https://doi.org/10.31891/2219-9365-2022-71-3-10
УДК 004.65

ВЯЧЕСЛАВ БОЙКО
Хмельницький національний університет
e-mail: bviacheslav.12@gmail.com
ЮРІЙ ФОРКУН
Хмельницький національний університет
https://orcid.org/0000-0002-7906-4191
e-mail: forkun@ridne.net

УДОСКОНАЛЕННЯ МЕТОДУ МАТЕРІАЛІЗОВАНИХ ПРЕДСТАВЛЕНЬ У РЕІНЖИНІРИНГУ БАЗИ ДАНИХ

Розглянуто основні методи реінжинірингу бази даних у сучасних програмних системах. Виконано порівняння та аналіз найбільш популярних та окреслено основні етапи реінжинірингу сховищ. Удосконалено метод матеріалізованих представлень, який використовується на етапі редизайну табличної структури бази даних, що дозволяє застосовувати представлення як один із оптимізаційних компонентів на постійній основі, а отже, і значно підвищує продуктивність запитів вибірки даних у програмній системі.

Ключові слова: реінжиніринг, редизайн, декомпозиція, база даних, матеріалізоване представлення.

VIACHESLAV BOIKO, YURIY FORKUN
Khmelnitsky National University

IMPROVING THE METHOD OF MATERIALIZATION VIEWS AS A PART OF A DATABASE REENGINEERING PROCESS

Data warehouse – is the most important component in the modern cloud web-applications. It has a significant impact on performance of the program system. According to its influence on whole system productivity, incorrect database schemas, none-optimized queries or unsuitable data organization structure can greatly reduce the performance of whole application. With a constantly growing amount of information and brand-new customer requirements that possibly can not be applied to the current architecture, the reengineering process is a strong necessary to refactor and reform the system architecture including data storages. There are a variety of reengineering methods that could make the system faster, easier to maintain and more stable. But some of them are very cost-expensive due to transforming into a more complex formation that it was before, thus maintenance of such system can be not so easy. On the other hand, some of methods are not able to give enough results to fit customer performance requirements. Because of that, solution should use the known database features with rethinking of its usage as a part of the reengineering process. The potential mechanism of increasing the database performance can be the relational database objects – materialized views with their own refreshing strategies. Having a big read performance, materialized views can be very useful to accessing a large amount of data, but due to some refresh constraints they cannot be used permanently. So, in the paper the main methods of database reengineering in modern software systems are considered. A comparison and analysis of the most popular ones was performed, and the main stages of data storage reengineering were outlined. The method of materialized views, which is used at the stage of redesigning the tabular structure of the database, has been improved, which allows use the view as one of the optimization components on a permanent basis, thus significantly increases the performance of data selection queries in the software system.

Keywords: reengineering, redesign, decomposition, database, materialized view.

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

Більшість сучасних програмних систем мають проблеми зі швидкістю обробки даних через постійно зростаючий обсяг інформації. Нові бізнес-вимоги вимагають реалізацію нових функціональних можливостей, які часто розробляються із використанням застарілих шаблонів та структур. Інформаційне сховище є основним артефактом впливу на продуктивність програмного забезпечення, оскільки характер та структура організації інформації є найвагомішими чинниками, що впливають на швидкість обробки даних. Для підвищення показників продуктивності застосовують різні методи реінжинірингу сховища даних, більшість з яких хоч і вирішують проблеми продуктивності, проте підтримка та супровід таких систем є дороговартісним та складним процесом. А тому аналіз та удосконалення існуючих методів реінжинірингу бази даних є актуальним насьогодні.

Зазвичай реінжиніринг проводиться поступово у декілька етапів: редизайну, декомпозиції та міграції. Проте, якщо необхідні показники приросту продуктивності та фінансових витрат уже досягнуті, процес реінжинірингу вважається завершеним на конкретному етапі лише тоді, якщо при подальших кроках показники погіршуватимуться або залишатимуться незмінними.

Крім того, процес редизайну вважається обов'язковим етапом. При цьому не завжди дає змогу отримати бажані результати. Варто зазначити, що за основу реінжинірингу взято сховище реляційного типу, а тому усі описані методи можуть застосовуватись лише до такого типу баз даних.

Аналіз досліджень та публікацій

На етапі редизайну виділяють декілька часто використовуваних методів, що дозволяють підвищити продуктивність роботи з базою даних. У статті [1] описані основні методи оптимізації запитів. Крім того, представлено метод денормалізації лічильників. Перевагою є те, що вибірка вже попередньо обрахованих значень дозволяє уникнути затратних запитів для підрахунку записів. Проте недоліком є неможливість застосування його до динамічного текстового пошуку.

Перебудова індексів також є одним із основних методів редизайну бази даних. У публікації [2] запропоновано вирішення проблеми автономної роботи із фрагментованими індексами в реляційній СУБД шляхом відстеження поточного стану індексів і, якщо вони стають фрагментованими, автоматично перебудовувати їх. Перевагами методу є те, що заходи із перебудови індексів, які проводяться на постійній основі дозволяють прискорити операції у базах даних.

Матеріалізація як метод реінжинірингу може значно скоротити час обробки запитів, особливо для агрегаційних запитів до великих таблиць, оскільки матеріалізовані представлення зберігають результат вихідного запиту, що є перевагою. У статті [3] описано використання представлення для оптимізації вибірки. Проте, недоліком є те, що застосування матеріалізованих представлень на постійній основі може призвести до проблем з продуктивністю, оскільки їх оновлення займає багато часу, а також у деяких системах керування базами даних відбувається блокування таблиць.

Отже, етап редизайну передбачає оптимізаційні зміни лише у рамках одного сховища даних. Але іноді внесених змін виявляється недостатньо для досягнення бажаних показників продуктивності. Це може відбутися за умови існування не властивої поточній базі структури організації даних, або відбувалася поступова та непомітна трансформація до такої структури шляхом внесення змін відповідно до вимог замовника. Для цього існують два методи реінжинірингу. Один із них – міграція даних. Хоча цей процес може бути громіздким і трудомістким, вважається, що переваги значно переважують кінцеві витрати [4]. Міграція також корисна, якщо на етапі редизайну не вдалося досягти бажаних показників. У публікації [5] наведено приклад методу реінжинірингу шляхом міграції даних з реляційного сховища до графової бази даних. Перевагами такого підходу є підвищення продуктивності, а недоліком – ускладнення підтримки та обслуговування нової СКБД, оскільки нова технологія вимагає специфічних навичок для якісної роботи.

Однак існують випадки, коли у системі дані різнорідної структури зберігаються у сховищі одного типу і виникає необхідність декомпозиції бази даних на декілька гетерогенних сховищ та подальшої міграції. Часто гетерогенні данні пов’язані між собою, а тому потребують шару абстракції для комунікації між собою. Це може бути зв’язувальне програмне забезпечення, що складається із обгортки, яка у свою чергу виконує функції формування запитів до гетерогенних сховищ. У статті [6] описано метод використання різнорідних баз даних у рамках реінжинірингу та дозволяє виконувати вибірку із декількох сховищ. Проте, підтримка таких систем може виявитись не простим завданням, оскільки масштабування сховищ даних до нових типів вимагає значних витрат та ресурсів.

Отже, найскладнішим етапом реінжинірингу є декомпозиція інформаційного сховища та міграція даних. З іншого боку, досить корисним методом редизайну є застосування матеріалізації у базах даних, що означає зберігання копії результату виконання певного запиту довільної складності. Матеріалізований запит будь-якої алгоритмічної складності при доступі до представлення матиме складність, що рівна складності простої вибірки. Це $O(1)$, якщо застосовано hash-індекс, $O(\log(n))$, якщо використано – b-tree індекс, або $O(n)$ – повне сканування за відсутності індексів. Як наслідок, зменшується навантаження на процесор і диск [7]. Проте недоліком матеріалізованих представлень є їх статичність, тобто, сама їх концепція полягає у тому, щоб зберігати великі набори даних уже у тому вигляді, у якому необхідно їх отримати. При синхронізації представлень та вихідної таблиці у деяких СКБД відбувається блокування таблиці, а час блокування залежить від тривалості оновлення представлення, що в свою чергу залежить від складності запити та об’єму даних у таблиці. Саме тому їх зазвичай використовують для отримання даних із рідко оновлюваних таблиць, а отже, потреба у синхронізації виникає лише один або декілька разів у певний проміжок часу [8].

Одним з методів збільшення швидкості оновлення матеріалізованих представлень є метод інкрементного оновлення. При перезаписі враховуються лише зміни у вихідних таблицях. Часто для цього використовуються різні структури даних, як, наприклад, LSM-дерево, яке є добре відомою структурою, прийнятою для покращення продуктивності запису [9]. Перевагами інкрементного оновлення є краща продуктивність перезапису матеріалізованих представлень за рахунок вибіркового застосування змін. Проте, не усі реляційні системи підтримують такий підхід. Крім того, процес оновлення матеріалізованих представлень блокує таблиці для запису, що може негативно вплинути на швидкість вставки та оновлення.

Метод відкладеного оновлення ґрунтується на перезаписі представлень один раз у деякий проміжок часу. Крім того, основним завданням є підтримка актуальності при вибірці, а це означає введення додаткових сховищ або таблиць у систему, що відповідатимуть за кешування вхідних даних. Але більшість авторів, які пропонують різні методи підтримки представлень, не наводять конкретних реалізацій [10]. Багато реляційних СКБД на сьогодні досі не реалізують ні один із наявних методів підтримки представлень.

Слід зазначити, що потенціал матеріалізованих представлень може бути ефективно використаний при роботі із гетерогенними даними. Таким чином проведення подальшого етапу реінжинірингу шляхом декомпозиції або міграції неоднорідної інформації виявиться надлишковим, а матеріалізовані представлення потенційно можуть вирішити проблему продуктивності на рівні редизайну сховища.

Формулювання цілей статті

Метою роботи є удосконалення методу матеріалізованих представлень, що дасть змогу використовувати усі переваги матеріалізації даних у комбінації із реалізованим на основі принципів кешування, алгоритмом синхронізації змін, а отже, підвищить показники продуктивності програмної системи та значно знизить вартість підтримки та супроводу апаратного та програмного забезпечення за рахунок досягнення ефективних результатів на етапі редизайну схеми та відмови від проведення декомпозиції сховища даних, як наступного етапу реінжинірингу бази.

Виклад основного матеріалу

Тому запропоновано удосконалення методу матеріалізованих представлень за рахунок застосування принципів стратегій кешування Write Back та Read Though у проектуванні табличної структури. Це дозволить використовувати матеріалізовані представлення на постійній основі, а також підвищить продуктивність програмної системи. За допомогою такого підходу реінжиніринг бази даних можна завершити на етапі редизайну. Для реалізації методу необхідно спроектувати спеціальну табличну структуру. Спрощена схема подана на рис. 1.

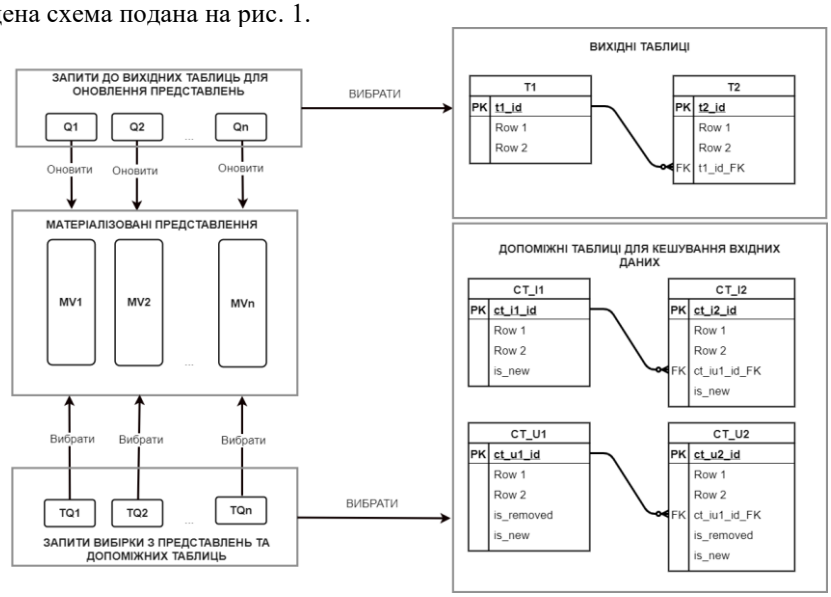

Рис. 1. Загальна схема табличної структури з використанням допоміжних таблиць та матеріалізованих представлень

Згідно рис. 1, таблиці T1 та T2 є вихідними. CT_I1 та CT_I2 – є копіями вихідних, що зберігатимуть дані для вставки, а CT_U1 та CT_U2 – для оновлення даних. Допоміжні таблиці кешуватимуть зміни та при синхронізації застосовуватимуть їх до вихідних таблиць.

MV1, MV2, …, MVn – матеріалізовані представлення, що зберігають результат виконання запитів Q1, Q2, …, Qn відповідно. Вибірка даних буде відбуватись з урахуванням допоміжних таблиць шляхом їх об'єднання з представленням. Дані, що помічені як is_removed не будуть враховуватись у вибірці з представлень, оскільки вони при синхронізації мають бути видалені з вихідних таблиць. У разі оновлення, записи із відповідним ідентифікатором у мат. представленнях не враховуватимуться при вибірці, оскільки в такому випадку поступає нова інформація, що зумовлює використання даних лише із допоміжних таблиць. Для синхронізації змін вибиратиметься проміжок часу, протягом якого необхідно здійснити оновлення. Найчастіше встановлюється час найменшого навантаження на сервер. Алгоритм подано на рисунку 2.

Асимптотична складність вибірки з представлення завжди рівна складності звичайної вибірки. Нехай, є дві вихідні таблиці із кількістю даних N_1 та N_2 відповідно. При цьому, нехай, вважатимемо, що у вихідних таблицях існує досить велика кількість записів, а тому $N_1 \gg 100, N_2 \gg 100$.

Зазвичай для прискорення вибірки із таблиць з великою кількістю даних встановлюють індекси на зовнішні ключі, що дозволяє виконувати вибірку за лінійний час, уникаючи вкладених запитів.

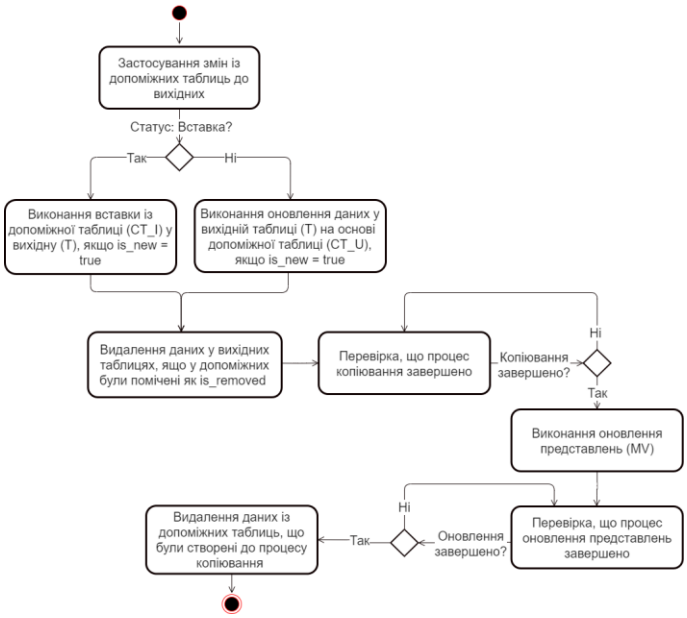

Рис. 2. Алгоритм синхронізації допоміжних та вихідних таблиць

Асимптотична складність алгоритму вибірки у кращому випадку при застосуванні техніки Sort-Merge за умови унікальності даних у двох таблицях становитиме:

$$O(N_1 + N_2), \tag{1}$$

де O – нотація Ландау;

N_1, N_2 – обсяг першої та другої вихідної таблиці відповідно [11].

Нехай у систему введено матеріалізовані представлення із кількістю даних M та тимчасові таблиці для вставки із обсягами I_1 та I_2. При цьому згідно алгоритму синхронізації, дані із цих таблиць періодично видаляються, а отже, максимальна кількість даних буде значно меншою за кількість у вихідних таблицях, тобто:

$I_1 \ll N_1$, $I_2 \ll N_2$, $N \gg 100$. При цьому, згідно з публікацією [12], асимптотична складність вибірки у кращому випадку при використанні b-tree індексу становитиме $O(log_2(M))$. Для вибірки даних об'єднуватимемо матеріалізоване представлення і допоміжні таблиці. Загальна асимптотична складність алгоритму у кращому випадку, взявши до уваги (1), становитиме:

$$B = O(log_2(M) + I_1 + I_2), \tag{2}$$

де B – асимптотична складність алгоритму вибірки із представлення та допоміжних таблиць для вставки;

M – кількість записів у матеріалізованому представленні;

I_1, I_2 – кількість записів у першій та другій допоміжній таблиці для вставки відповідно.

Проте існують випадки, коли записи у одну із таблиць не виконуються, а використовуються існуючі ключі. Може виникнути ситуація, при якій нові дані записані у одну з таблиць, але зовнішній ключ вказує на уже існуючі дані, а отже, у іншій зв'язаній допоміжній таблиці нового запису не буде і доведеться виконувати вибірку із вихідної таблиці. Час виконання зросте, проте, для того, щоб уникнути подальших звернень до вихідних таблиць, дані, що вперше отримані із неї необхідно кешувати шляхом перенесення у допоміжну таблицю і подальші звернення використовуватимуть дані у ній. При цьому такі записи необхідно помітити, щоб уникнути їх дублювання при перенесенні у вихідну таблицю (стовбець is_new на рис. 1).

Додамо таблиці, що будуть містити зміни, які у вихідних таблицях необхідно застосувати для оновлення записів: U_1 та U_2. Оскільки при вибірці необхідно враховувати лише дані з таблиць для оновлення і не враховувати дані на видалення, асимптотична складність алгоритму збільшиться, проте це не суттєво для невеликих об'ємів даних, оскільки, $U_1 \ll N_1$ та $U_2 \ll N_2$. Взявши до уваги B з (2), отримаємо таку складність алгоритму у кращому випадку:

$$B + O(U_1 + U_2 + X), \tag{3}$$

де U_1, U_2 – кількість записів у першій та другій допоміжній таблиці для оновлення відповідно;
X – додаткова складність, яка може виникнути при відсіюванні застарілих або видалених даних.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Загалом було проаналізовано основні методи та етапи реінжинірингу бази даних, встановлено їх переваги та недоліки, детально досліджено матеріалізацію даних, як один із методів реінжинірингу інформаційного сховища та спосіб збільшення показників продуктивності бази даних. Запропоноване удосконалення методу матеріалізованих представлень дозволяє підвищити продуктивність виконання запитів вибірки будь-якої складності. Подальші дослідження спрямовані на підвищення продуктивності вибірки даних із допоміжних таблиць шляхом удосконалення методів фільтрації застарілих та видалених даних, а також удосконалення процесів синхронізації даних між допоміжними та вихідними таблицями.

Література

1. N. Kumari. SQL Server Query Optimization Techniques - Tips for Writing Efficient and Faster Queries / Kumari N. // International Journal of Scientific and Research Publications. – Punjab, India, June 2012. – P. 1.

2. E. Morelli. Autonomous Re-indexing / Morelli E., Almeida A., Lifschitz S., Maria Monteiro J., Machado J. // Proceedings of the 27th Annual ACM Symposium on Applied Computing. – USA, March 2012. – P. 893–896.

3. J. Goldstein. Optimizing Queries Using Materialized Views: A practical, scalable solution / Goldstein J., Larson P.-Å. // Proceedings of the 2001 ACM SIGMOD international conference on Management of data. – June 2001. – DOI: 10.1145/375663.375706.

4. F. Oladipo. Re-engineering legacy data migration methodologies in critical sensitive systems / Oladipo F., Raiyetumbi J. // Journal of Global Research in Computer Science . – Kogi, Nigeria, November 2015.

5. Y. Unal. Migration of Data from Relational Database to Graph Database / Unal Y., Oguztuzun H. // International Conference on Information and Software Technologies. – Istanbul, Turkey, March 2018. – DOI: 10.1145/3200842.3200852.

5. H. Zhang. Unified SQL Query Middleware for Heterogeneous Databases / Zhang H. // Journal of Physics: Conference Series. – Shanghai, China, 2021.

6. A. Gupta. Problems, Techniques, and Applications / Gupta A., Mumick I. // IEEE Data Eng. Bull. – P. 1.

7. V. Kumar. Materialized View Selection Using Iterative Improvement / Kumar V., Kumar S. // Advances in Intelligent Systems and Computing. – New Delhi, India, 2013. – DOI: 10.1007/978-3-319-91458-9_42

8. H. Duan. Incremental Materialized View Maintenance on Distributed Log-Structured Merge-Tree / Duan H., Hu H., Qian W., Ma H., Wang X., Zhou A. // Database Systems for Advanced Applications. – May 2018.

9. A. Gosaina. Architecture Based Materialized View Evolution: A Review / Gosaina A., Sabharwal S., Gupta R. // Procedia Computer Science. – 2015. – P. 257. – DOI: 10.1016/j.procs.2015.04.179.

10. M.-C. Albutiu. Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems / Albutiu M.-C., Kemper A., Neuman T. // Proceedings of the VLDB Endowment. – Germany, June 2012.

11. D. Comer. Computing Surveys: The Ubiquitous B-Tree / Comer D. // ACM Computing Surveys. – West Lafayette, Indiana, USA, June 1979. – P. 127.

References

1. N. Kumari. SQL Server Query Optimization Techniques - Tips for Writing Efficient and Faster Queries / Kumari N. // International Journal of Scientific and Research Publications. – Punjab, India, June 2012. – P. 1.

2. E. Morelli. Autonomous Re-indexing / Morelli E., Almeida A., Lifschitz S., Maria Monteiro J., Machado J. // Proceedings of the 27th Annual ACM Symposium on Applied Computing. – USA, March 2012. – P. 893–896.

3. J. Goldstein. Optimizing Queries Using Materialized Views: A practical, scalable solution / Goldstein J., Larson P.-Å. // Proceedings of the 2001 ACM SIGMOD international conference on Management of data. – June 2001. – DOI: 10.1145/375663.375706.

4. F. Oladipo. Re-engineering legacy data migration methodologies in critical sensitive systems / Oladipo F., Raiyetumbi J. // Journal of Global Research in Computer Science . – Kogi, Nigeria, November 2015.

5. Y. Unal. Migration of Data from Relational Database to Graph Database / Unal Y., Oguztuzun H. // International Conference on Information and Software Technologies. – Istanbul, Turkey, March 2018. – DOI: 10.1145/3200842.3200852.

6. H. Zhang. Unified SQL Query Middleware for Heterogeneous Databases / Zhang H. // Journal of Physics: Conference Series. – Shanghai, China, 2021.

7. A. Gupta. Problems, Techniques, and Applications / Gupta A., Mumick I. // IEEE Data Eng. Bull. – P. 1.

8. V. Kumar. Materialized View Selection Using Iterative Improvement / Kumar V., Kumar S. // Advances in Intelligent Systems and Computing. – New Delhi, India, 2013. – DOI: 10.1007/978-3-319-91458-9_42

9. H. Duan. Incremental Materialized View Maintenance on Distributed Log-Structured Merge-Tree / Duan H., Hu H., Qian W., Ma H., Wang X., Zhou A. // Database Systems for Advanced Applications. – May 2018.

10. A. Gosaina. Architecture Based Materialized View Evolution: A Review / Gosaina A., Sabharwal S., Gupta R. // Procedia Computer Science. – 2015. – P. 257. – DOI: 10.1016/j.procs.2015.04.179.

11. M.-C. Albutiu. Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems / Albutiu M.-C., Kemper A., Neuman T. // Proceedings of the VLDB Endowment. – Germany, June 2012.

12. D. Comer. Computing Surveys: The Ubiquitous B-Tree / Comer D. // ACM Computing Surveys. – West Lafayette, Indiana, USA, June 1979. – P. 127.