Advanced SQL Exercises for Online Retail Store

**Exercise 1: Ranking and Window Functions**

Goal: Use ROW\_NUMBER(), RANK(), DENSE\_RANK(), OVER(), and PARTITION BY.

Scenario:

Find the top 3 most expensive products in each category using different ranking functions.

Steps:

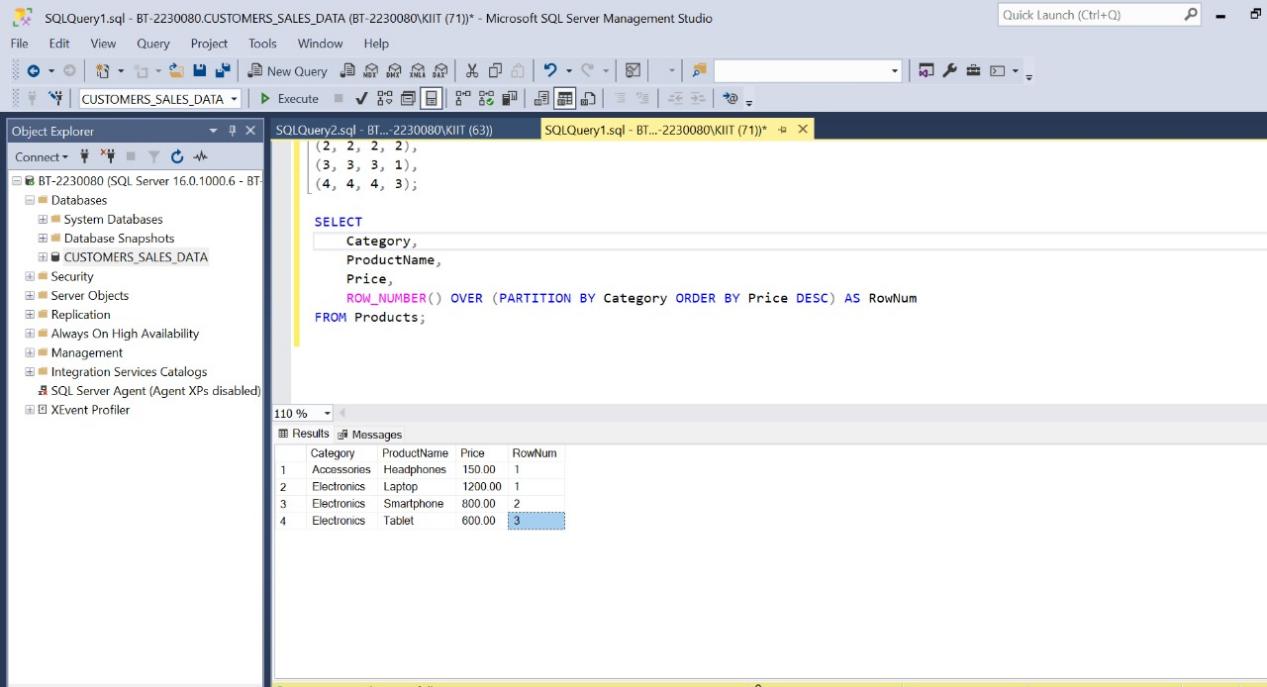
1. Use ROW\_NUMBER() to assign a unique rank within each category.

2. Use RANK() and DENSE\_RANK() to compare how ties are handled.

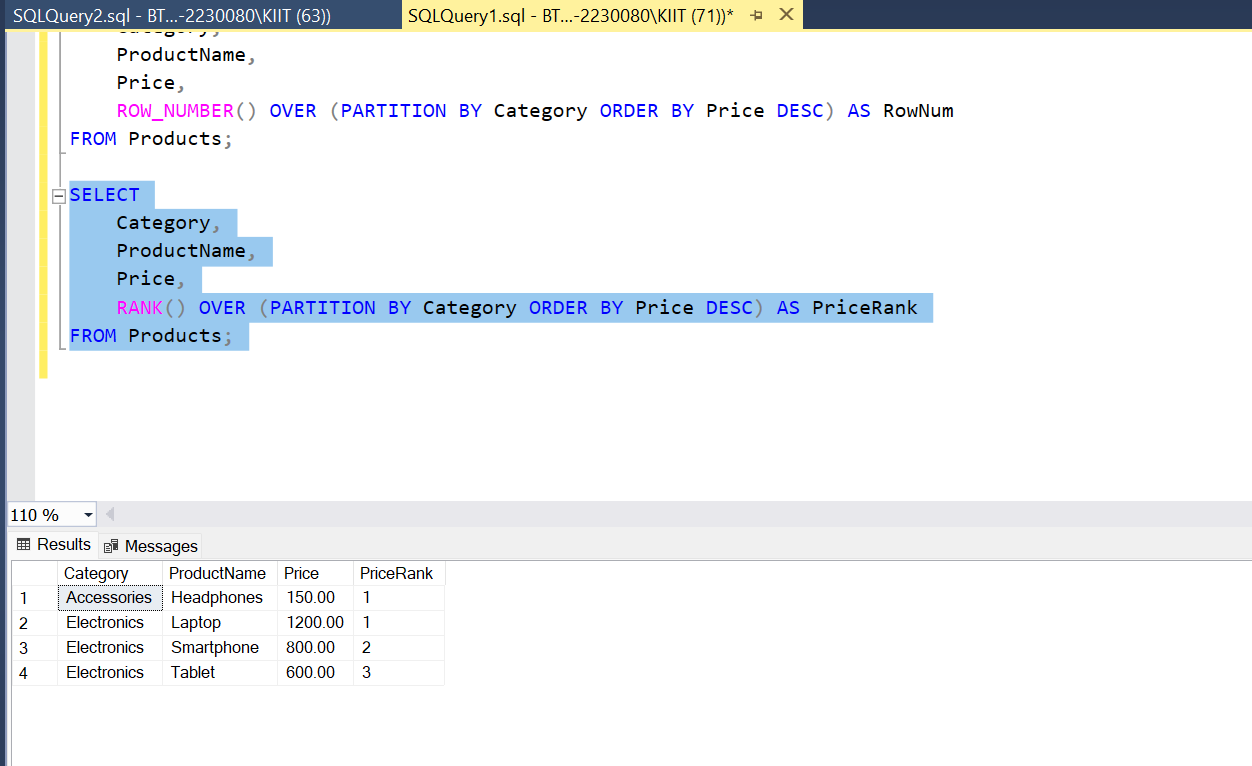
3. Use PARTITION BY Category and ORDER BY Price DESC

Using sample data from Index.sql file

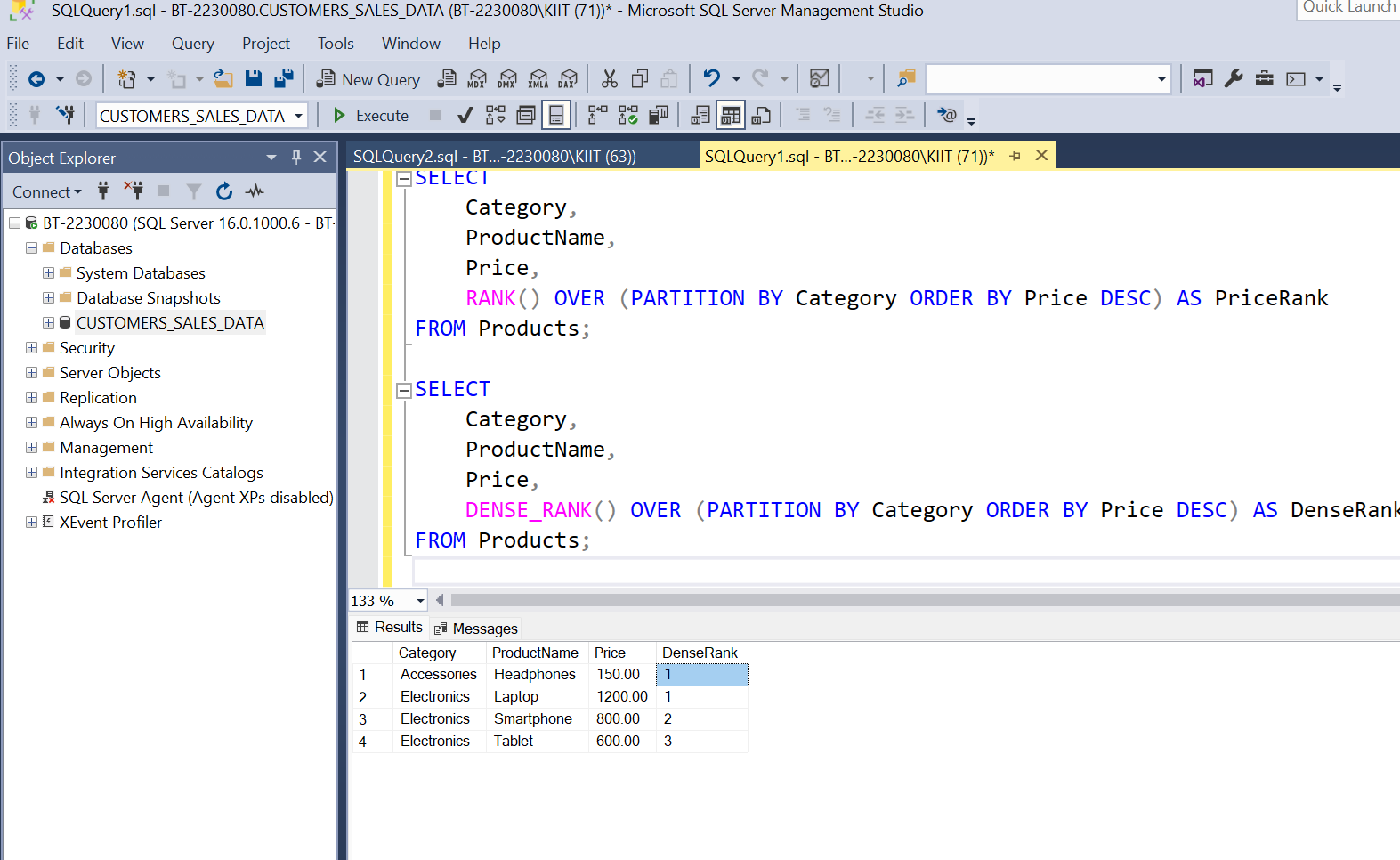
ROW\_NUMBER()



RANK()

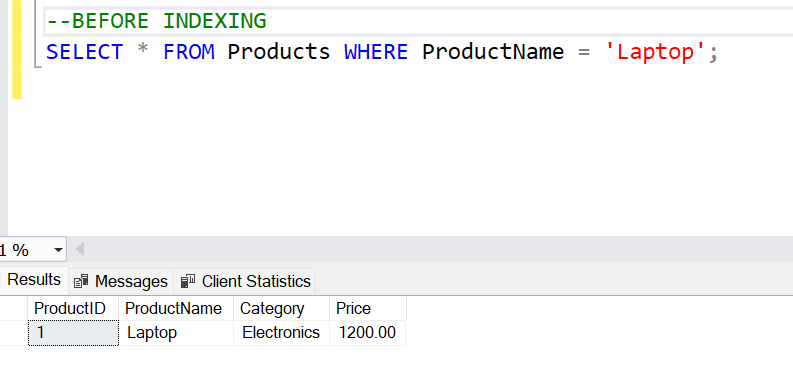


DENSE

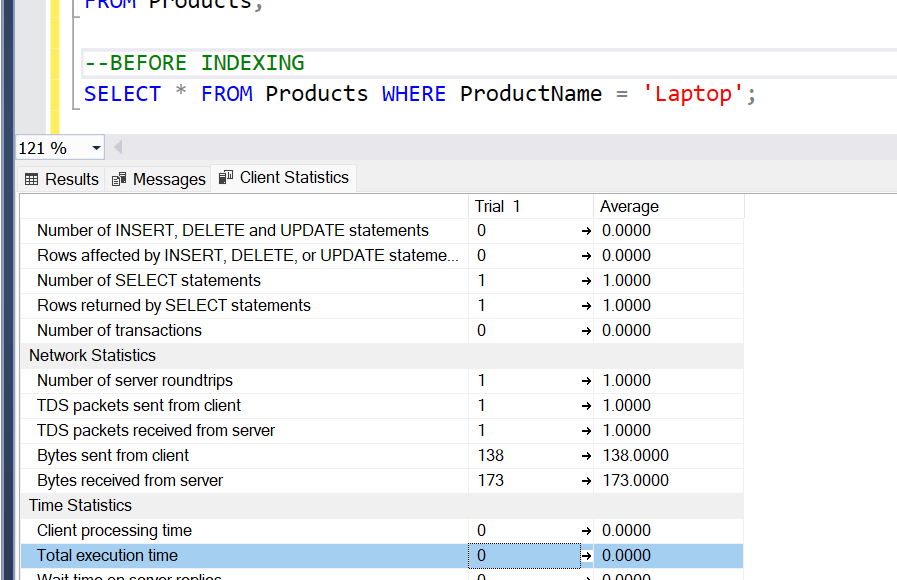


**INDEXING**

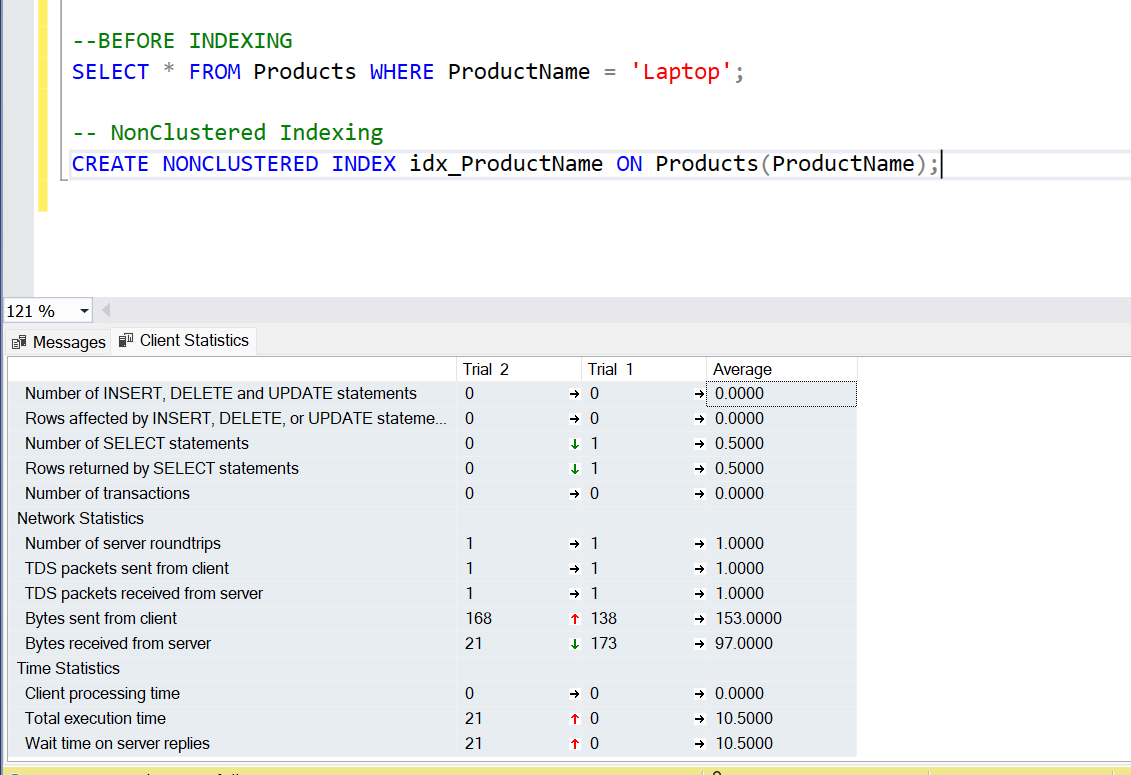
BEFORE INDEXING



Exection time is negligible as the number of entries are minimal and limited

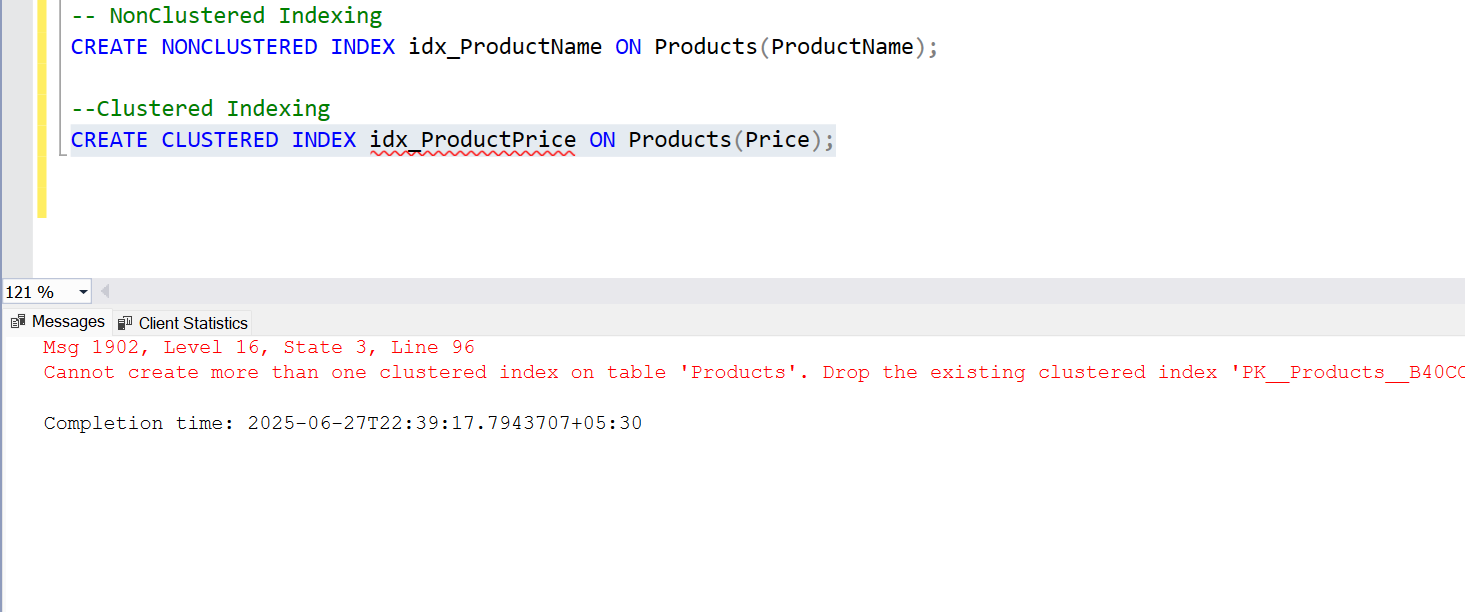


After Non- clustered Indexing :

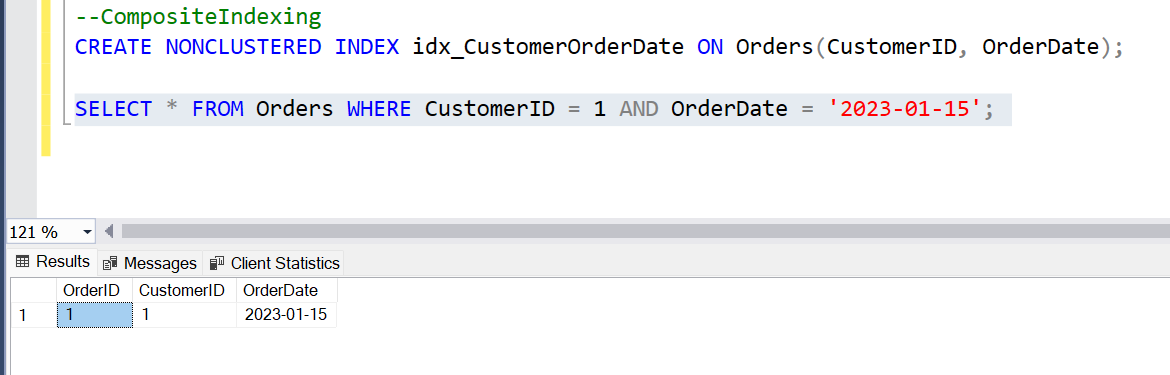


clustered Indexing :

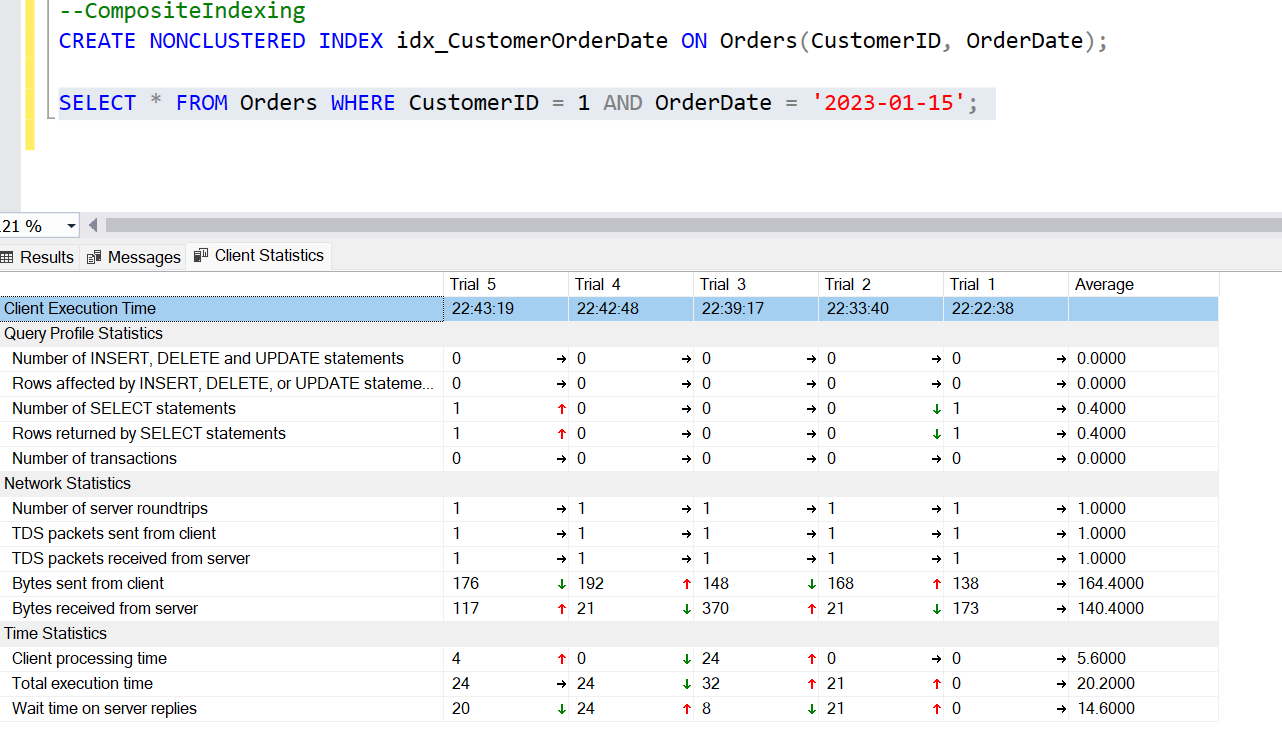
Since only one clustered indexing is possible in a table and most have a default clustered indexing on primary key , hence showsb error while creating another clustered index.



Composite Indexing:



Exection statistics :



A composite non-clustered index was created on Orders(CustomerID, OrderDate) to improve query performance

**Before indexing**, queries had higher wait times (Trial 3: **32 ms** total execution).

**After indexing**, wait time dropped ( Trial 5: **24 ms total**, **20 ms server wait**).

Average execution time across trials was **20.2 ms**, with consistent **1 server roundtrip** and **1 row returned**.

This confirms that the index allowed SQL Server to quickly locate results, improving performance without changing the query logic.