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.author, b.genre
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 хвилини.

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_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_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

4.

```
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 ab.price IS NOT NULL
```

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 відбувається фільтрація на момент об'єднання

Останній крок – оптимізація за допомогою підказок 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.