічні системи» / Упоряд. І. Ш. Невлюдов, Р. В. Артюх, В. В. Безкоровайний, Н. П. Демська, В. В. Євсєєв, О. І. Филипенко, О. М. Цимбал. Харків: ХНУРЕ, 2021. 55 с.

4. Методи оптимізації бази даних для забезпечення швидкого пошуку в цій базі даних / Кулик О. О., Трофимов Н. С., Невлюдова В. В. – Харків: Topical issues of modern science, society and education. Proceedings of the 6th International scientific and practical conference, 2021. 414 с.

5. Relational vs. NoSQL data. URL: https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data (дата звернення: 09.11.2022).

6. OLTP. URL: https://uk.wikipedia.org/wiki/OLTP (дата звернення: 09.11.2022).

7. Оперативна обробка транзакцій (OLTP). URL: https://learn.microsoft.com/ru-ru/azure/architecture/data-guide/relational-data/online-transaction-processing (дата звернення: 09.11.2022).

8. In-Memory OLTP overview and usage scenario. URL: https://learn.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/overview-and-usage-scenarios?view=sql-server-ver16 (дата звернення: 23.11.2022).

9. Analytics architecture design. URL: https://learn.microsoft.com/en-us/azure/architecture/solution-ideas/articles/analytics-start-here (дата звернення: 23.11.2022).

10. Огляд онлайнової аналітичної обробки (OLAP). URL: https://support.microsoft.com/uk-ua/office/огляд-онлайнової-аналітичної-обробки-olap-15d2cdde-f70b-4277-b009-ed732b75fdd6 (дата звернення: 10.11.2022).

11. Online analytical processing (OLAP). URL: https://learn.microsoft.com/en-us/azure/architecture/data-guide/relational-data/online-analytical-processing (дата звернення: 23.11.2022).

12. Top 10 questions and answers about SQL Server Indexes. URL: https://learn.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver16 (дата звернення: 09.11.2022).

13. Індекси Columnstore. Огляд. URL: https://learn.microsoft.com/ru-ru/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-ver16 (дата звернення: 10.11.2022).

14. Clustered and nonclustered indexes described. URL: https://learn.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver16 (дата звернення: 09.11.2022).

15. Columnstore indexes – Query performance. URL: https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-query-performance?view=sql-server-ver16 (дата звернення: 23.10.2022).

16. Columnstore indexes – Data Warehouse. URL: https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-data-warehouse?view=sql-server-ver16 (дата звернення: 11.11.2022).

17. Real-Time Operational Analytics Explained. URL: https://learn.microsoft.com/en-us/sql/relational-databases/indexes/get-started-with-columnstore-for-real-time-operational-analytics?view=sql-server-ver16 (дата звернення: 10.11.2022).

18. Big data options on the Microsoft SQL Server platform. URL: https://learn.microsoft.com/en-us/sql/big-data-cluster/big-data-options?view=sql-server-ver16 (дата звернення: 23.11.2022).

19. Understanding distributed NoSQL databases. URL: https://learn.microsoft.com/en-us/azure/cosmos-db/distributed-nosql?toc=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fazure%2Farchitecture%2Ftoc.json&bc=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fazure%2Farchitecture%2Fbread%2Ftoc.json (дата звернення: 23.11.2022).

20. What is Data Compression. URL: https://www.barracuda.com/support/glossary/data-compression (дата звернення: 24.11.2022).

21. Data compression. URL: https://en.wikipedia.org/wiki/Data_compression (дата звернення: 24.11.2022).

22. Aggregate Functions (Transact-SQL). URL: https://learn.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql?view=sql-server-ver16 (дата звернення: 24.11.2022).

23. Database Security Best Practices. URL: https://backendless.com/database-security-best-practices/ (дата звернення: 24.11.2022).