Неоптимізований запит:

**SELECT**

pb.genre,

pb.author,

**COUNT**(**DISTINCT** ab.title) **AS** total\_books\_amazon,

**ROUND**(**AVG**(**CAST**(**REPLACE**(ab.price, '₹', '') **AS** **DECIMAL**(10,2))), 2) **AS** avg\_price\_amazon,

**AVG**(ab.rating) **AS** avg\_rating\_amazon,

**AVG**(pb.rating) **AS** avg\_rating\_perfect,

(

**SELECT** **COUNT**(\*)

**FROM** amazon\_popular\_books apb

**WHERE** apb.author = pb.author **AND** apb.rating > 4.5

) **AS** highly\_rated\_popular\_books,

**RANK**() **OVER** (**PARTITION** **BY** pb.genre **ORDER** **BY** **AVG**(pb.total\_ratings) **DESC**) **AS** author\_rank\_in\_genre

**FROM** perfect\_books pb

**LEFT** **JOIN** amazon\_books ab **ON** pb.title = ab.title **AND** pb.author = ab.author

**WHERE** pb.rating >= 4.0

**AND** pb.total\_ratings > 100

**AND** ab.price **IS** **NOT** **NULL**

**GROUP** **BY** pb.genre, pb.author

**HAVING** **COUNT**(**DISTINCT** ab.title) >= 2

**ORDER** **BY** avg\_rating\_amazon **DESC**

**LIMIT** 50;

Результат першого кроку оптимізації:

**WITH** *books\_joined* **AS** (

**SELECT**

*pb*.genre **AS** *genre*,

*pb*.author **AS** *author*,

*ab*.title **AS** *title*,

**CAST**(**REPLACE**(*ab*.price, '₹', '') **AS** **DECIMAL**(10,2))**AS** *price*,

*ab*.rating **AS** *amazon\_rating*,

*pb*.rating **AS** *perfect\_rating*,

*pb*.total\_ratings **AS** *total\_ratings*

**FROM** perfect\_books *pb*

**LEFT** **JOIN** amazon\_books *ab*

**ON** *pb*.title = *ab*.title **AND** *pb*.author = *ab*.author

**WHERE** *pb*.rating >= 4.0

**AND** *pb*.total\_ratings > 100

**AND** *ab*.price **IS** **NOT** **NULL**

),

*popular\_stats* **AS** (

**SELECT**

author,

**COUNT**(\*) **AS** *highly\_rated\_books*

**FROM** amazon\_popular\_books

**WHERE** rating > 4.5

**GROUP** **BY** author

)

**SELECT**

*b*.genre,

*b*.author,

**COUNT**(**DISTINCT** *b*.title) **AS** *total\_books\_amazon*,

**ROUND**(**AVG**(*b*.price),2) **AS** *avg\_price\_amazon*,

**AVG**(*b*.amazon\_rating) **AS** *avg\_rating\_amazon*,

**AVG**(*b*.perfect\_rating) **AS** *avg\_rating\_perfect*,

**MAX**(**COALESCE**(*ps*.highly\_rated\_books, 0)) **AS** *highly\_rated\_popular\_books*,

**RANK**() **OVER** (**PARTITION** **BY** *b*.genre **ORDER** **BY** **AVG**(*b*.total\_ratings) **DESC**) **AS** *author\_rank\_in\_genre*

**FROM** books\_joined *b*

**LEFT** **JOIN** popular\_stats *ps*

**ON** *ps*.author = *b*.author

**GROUP** **BY** *b*.genre, *b*.author

**HAVING** **COUNT**(**DISTINCT** *b*.title) >= 2

**ORDER** **BY** *avg\_rating\_amazon* **DESC**

**LIMIT** 50;

Пояснення:

Я переписала запит створивши дві CTE

* Books\_joined

З’єднує perfect\_books з amazon\_books

Відразу використовується фільтрація

**WHERE** *pb*.rating >= 4.0

**AND** *pb*.total\_ratings > 100

**AND** *ab*.price **IS** **NOT** **NULL**

* Popular\_stats

Окремо підраховуємо кількість книг з рейтингом більше ніж 4.5 у таблиці amazon\_popular\_books

Замість сабселекту зробили left join

Було:

(

**SELECT** **COUNT**(\*)

**FROM** amazon\_popular\_books apb

**WHERE** apb.author = pb.author **AND** apb.rating > 4.5

) **AS** highly\_rated\_popular\_books,

Стало:

**LEFT** **JOIN** popular\_stats *ps*

**ON** *ps*.author = *b*.author

Це дозволило уникнути повторного виконання підзапиту для кожного рядка

Також у нас всі обчислення винесені у фінальний селект, а підготовка даних до них у CTE

Перший крок оптимізації з використанням CTE дозволяє нам уникнути дублювання, полегшити розуміння коду, запит став оброблятися декілька секунд, неоптимізований – близько 30с.   
  
Крок два, додавання індексів:

**CREATE** **INDEX** idx\_perfect\_rating **ON** perfect\_books(rating);

**CREATE** **INDEX** idx\_perfect\_total\_rating **ON** perfect\_books(total\_ratings);

**CREATE** **INDEX** idx\_perfect\_author\_genre **ON** perfect\_books(author, genre);

**CREATE** **INDEX** idx\_amazon\_books\_title\_author **ON** amazon\_books(title(100), author(100));

**CREATE** **INDEX** idx\_popular\_authot\_rating **ON** amazon\_popular\_books(author, rating);

Пояснення кожного індексу:

1.

**CREATE** **INDEX** idx\_perfect\_rating **ON** perfect\_books(rating);

Тут ми робимо фільтрацію

**WHERE** *pb*.rating >= 4.0

Індекс прискорює відбір рядків, не буде повного сканування таблиці, бо b-tree вибере тільки те що нам потрібно

2.

**CREATE** **INDEX** idx\_perfect\_total\_rating **ON** perfect\_books(total\_ratings);

Теж саме що і з першим, використовується у фільтрації і сортуванні

3.

**CREATE** **INDEX** idx\_perfect\_author\_genre **ON** perfect\_books(author, genre);

У нас є групування

**GROUP** **BY** *b*.genre, *b*.author

Покриває одразу дві умови і шукає швидше ніж якщо би було окремо

4.

**CREATE** **INDEX** idx\_amazon\_books\_title\_author **ON** amazon\_books(title(100), author(100));

У нас є джойн

**LEFT** **JOIN** amazon\_books *ab*

**ON** *pb*.title = *ab*.title **AND** *pb*.author = *ab*.author

За допомогою індексу у нас є швидкий доступ до записів із назвою та автором, якщо би його не було, воно би шукало збіги повним перебором

5.

**CREATE** **INDEX** idx\_popular\_authot\_rating **ON** amazon\_popular\_books(author, rating);

**WHERE** rating > 4.5

**GROUP** **BY** author

Індекс дозволяє нам швидко вибрати тільки рядки де рейтинг більший за 4.5 і потім Group by працює швидше через одразу відсортованих авторів

На третьому етапі оптимізації я зробила декілька змін:

Було:

**LEFT** **JOIN** amazon\_books *ab*

**ON** *pb*.title = *ab*.title **AND** *pb*.author = *ab*.author

**WHERE** *pb*.rating >= 4.0

**AND** *pb*.total\_ratings > 100

**AND** *ab*.price **IS** **NOT** **NULL**

Стало:

**INNER** **JOIN** amazon\_books *ab*

**ON** *pb*.title = *ab*.title

**AND** *pb*.author = *ab*.author

**AND** *ab*.price **IS** **NOT** **NULL**

**WHERE** *pb*.rating >= 4.0

**AND** *pb*.total\_ratings > 100

Я змінила Left join на Inner Join, оскільки спочатку ми просто додавали до лівої таблиці праву і витрачали час на пусті рядки, які замінялись на NULL, отже у результаті з звичайним Join в нас фільтрується зайве

Також у нас ця строчка

**AND** *pb*.total\_ratings > 100

Фільтрувалась через where, я її переставила до join

Тобто раніше через Left Join у нас створювалися NULL значення які ми потім прибирали, зараз в нас з Join відувається фільтрація на момент об’єднання

Останній крок – оптимізація за допомогою підсказок для індексів

Що змінено:

Було:

**INNER** **JOIN** amazon\_books *ab*

**ON** *pb*.title = *ab*.title

**AND** *pb*.author = *ab*.author

**AND** *ab*.price **IS** **NOT** **NULL**

Стало:

**FORCE** **INDEX** (idx\_perfect\_rating, idx\_perfect\_total\_rating)

**STRAIGHT\_JOIN** amazon\_books ab

**USE** **INDEX** (idx\_amazon\_books\_title\_author)

**ON** pb.title = ab.title

**AND** pb.author = ab.author

**AND** ab.price **IS** **NOT** **NULL**

Було:

**SELECT**

author,

**COUNT**(\*) **AS** *highly\_rated\_books*

**FROM** amazon\_popular\_books

**WHERE** rating > 4.5

**GROUP** **BY** author

Стало:

**SELECT**

author,

**COUNT**(\*) **AS** highly\_rated\_books

**FROM** amazon\_popular\_books

**FORCE** **INDEX** (idx\_popular\_authot\_rating)

**WHERE** rating > 4.5

**GROUP** **BY** author

Насправді, explain показав що і без force index, straight\_join, use index деякі з них він використовує, але таким чином ми забезпечили контроль над селектами і точно можемо сказати, що вони буде виконуватися найефективнішим способом

Ці додаткові підказки довзоляють нам направити MySQL у правильному напрямку. Іноді він може сканувати таблиці повністю та ігнорувати наявні індекси, приєднувати таблиці у неправильному порядку. Тому використовуючи підказки для нього, ми можемо заставити зробити запит, так як ми вважаємо за потрібне, через це може підвищитися ефективність та ми можемо гарантувати стабільне виконання навіть при зміні даних.