1. **Чому другий запит (CTE + віконні функції) працює швидше за оригінальний?**

    - **Початковий варіант -** містив два корельовані підзапити (для avg і для ранжування), які для кожного рядка таблиці `car_table` робили повний скан усіх ~4020 записів. Це давало приблизно O(N²) операцій при великій кількості результатів.
    - **Другий запит -** (з CTE) один раз сканує всю таблицю для обчислення віконних функцій AVG(MSRP) OVER (PARTITION BY Make, Year) та RANK() OVER (PARTITION BY Year ORDER BY MSRP), а далі працює лише з “відібраними” даними (446 рядків після фільтрації).
    - У підсумку замість кількох тисяч сканів ми отримали два одноразові сканування (4020+446), що зменшило обсяг читання майже в 3 рази. Саме тому другий варіант виявився суттєво швидшим.

2. **Чому після додавання індексів `(Year, MSRP)` та `(Make, Year)` запит уповільнився?**

    - Створений індекс `(Year, MSRP)` мав низьку селективність: із 4020 записів приблизно 3800 вже задовольняють умову `Year BETWEEN 2015 AND 2020 AND MSRP IS NOT NULL`. Тобто індекс не скорочував кількість читаних рядків, але при цьому додав додаткові звертання до B-Tree.
    - Навіть коли індекс відсікав кілька сотень записів, MySQL усе одно мусив виконувати `filesort` для `ORDER BY Make, Model`, що створювало додатковий оверхед.
    - У результаті кожне звертання по індексу (щоб знайти рядки в діапазоні Year–MSRP) плюс «створення тимчасового сортування» виявлялися дорожчими за простий повний скан у пам’яті. Тому реальний час виконання зріс.