УДК 528.852
DOI https://doi.org/10.32782/2663-5941/2025.3.2/14

Бойко О.В.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Ульяницька К.О.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Коломієць У.Д.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

ДО ПИТАННЯ ОПТИМІЗАЦІЇ ЗАПИТІВ РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ ДЛЯ ВИРІШЕННЯ АНАЛІТИЧНИХ ЗАВДАНЬ

У роботі проаналізовано методи оптимізації запитів до реляційної бази даних з метою підвищення їхньої продуктивності під час вирішення повсякденних завдань аналітиками в умовах постійно зростаючих обсягів інформації для подальшого аналітичного аналізу.

Було визначено, що хоча сучасні оптимізатори запитів для реляційних баз даних мають значний потенціал і надають багато корисних функцій, вони можуть не відповідати очікуванням аналітиків. Оптимізатори запитів мають обмежені можливості для ефективної оптимізації складних або дуже складних запитів, які зазвичай використовуються в роботі аналітиків. При наявності великої кількості таблиць, складних умов, підзапитів та функцій можуть виникнути труднощі під час оптимізації запитів. Тобто, оптимізатори не можуть автоматично адаптуватися до змін навантаження або до конкретних умов виконання запиту. Оптимізатори часто зосереджуються на оптимізації самого запиту, але не забезпечують глибокого аналізу структури даних.

Крім того, наведено практичні рекомендації щодо ручної оптимізації запитів, тобто переписування існуючого запиту з урахуванням поточної ситуації на основі глибокого розуміння особливостей структурованої мови запитів, що, як правило, залишається поза досвідом звичайного аналітика та які не враховуються існуючими оптимізаторами. Отже, створення нового оптимізатора може бути виправдане необхідністю врахування цих недоліків та забезпечення оптимального та адаптивного підходу до оптимізації запитів. Новий оптимізатор може зосередити свої ресурси на забезпеченні ефективного реагування на зміни в даних, використанні обчислювальних ресурсів для генерації швидких та оптимальних планів виконання, а також максимальному використанні паралельного та розподіленого виконання для забезпечення високої продуктивності в різноманітних середовищах даних.

Ключові слова: *оптимізація запитів, реляційна база даних, структурована мова запитів, аналітик, аналітична робота.*

Постановка проблеми. Відомо [1], що однієї з основних умов прийняття єдино правильного, ефективного в умовах непередбаченості і кризових явищ управлінського рішення є забезпечення особи, яка приймає рішення (управлінця), необхідною і достатньою кількістю аналітичної інформації. Таким чином, інформаційно-аналітична діяльність певною мірою убезпечує, захищає керівників, управлінців від ризиків, небезпек і викликів сьогодення, рекомендуючи те чи інше ефективне управлінське рішення, прогнозуючи наперед наслідки його прийняття чи неприйняття, чи бездіяльності. На думку вчених [1], інформаційно-аналітична діяльність – це сукупність дій на основі концепцій, методів, засобів, нормативно-методичних матеріалів для збору, накопичення, обробки та аналізу даних з метою обґрунтування та прийняття рішень. Однак в останні роки спостерігається тенденція лавинообразного збільшення обсягів інформації, яку аналітику треба своєчасно опрацювати та проаналізувати. Відомо, що з кожним роком значно зростає кількість оброблюваних цифрових даних у світі. За 13 років цей показник зріс у 60 разів, починаючи з 2 зетабайтів у 2010 році. Так очікується [2], що 120 зетабайт, згенерованих у 2023 році, у 2025 році зростуть більш ніж на 150% і досягнуть 181 зетабайт.

Для зберігання, зміни та обробки взаємозалежної інформації, переважно великих обсягів, застосовуються бази даних, зокрема реляційні, запити на аналітику до яких стають більш складними, а користувачі очікують оперативних відповідей на запити у реальному часі. На сьогодні однієї з найпопулярніших мов програмування щодо оперування даними, що зберігаються в реляційній базі даних, є SQL [3] та 78% професійних аналітиків даних регулярно використовують його у своїй повсякденній роботі [3]. Тому оптимізація складних запитів до реляційних баз даних (РБД) може допомогти зменшити час та ресурси, витрачені на аналітичну обробку, та стає ключовим чинником для забезпечення швидкості обробки даних, ефективного використання ресурсів та підтримання конкурентоспроможності в сучасному бізнес-середовищі.

Аналіз останніх досліджень і публікацій. Теоретичні засади оптимізації запитів до баз даних з метою підвищення продуктивності баз даних висвітлено у [4–7]. В дослідженнях А. Бари, І. Лунгу та інших [4–5] підкреслюють важливість аналізу вартості виконання SQL запитів у базах даних Oracle. Роботи Белоуса Р., Крилова Є., Анікіна В. [7] присвячено підходам до оптимізації продуктивності запитів у розподілених базах даних, для зниження витрат на опрацювання множинних запитів. Ці роботи демонструють важливість розвитку в галузі методів оптимізації баз даних, з врахуванням особливостей збільшення обсягів даних, однак питання оптимізації складних запитів, які сьогодні здебільшого використовуються у аналітичної діяльності залишаються поза увагою цих науковців.

Постановка завдання. Метою статті є аналіз методів та можливостей сучасних оптимізаторів запитів до реляційної бази даних з метою підвищення продуктивності під час вирішення аналітиками повсякденних завдань в умовах сьогодення, а саме постійного збільшення обсягів інформації для аналітичного аналізу.

Виклад основного матеріалу. Згідно з дослідженням McKinsey & Company, проведеним у 2024 році, компанії, які ефективно використовують передову аналітику даних, в середньому мають на 23% вищий приріст доходу, ніж їхні конкуренти. Автоматизація рутинних аналітичних завдань може заощадити аналітикам до 40% їхнього часу, звільняючи їх для зосередження на більш стратегічних завданнях [3]. Саме тому оптимізація запитів SQL є важливою навичкою для аналітиків, оскільки вона може значно підвищити ефективність використання баз даних під час інформаційно-аналітичної діяльності. Тому однією з головних вимог, яким сьогодні має відповідати аналітик, є навички написання оптимальних запитів до РБД, якими не завжди володіють відповідні фахівці з аналітичної діяльності.

В той же час сучасні РСУБД в своєму складі мають оптимізатори. Оптимізатор SQL – це частина механізму баз даних, призначена для створення найефективнішого плану виконання SQL-запитів. Тобто це алгоритм, який аналізує запит і вибирає найефективніший спосіб його виконання, наприклад шляхом вибору відповідних індексів, способу об'єднання таблиць або порядку операцій. Як правило, оптимізатор аналізує SQL-запит, враховуючи його структуру, використані оператори та доступні індекси. Після цього оптимізатор генерує різні плани виконання, тобто послідовності операцій, які можна використовувати для виконання запиту. Далі здійснюється оцінка продуктивності кожного згенерованого плану (на основі статистики бази даних, часу виконання, споживання ресурсів) та вибір найефективнішого. Вибраний план виконання потім використовується для виконання запиту. Оптимізатори SQL призначені, як правило, для адміністраторів баз даних та розробників, які створюють та підтримують запити. Розуміючи, як працює оптимізатор, користувачі, до яких можна віднести аналітиків, можуть оптимізувати свої запити, покращуючи продуктивність використання програми.

Розглянемо оптимізатори, що використовують популярні РБД.

1. SQL Server Query Optimizer – це оптимізатор, що працює на основі витрат. Кожен можливий план виконання має пов'язані з ним витрати з точки зору обсягу використаних обчислювальних ресурсів. Оптимізатор запитів повинен проаналізувати можливі плани та вибрати той, що має найнижчу розрахункову вартість. Оптимізатор спирається на статистику розподілу, коли оцінює витрати ресурсів різних методів для вилучення інформації з таблиці або індексу. Статистика розподілу зберігається для стовпців та індексів і містить інформацію про щільність базових даних. Щільність визначає розподіл унікальних значень, що існують у даних, або середню кількість дублікатів значень для заданого стовпця. Зі зменшенням щільності селективність значення зростає.

2. Oracle SQL Optimizer – інструмент розроблений для оптимізації запитів до баз даних Oracle. Його головна мета – мінімізувати витрати (час), необхідні для отримання результатів запиту. Оптимізатор працює на основі багатьох факторів, до яких можна віднести.

Статистика бази даних. Оптимізатор використовує статистику, зібрану з бази даних, таку як розподіл значень у стовпцях, кількість рядків у таблицях та розмір блоків даних. Якщо статистика застаріла або неточна, оптимізатор може приймати неоптимальні рішення.

Структура запиту. Складні запити з багатьма реченнями JOIN, підзапитами або функціями може бути складніше оптимізувати. Структура запиту та оператори, що використовуються в ньому, впливають на вибір плану оптимізатором.

Доступність індексів. Індекси є ключовим інструментом для прискорення запитів, що дозволяє оптимизатору швидко отримувати доступ до необхідних даних. Відсутність належних індексів або їх неправильне використання може призвести до уповільнення виконання запитів.

Параметри конфігурації:. Параметри конфігурації бази даних, такі як розмір буферної пам'яті та параметри паралельної обробки, можуть впливати на продуктивність запитів.

Історія виконання запитів. Оптимізатор може використовувати інформацію про попередні виконання запитів, щоб краще адаптувати плани до поточних потреб.

3. PostgreSQL Query Optimizer – оптимізатор запитів для PostgreSQL. Він використовує генетичні алгоритми та здійснює планування запитів, використовуючи евристичний пошук. Це дозволяє скоротити час планування для складних запитів (у яких з'єднуються багато відносин), за рахунок того, що іноді отримані плани поступаються за якістю планам, що вибираються при переборі. Тип оптимізатора – вартісний. Особливості: ліцензійна плата за використання; складний у використанні для початківців; вимагає налаштування та підтримки спеціаліста, який має глибокі знання у даній сфері, якими, нажаль володіють не всі аналітики.

Для оцінки ефективності оптимізації запиту відповідними оптимизаторами застосовують наступні показники.

Час виконання запиту – загальний час, який потрібно для виконання запиту до бази даних. Чим менше час виконання, тим ефективніше запит.

План виконання запиту – візуалізація запропонованого після оптимізації плану виконання запиту, тобто порядку виконання операцій, використання індексів, методів з'єднання таблиць тощо.

Кількість прочитаних/записаних рядків – інформація про кількість прочитаних або записаних рядків під час виконання запиту. Менша кількість операцій зчитування/запису може свідчити про більш ефективне виконання запиту.

Використання індексів – які індекси були використані під час виконання запиту. Використання індексів може покращити продуктивність запиту.

Обсяг пам'яті та ресурсів. Це може допомогти виявити проблеми з використанням ресурсів та встановити оптимальні значення.

Статистика запиту. Це може допомогти виявити слабкі місця та виконати точкову оптимізацію, тощо

Проведемо експеримент, чи здатна система, де застосовується існуючий оптимізатор, оптимізувати запит, який використовують аналітики у реальній повсякденної діяльності так, як це виконує людина. Усі дослідження проводилися в тому ж тестовому середовищі, тобто на комп’ютері (операційна система Windows 10 Home 64bit, процесор AMD Ryzen 5 2600, Ram 16 GB 3000MHz CL15, SSD Samsung 970 1TB). Щоб забезпечити точніші вимірювання на момент дослідження на комп’ютері працювали лише операційна система, тестова програма, та відповідна РСУБД. Комп’ютер було відключено від мережі, а решта програм, особливо антивірусне програмне забезпечення і брандмауер було вимкнено. Для дослідження було використано базу даних кол-центру, яка містить 10 млн. записів. Треба визначити кількість дзвінків від клієнтів за місяць. Було створено два різні запити (запит 1, запит 2) в результаті яких було розв’язано одне й теж саме завдання та отримано одну й ту ж інформацію, але під час написання запитів використовувалися різні оператори та логіка написання у зв’язку з різним практичним досвідом аналітиків у створенні запитів (рис. 1, рис. 2). Причому запит 2 написано більш досвідченим фахівцем у написанні запитів SQL. Також визначено план виконання запиту 1 та запиту 2 за допомогою оптимізатора PostgreSQL Query Optimizer (рис. 3, рис. 4).

```
explain(buffers,analyze)
select count(*), min_login_date::date
from
(
  select min(login_date) min_login_date
  from users u
    left join account a on u.id = a.user_id
    left join account_log al on al.account_id = a.id
  group by u.id
) t
where min_login_date between '2023-04-01' and '2023-04-30'
group by min_login_date::date
order by min_login_date::date
```

Рис. 1. Запит 1 для розрахунку кількості клієнтів за місяць

postgres/postgres@PostgreSQL 11

Query Editor | Query History

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
15	-> Merge Left Join (cost=0.99..550060.76 rows=10000020 width=16) (actual time=0.032..5334.003 rows=10048024 loops=1)
16	Merge Cond: (u.id = a.user_id)
17	Buffers: shared hit=970813 read=223089 written=428
18	-> Index Only Scan using users_pkey on users u (cost=0.43..423617.73 rows=10000020 width=8) (actual time=0.013..2697.031 rows=10000000 loops=1)
19	Heap Fetches: 10000000
20	Buffers: shared hit=4 read=191251 written=370
21	-> Materialize (cost=0.55..88942.97 rows=1000000 width=16) (actual time=0.016..1485.136 rows=1000000 loops=1)
22	Buffers: shared hit=970809 read=31838 written=58
23	-> Nested Loop Left Join (cost=0.55..86442.97 rows=1000000 width=16) (actual time=0.012..1370.956 rows=1000000 loops=1)
24	Join Filter: (al.account_id = a.id)
25	Buffers: shared hit=970809 read=31838 written=58
26	-> Index Scan using ix__account__user_id on account a (cost=0.42..71434.83 rows=1000000 width=16) (actual time=0.006..1201.071 rows=1000000...
27	Buffers: shared hit=970808 read=31838 written=58
28	-> Materialize (cost=0.13..8.14 rows=1 width=16) (actual time=0.000..0.000 rows=0 loops=1000000)
29	Buffers: shared hit=1
30	-> Index Scan using ix__account_log__account_id on account_log al (cost=0.13..8.14 rows=1 width=16) (actual time=0.002..0.002 rows=0 loops=1)
31	Buffers: shared hit=1
32	Planning Time: 0.622 ms
33	Execution Time: 7168.892 ms

Рис. 2. План виконання запиту 1

postgres/postgres@PostgreSQL 11

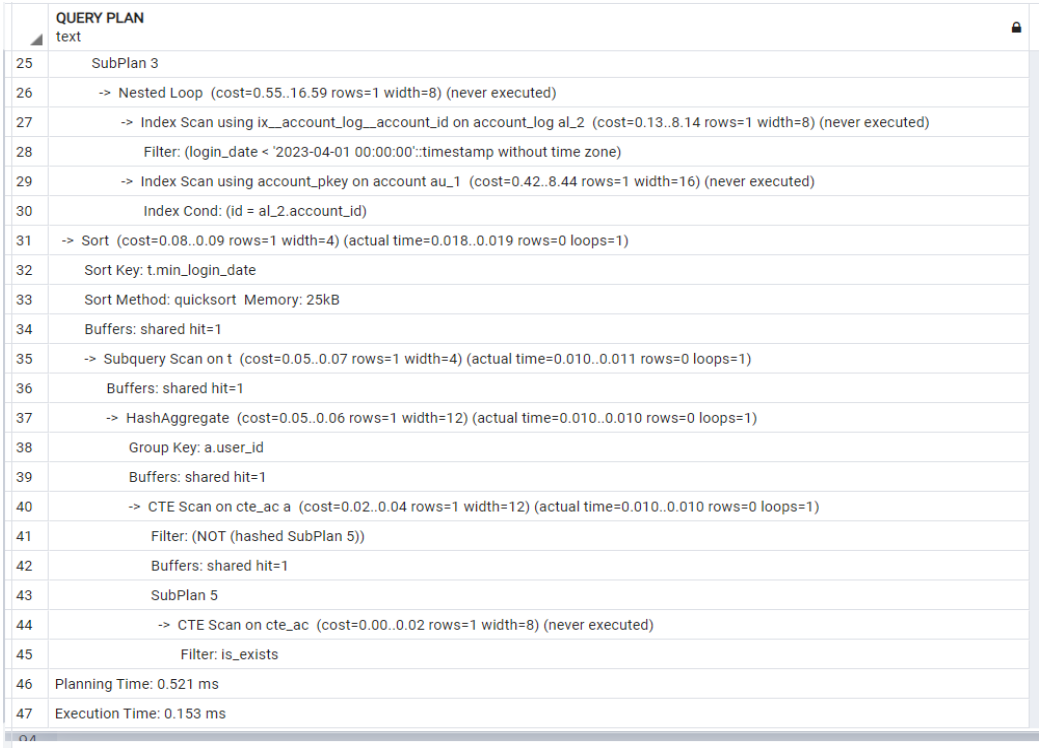
Query Editor | Query History

```
with cte
as
(
 select min(login_date :: date) min_login_date, al.account_id
  from  account_log al
  where al.login_date between '2023-04-01' and '2023-04-30'
  group by al.account_id
),
cte_ac
as
(
  select min_login_date, a.user_id,
           exists(select * from account_log al
           join account au on au.id = al.account_id where au.user_id = a.user_id and al.login_date < '2023-04-01' ) is_exists
  from cte
    join account a on a. id = cte.account_id
)
select count(*), min_login_date
from
(
  select a.user_id, min(min_login_date) min_login_date
  from cte_ac a
  where a.user_id not in (select user_id from cte_ac where is_exists)
  group by a.user_id
) t
group by min_login_date
order by min_login_date
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	GroupAggregate (cost=33.30..33.32 rows=1 width=12) (actual time=0.009..0.011 rows=0

Рис. 3. Запит 2 для розрахунку кількості клієнтів за місяць



	QUERY PLAN text
25	SubPlan 3
26	-> Nested Loop (cost=0.55..16.59 rows=1 width=8) (never executed)
27	-> Index Scan using ix__account_log__account_id on account_log al_2 (cost=0.13..8.14 rows=1 width=8) (never executed)
28	Filter: (login_date < '2023-04-01 00:00:00'::timestamp without time zone)
29	-> Index Scan using account_pkey on account au_1 (cost=0.42..8.44 rows=1 width=16) (never executed)
30	Index Cond: (id = al_2.account_id)
31	-> Sort (cost=0.08..0.09 rows=1 width=4) (actual time=0.018..0.019 rows=0 loops=1)
32	Sort Key: t.min_login_date
33	Sort Method: quicksort Memory: 25kB
34	Buffers: shared hit=1
35	-> Subquery Scan on t (cost=0.05..0.07 rows=1 width=4) (actual time=0.010..0.011 rows=0 loops=1)
36	Buffers: shared hit=1
37	-> HashAggregate (cost=0.05..0.06 rows=1 width=12) (actual time=0.010..0.010 rows=0 loops=1)
38	Group Key: a.user_id
39	Buffers: shared hit=1
40	-> CTE Scan on cte_ac a (cost=0.02..0.04 rows=1 width=12) (actual time=0.010..0.010 rows=0 loops=1)
41	Filter: (NOT (hashed SubPlan 5))
42	Buffers: shared hit=1
43	SubPlan 5
44	-> CTE Scan on cte_ac (cost=0.00..0.02 rows=1 width=8) (never executed)
45	Filter: is_exists
46	Planning Time: 0.521 ms
47	Execution Time: 0.153 ms

Рис. 4. План виконання запиту 2

Відповідно у запиті 1 Planning Time = 0.6 мс та Execution Time = 7168.1 мс, а у запиті 2 Planning Time = 0.5 мс та Execution Time = 0.153 мс. Отже продуктивність запиту SQL залежить від досвіду фахівця в створенні запитів SQL а використання сучасних засобів оптимізації поки не взмозі повною мірою замінити цей досвід.

Тому існують проблеми [7–9], які не не враховують сучасні оптимізатори, але їх треба враховувати досвідченим фахівцям під час створення запитів, але які доступні для розуміння не кожному аналітику з урахуванням його професійної спрямованості. А саме:

Синтаксичні помилки, які можуть призвести до неправильних або неефективних запитів, тому запити не інтерпретуються обробником баз даних належним чином.

Надто складні запити, які важко зрозуміти оптимізувати. Використання CTE замість вкладених підзапитів може покращити як читабельність, так і продуктивність.

Правильне використання агрегатних функцій, таких як COUNT, SUM, AVG, є важливим для ефективності операцій з даними. Важливо використовувати їх у контексті для отримання оптимальних результатів.

Уникнення надмірної кількості об’єднань: Занадто багато об’єднань між таблицями призводить до збільшення складності запитів та збільшення часу їх виконання. Зосередження на мінімізації кількості підключень допомагає покращити продуктивність.

Неправильний вибір типів даних призводить до неефективного зберігання інформації та проблем із продуктивністю.

Хаотичне використання функцій JOIN, занадто багато вкладених запитів та відсутність належних індексів.

Відсутність індексів для стовпців, що використовуються в операторі WHERE.

Уникати IN та використовувати EXISTS, щоб перевірити чи існують запитувані дані в таблиці.

Уникати HAVING та, де можливо, використовувати WHERE.

Обовязкове використання параметрів у SQL-запитах. Загрози безпеці SQL, включаючи атаки SQL-ін’єкцій, можуть призвести до серйозних порушень даних та втрати довіри користувачів. Тому цей метод запобігає вставці шкідливого коду в запити неавторизованими користувачами, тим самим захищаючи базу даних від атак та покращити читабельність коду, що полегшуєйого підтримку.

Отже, оптимізація запитів – це складний процес, і багато факторів, включаючи згадану раніше статистику бази даних, можуть призвести до повільнішого виконання запитів, ніж очікувалося.

Висновки. Оптимізація аналітичних запитів вимагає від аналітиків глибокого розуміння мови SQL та її принципів, тому що без розуміння і того, як працює механізм SQL, як він організовує дані та як інтерпретує команди неможливо створити оптимальний запит в сучасних умовах, що значно підвищує вимоги до компетентностей аналітиків.

Хоча сучасні оптимізатори запитів до баз даних мають значний потенціал і надають багато корисних функцій, все ж існують певні аспекти, в яких вони можуть не відповідати очікуванням аналітиків, а саме, комплексність запитів, тобто оптимізатори запитів мають обмежену здатність до ефективної оптимізації складних або дуже складних запитів, які зазвичай використовують у роботі аналітики. При наявності багатьох таблиць, складних умов, підзапитів і функцій можуть виникати виклики результативної оптимізації. Це означає, що вони не можуть автоматично адаптуватись до змін у навантаженні або конкретних умов виконання запиту. Як результат, вони можуть не завжди найкращим чином пристосовуватись до змінних умов та динамічних вимог. Отже, створення нового оптимізатора може бути обгрунтовано необхідністю вирішення цих недоліків та надання оптимального та адаптивного підходу до оптимізації запитів.

Список літератури:

1. Варенко В.М. Інформаційно-аналітична діяльність: Навч. посіб. / К.: Університет «Україна», 2014. 417 с.
2. Fabio Duarte. Amount of Data Created Daily. 2025. URL: https://explodingtopics.com/blog/data-generated-per-day
3. Kamila Ostrowska. The Most Popular Databases in 2024. URL: https://learnsql.com/blog/most-popular-databases-2024/
4. Rupley, M. L. Introduction to query processing and optimization. Indiana University, South Bend, South Bend, IN, USA, TechReport TR–20080105–1.2.Bhajipale, R.,Bisen, P., Meshram, A., & Thakur, S. S. 2016.
5. Ahmad K. Query Performance in Database Operation. URL: https://www.ftsm.ukm.my/v5/file/research/technicalreport/PS-FTSM-2020-045.pdf
6. Tuning the SQLquery in order to reduce time consumption. International Journalof Computer Science Issues, 9 (4/3), 418–423.4.Habimana, J. 2015.
7. Query optimization techniques – tips for writing efficient and faster SQL queries. International Journal of Scientific &Technology Research, 4 (10), 22–26.5.Sahal, R., Nihad, M., Khafagy, M.H., & Omara, F. A. 2018.
8. Белоус Р., Крилов Є., Анікін В. Методи оптимізації запитів розподілених БД. Адаптивні системи автоматичного управління. 2021. Т. 2, № 39. С. 3–11. URL: https://doi.org/10.20535/1560-8956.39.2021.247364
9. Query optimization techniques in SQL Server: Database Design and Architecture July 13, 2018 by Ed Pollack URL: https://surli.cc/nsnbgq

Boiko O.V., Ulianytska K.O., Kolomiiets U.D. ON THE QUESTION OF OPTIMIZATION OF RELATIONAL DATABASE QUERIES FOR SOLVING ANALYTICAL PROBLEMS

The work considers analyzed methodsfor optimizing queries to a relational database in order to increase their productivity when solving everyday tasks by analysts in conditions of constantly increasing amounts of information for further analytical analysis.

It was determined that although modern query optimizers for relational databases have significant potential and provide many useful functions, they may not meet the expectations of analysts. Query optimizers have a limited ability to effectively optimize complex or very complex queries, which are usually used in the work of analysts. When there are many tables, complex conditions, subqueries and functions, difficulties may arise when optimizing queries. That is, optimizers cannot automatically adapt to changes in the load or to specific conditions for executing the query. Optimizers often focus on optimizing the query itself, but do not provide a deep analysis of the data structure.

In addition, there are practical recommendations for manual query optimization, i.e. rewriting an existing query taking into account the current situation based on a deep understanding of the features of the structured query language, which, as a rule, remains beyond the experience of an ordinary analyst and which are not taken into account by existing optimizers. Therefore, the creation of a new optimizer can be justified by the need to take these shortcomings into account and provide an optimal and adaptive approach to query optimization. The new optimizer can focus its resources on ensuring effective response to changes in data, using computational resources to generate fast and optimal execution plans, and maximizing the use of parallel and distributed execution to ensure high performance in a variety of data environments.

Key words: *query optimization, relational database, structured query language, analyst, analytical work.*