

# SQL - Funkcje okna (Window functions)

## Lab 1-2

Imię i Nazwisko: 

|                                       |
|---------------------------------------|
| Bartłomiej Jamiołkowski, Ada Bodziony |
|---------------------------------------|

Celem ćwiczenia jest zapoznanie się z działaniem funkcji okna (window functions) w SQL, analiza wydajności zapytań i porównanie z rozwiązaniami przy wykorzystaniu "tradycyjnych" konstrukcji SQL

Swoje odpowiedzi wpisuj w **czerwone pola**.

Ważne/wymagane są komentarze.

Zamieść kod rozwiązania oraz zrzuty ekranu pokazujące wyniki, (dołącz kod rozwiązania w formie tekstowej/źródłowej)

Zwróć uwagę na formatowanie kodu

## Oprogramowanie - co jest potrzebne?

Do wykonania ćwiczenia potrzebne jest następujące oprogramowanie:

- MS SQL Server - wersja 2019, 2022
- PostgreSQL - wersja 15/16
- SQLite
- Narzędzia do komunikacji z bazą danych
  - o SSMS - Microsoft SQL Managment Studio
  - o DłataGrip lib DBeaver
- Przykładowa baza Northwind
  - o W wersji dla każdego z wymienionych serwerów

Oprogramowanie dostępne jest na przygotowanej maszynie wirtualnej

## Dokumentacja/Literatura

- Kathi Kellenberger, Clayton Groom, Ed Pollack, Expert T-SQL Window Functions in SQL Server 2019, Apres 2019
- Itzik Ben-Gan, T-SQL Window Functions: For Data Analysis and Beyond, Microsoft 2020

Kilka linków do materiałów które mogą być pomocne

- <https://learn.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql?view=sql-server-ver16>
- <https://www.sqlservertutorial.net/sql-server-window-functions/>

- <https://www.sqlshack.com/use-window-functions-sql-server/>
- <https://www.postgresql.org/docs/current/tutorial-window.html>
- <https://www.postgresqltutorial.com/postgresql-window-function/>
- <https://www.sqlite.org/windowfunctions.html>
- <https://www.sqlitetutorial.net/sqlite-window-functions/>

Ikonki używane w graficznej prezentacji planu zapytania w SSMS opisane są tutaj:

- <https://docs.microsoft.com/en-us/sql/relational-databases/showplan-logical-and-physical-operators-reference>

## Przygotowanie

Uruchom SSMS.

- Skonfiguruj połączenie z bazą Northwind na lokalnym serwerze MS SQL

Uruchom DataGrip (lub Dbeaver)

- Skonfiguruj połączenia z bazą Northwind
  - na lokalnym serwerze MS SQL
  - na lokalnym serwerze PostgreSQL
  - z lokalną bazą SQLite

## Zadanie 1 - obserwacja

Wykonaj i porównaj wyniki następujących poleceń.

```
select avg(unitprice) avgprice
from products p;

select avg(unitprice) over () as avgprice
from products p;

select categoryid, avg(unitprice) avgprice
from products p
group by categoryid

select avg(unitprice) over (partition by categoryid) as avgprice
from products p;
```

Jaka jest są podobieństwa, jakie różnice pomiędzy grupowaniem danych a działaniem funkcji okna?

Wynikiem pierwszego polecenia jest średnia wszystkich cen z kolumny unitprice z tabeli products zawarta w nowej tabeli z jednym wierszem i jedną kolumną zatytułowaną avgprice.

Exercise\_1.sql - DE...8VFIDA8\Bartek (65))\*

```
USE [nortwind3-ms-2019]
select avg(unitprice) avgprice
from products p;
```

100 %

Results Messages

|   | avgprice |
|---|----------|
| 1 | 28,8663  |

Wynikiem drugiego polecenia jest tablica składająca się z jednej kolumny zatytułowanej avgprice oraz 77 wierszy (jedno okno), w której wszystkie komórki posiadają wartość średniej wszystkich cen z kolumny unitprice tabeli products.

Exercise\_1.sql - DE...8VFIDA8\Bartek (65))\*

```
USE [nortwind3-ms-2019]
select avg(unitprice) over () as avgprice
from products p;
```

100 %

Results Messages

|   | avgprice |
|---|----------|
| 1 | 28,8663  |
| 2 | 28,8663  |
| 3 | 28,8663  |
| 4 | 28,8663  |
| 5 | 28,8663  |
| 6 | 28,8663  |
| 7 | 28,8663  |

Wynikiem trzeciego polecenia jest tabela składająca się z dwóch kolumn zatytułowanych categoryid i avgprice oraz 8 wierszy. Każda z 8 kategorii posiada swoje id, której przypisana jest średnia cena produktu z danej kategorii.

Exercise\_1.sql - DE...8VFIDA8\Bartek (65))\* -> X

USE [nortwind3-ms-2019]

```
select categoryid, avg(unitprice) avgprice
from products p
group by categoryid
```

100 %

Results Messages

|   | categoryid | avgprice |
|---|------------|----------|
| 1 | 1          | 37,9791  |
| 2 | 2          | 23,0625  |
| 3 | 3          | 25,16    |
| 4 | 4          | 28,73    |
| 5 | 5          | 20,25    |
| 6 | 6          | 54,0066  |
| 7 | 7          | 32,37    |
| 8 | 8          | 20,6825  |

Wynikiem czwartego polecenia jest tabela z jedną kolumną avgprice oraz 77 wierszami. Kolumna avgprice zawiera średnie ceny produktów będącymi wartościami zagregowanymi obliczonymi na podstawie kolumny unitprice oraz okien wierszy z kolumny categoryid tabeli products.

Exercise\_1.sql - DE...8VFIDA8\Bartek (65))\* -> X

USE [nortwind3-ms-2019]

```
select avg(unitprice) over (partition by categoryid) as avgprice
from products p;
```

100 %

Results Messages

|    | avgprice |
|----|----------|
| 1  | 37,9791  |
| 2  | 37,9791  |
| 3  | 37,9791  |
| 4  | 37,9791  |
| 5  | 37,9791  |
| 6  | 37,9791  |
| 7  | 37,9791  |
| 8  | 37,9791  |
| 9  | 37,9791  |
| 10 | 37,9791  |
| 11 | 37,9791  |
| 12 | 37,9791  |
| 13 | 23,0625  |
| 14 | 23,0625  |
| 15 | 23,0625  |
| 16 | 23,0625  |

Podobieństwa pomiędzy grupowaniem danych a działaniem funkcji okna:

- mogą wykorzystywać funkcje agregujące np. AVG(),
- pozwalają na podział danych widoczny w wynikach zapytań,
- pozwalają na manipulowanie zestawami danych.

Różnice pomiędzy grupowaniem danych a działaniem funkcji okna:

- GROUP BY służy do grupowania danych i agregowania wyników, podczas gdy funkcje okna pozwalają na obliczenia w kontekście określonych "okien" danych bez zmiany liczby wierszy w wyniku zapytania.

## Zadanie 2 - obserwacja

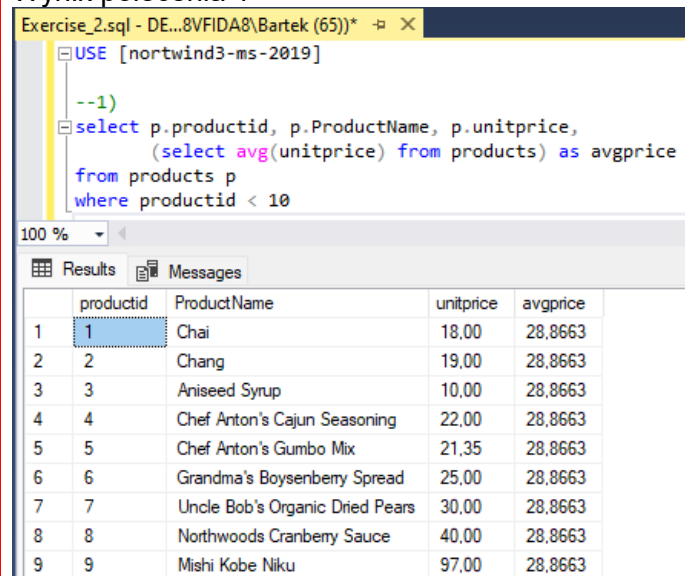
Wykonaj i porównaj wyniki następujących poleceń.

```
--1)
select p.productid, p.ProductName, p.unitprice,
       (select avg(unitprice) from products) as avgprice
from products p
where productid < 10

--2)
select p.productid, p.ProductName, p.unitprice,
       avg(unitprice) over () as avgprice
from products p
where productid < 10
```

Jaka jest różnica? Czego dotyczy warunek w każdym z przypadków? Napisz polecenie równoważne 1) z wykorzystaniem funkcji okna. Napisz polecenie równoważne 2) z wykorzystaniem podzapytania

### Wynik polecenia 1



|   | productid | ProductName                     | unitprice | avgprice |
|---|-----------|---------------------------------|-----------|----------|
| 1 | 1         | Chai                            | 18,00     | 28,8663  |
| 2 | 2         | Chang                           | 19,00     | 28,8663  |
| 3 | 3         | Aniseed Syrup                   | 10,00     | 28,8663  |
| 4 | 4         | Chef Anton's Cajun Seasoning    | 22,00     | 28,8663  |
| 5 | 5         | Chef Anton's Gumbo Mix          | 21,35     | 28,8663  |
| 6 | 6         | Grandma's Boysenberry Spread    | 25,00     | 28,8663  |
| 7 | 7         | Uncle Bob's Organic Dried Pears | 30,00     | 28,8663  |
| 8 | 8         | Northwoods Cranberry Sauce      | 40,00     | 28,8663  |
| 9 | 9         | Mishi Kobe Niku                 | 97,00     | 28,8663  |

### Wynik polecenia 2

Exercise\_2.sql - DE...8VFIDA8\Bartek (65))\* -> X

```
USE [nortwind3-ms-2019]

--2)
select p.productid, p.ProductName, p.unitprice,
       avg(unitprice) over () as avgprice
from products p
where productid < 10
```

100 %

Results Messages

|   | productid | ProductName                     | unitprice | avgprice |
|---|-----------|---------------------------------|-----------|----------|
| 1 | 1         | Chai                            | 18,00     | 31,3722  |
| 2 | 2         | Chang                           | 19,00     | 31,3722  |
| 3 | 3         | Aniseed Syrup                   | 10,00     | 31,3722  |
| 4 | 4         | Chef Anton's Cajun Seasoning    | 22,00     | 31,3722  |
| 5 | 5         | Chef Anton's Gumbo Mix          | 21,35     | 31,3722  |
| 6 | 6         | Grandma's Boysenberry Spread    | 25,00     | 31,3722  |
| 7 | 7         | Uncle Bob's Organic Dried Pears | 30,00     | 31,3722  |
| 8 | 8         | Northwoods Cranberry Sauce      | 40,00     | 31,3722  |
| 9 | 9         | Mishi Kobe Niku                 | 97,00     | 31,3722  |

Różnica polega na tym, że w obu tablicach w kolumnach avgprice są różne wartości średniej ceny, dlatego że w zapytaniu 1 średnia unitprice jest wyliczana jako średnia wszystkich rekordów w tabeli, a zapytaniu 2 średnia unitprice jest wyliczana na ograniczonej liczby rekordów productid < 10.

W każdym z przypadków warunek dotyczy zapytania, którego wynikiem mają być dane dotyczące produktów o id mniejszym od 10.

Polecenie równoważne 1) z wykorzystaniem funkcji okna.

```
select *
from ( select
        productid,
        ProductName,
        unitprice,
        avg(unitprice) over() as avgprice
      from products
    ) as tmp
where productId < 10
```

Wynik:

100 %

Results Messages

|   | productid | ProductName                     | unitprice | avgprice |
|---|-----------|---------------------------------|-----------|----------|
| 1 | 1         | Chai                            | 18,00     | 28,8663  |
| 2 | 2         | Chang                           | 19,00     | 28,8663  |
| 3 | 3         | Aniseed Syrup                   | 10,00     | 28,8663  |
| 4 | 4         | Chef Anton's Cajun Seasoning    | 22,00     | 28,8663  |
| 5 | 5         | Chef Anton's Gumbo Mix          | 21,35     | 28,8663  |
| 6 | 6         | Grandma's Boysenberry Spread    | 25,00     | 28,8663  |
| 7 | 7         | Uncle Bob's Organic Dried Pears | 30,00     | 28,8663  |
| 8 | 8         | Northwoods Cranberry Sauce      | 40,00     | 28,8663  |
| 9 | 9         | Mishi Kobe Niku                 | 97,00     | 28,8663  |

Polecenie równoważne 2) z wykorzystaniem podzapytania.

```
select
```

```

    p1.productid,
    p1.ProductName,
    p1.unitprice,
    avgprice.avg_price as avgprice
from products as p1
join (select avg(unitprice) as avg_price from products where ProductID < 10 ) as
avgprice on 1=1
where productid < 10

```

Wynik:

|   | productid | ProductName                     | unitprice | avgprice |
|---|-----------|---------------------------------|-----------|----------|
| 1 | 1         | Chai                            | 18,00     | 31,3722  |
| 2 | 2         | Chang                           | 19,00     | 31,3722  |
| 3 | 3         | Aniseed Syrup                   | 10,00     | 31,3722  |
| 4 | 4         | Chef Anton's Cajun Seasoning    | 22,00     | 31,3722  |
| 5 | 5         | Chef Anton's Gumbo Mix          | 21,35     | 31,3722  |
| 6 | 6         | Grandma's Boysenberry Spread    | 25,00     | 31,3722  |
| 7 | 7         | Uncle Bob's Organic Dried Pears | 30,00     | 31,3722  |
| 8 | 8         | Northwoods Cranberry Sauce      | 40,00     | 31,3722  |
| 9 | 9         | Mishi Kobe Niku                 | 97,00     | 31,3722  |

### Zadanie 3

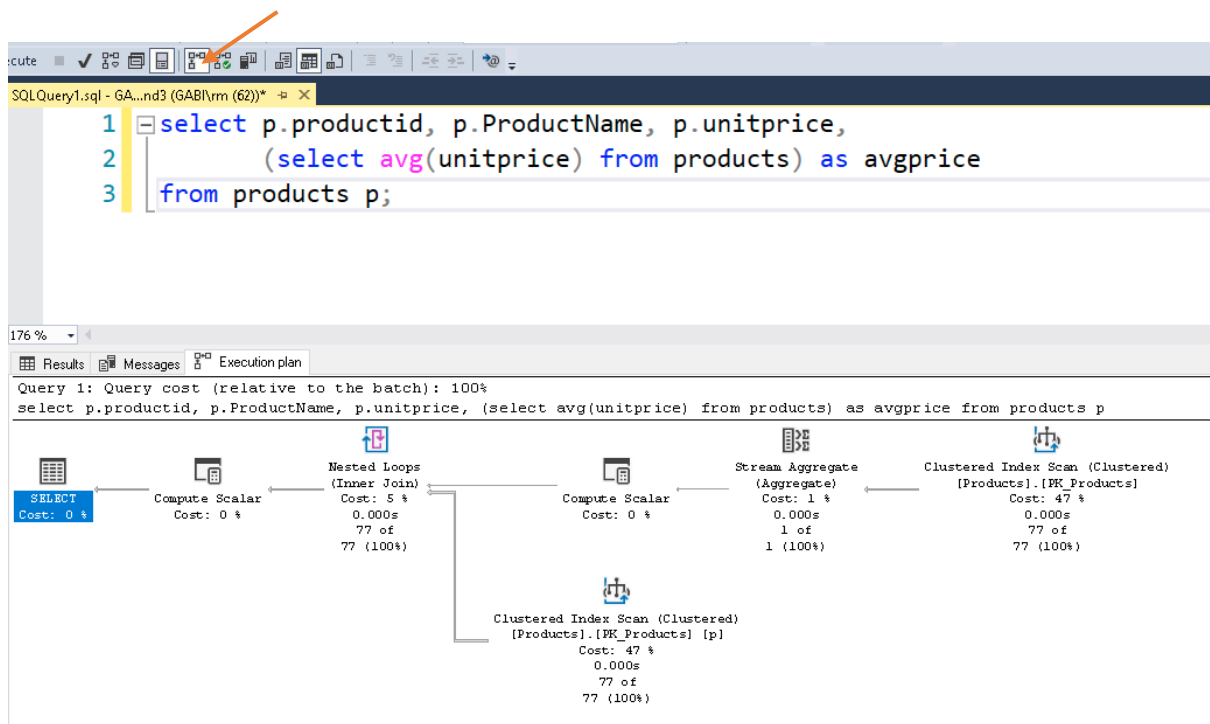
Baza: Northwind, tabela: products

Napisz polecenie, które zwraca: id produktu, nazwę produktu, cenę produktu, średnią cenę wszystkich produktów.

Napisz polecenie z wykorzystaniem z wykorzystaniem podzapytania, join'a oraz funkcji okna. Porównaj czasy oraz plany wykonania zapytań.

Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSQL, SQLite)

W SSMS włącz dwie opcje: Include Actual Execution Plan oraz Include Live Query Statistics



W DataGrip użyj opcji Explain Plan/Explain Analyze

The screenshot shows the DataGrip interface with a query editor, a context menu, and a results table.

The query in the editor is:

```
1 select p.productid, p.ProductName, p.unitprice,
2       (select avg(unitprice) from products) as avgprice
3 from products p;
```

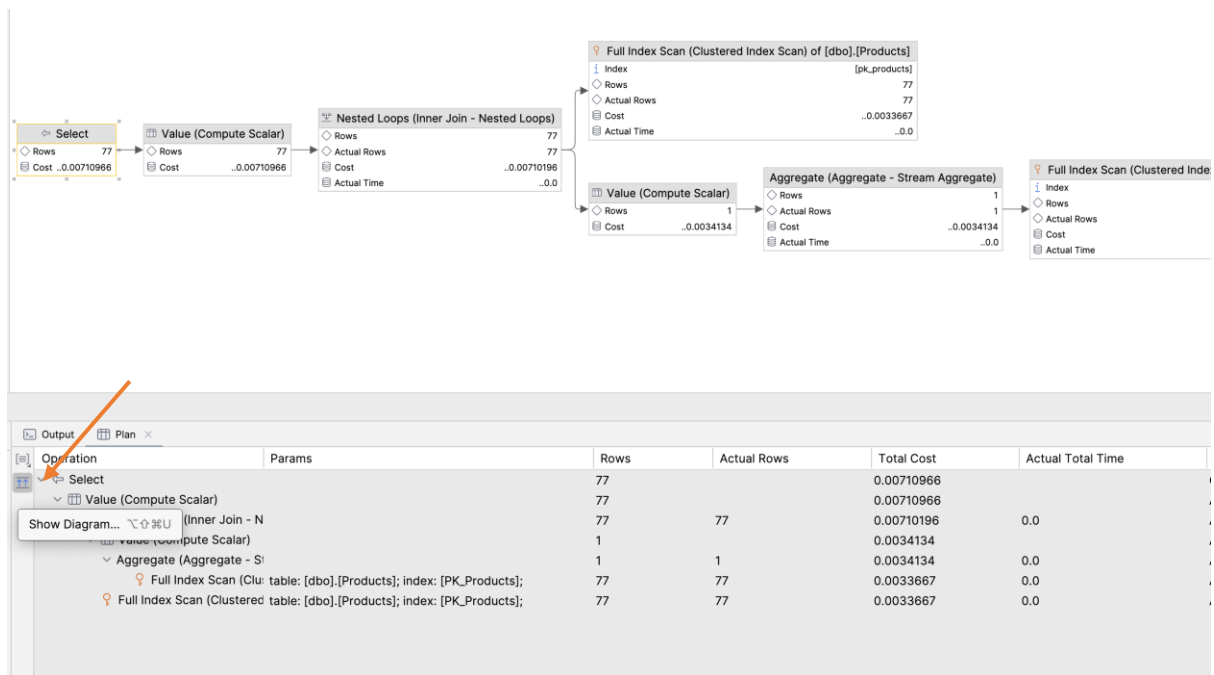
The context menu is open, showing the following options:

- Show Context Actions
- Paste
- Copy / Paste Special
- Column Selection Mode
- Refactor
- Folding
- Save as Live Template...
- Reformat Code
- Go To
- Generate...
- Run 'l.sql'
- More Run/Debug
- Switch Session (w)
- Explain Plan**
- Execute
- Execute to File
- Open In

The results table shows the following data:

|                | Rows | Actual Rows | Total Cost | Actual Total Ti |
|----------------|------|-------------|------------|-----------------|
|                | 77   |             | 0.00710966 |                 |
|                | 77   |             | 0.00710966 |                 |
|                | 77   | 77          | 0.00710196 | 0.0             |
|                | 1    |             | 0.0034134  |                 |
|                | 1    | 1           | 0.0034134  | 0.0             |
| [PK_Products]; | 77   | 77          | 0.0033667  | 0.0             |
| [PK_Products]; | 77   | 77          | 0.0033667  | 0.0             |





Polecenie 1 z wykorzystaniem podzapytania

```
select
    ProductID,
    ProductName,
    UnitPrice,
    (select avg(UnitPrice) as avg_price from Products) as avgprice
from Products
```

Polecenie 2 z wykorzystaniem join'a

```
select
    p1.ProductID,
    p1.ProductName,
    p1.UnitPrice,
    avg(p2.UnitPrice) as avgprice
from Products as p1
inner join Products as p2 on 1=1
group by p1.ProductID,
         p1.ProductName,
         p1.UnitPrice
```

Polecenie 3 z wykorzystaniem funkcji okna.

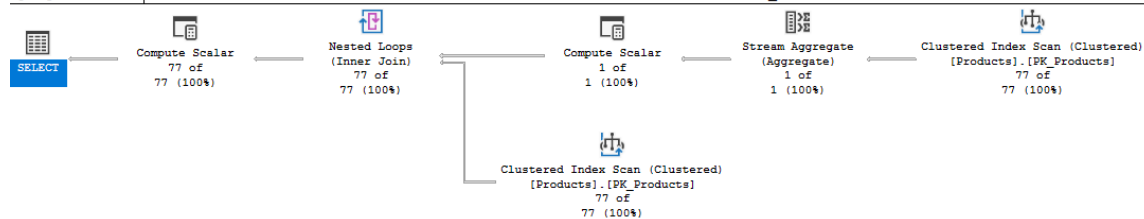
```
select
    ProductID,
    ProductName,
    UnitPrice,
    avg(unitprice) over() as avgprice
from Products
```

Porównanie czasów oraz planów wykonania zapytań

a) MS SQL Server

Polecenie 1

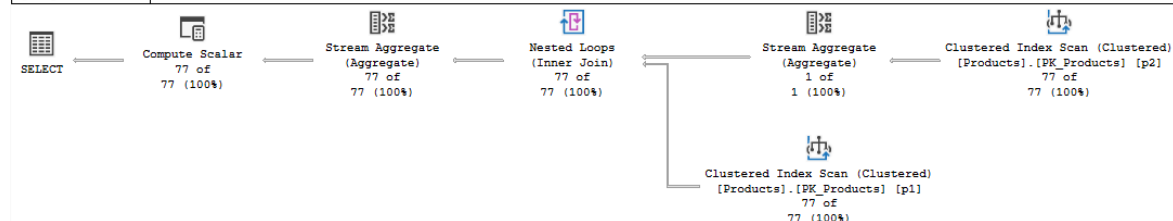
Estimated query progress:100% Query 1: Query cost (relative to the batch): 37%  
 select ProductID, ProductName, UnitPrice, (select avg(UnitPrice) as avg\_price from Products) as avgprice from Products



Estimated Subtree Cost: 0,0071097  
 Query cost (relative to the batch): 37%

## Polecenie 2

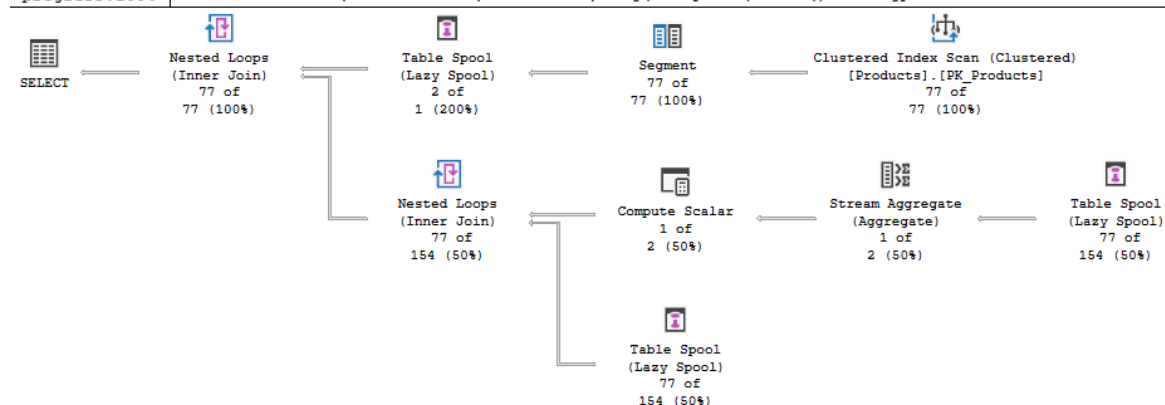
Estimated query progress:100% Query 2: Query cost (relative to the batch): 38%  
 select p1.ProductID, p1.ProductName, p1.UnitPrice, avg(p2.UnitPrice) as avgprice from Products as p1 inner join Prod



Estimated Subtree Cost: 0,007944  
 Query cost (relative to the batch): 38%

## Polecenie 3

Estimated query progress:100% Query 3: Query cost (relative to the batch): 25%  
 select ProductID, ProductName, UnitPrice, avg(unitprice) over() as avgprice from Products



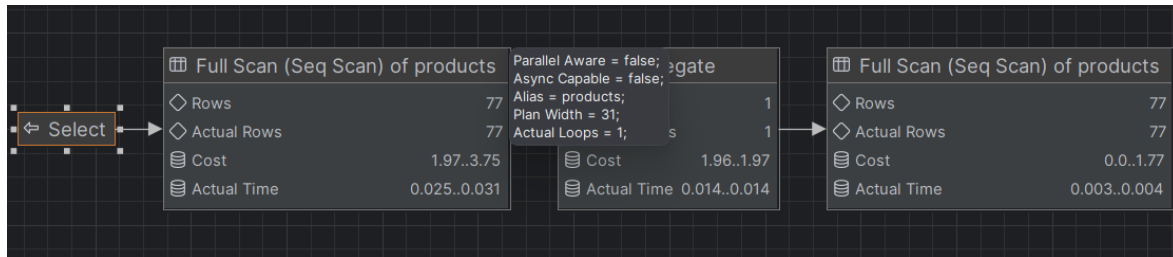
Estimated Subtree Cost: 0,0048281  
 Query cost (relative to the batch): 25%

## Porównanie:

Najmniejszy koszt Estimated Subtree Cost=0,0048281 uzyskano dla polecenia 3 z wykorzystaniem funkcji okna. Kolejny wynik Estimated Subtree Cost=0,0071097 uzyskano dla polecenia 1 z podzapytaniem. 3 najgorszy wynik Estimated Subtree Cost=0,007944 uzyskano dla polecenia 2 z użyciem join. Te wnioski potwierdza również Query cost (relative to the batch) pokazujący udziały poszczególnych poleceń w koszcie uruchomienia 3 poleceń naraz. Najmniejszy udział 25% ma polecenie 3 z użyciem funkcji okna. Następnie polecenie 1 zajmuje drugie miejsce pod względem udziału w ogólnym koszcie poleceń (37%). Trzecie najgorsze miejsce zajmuje polecenie 2 z join 38%. Z obserwacji wynika, że najwydajniejszym poleceniem jest polecenie 3 z wykorzystaniem funkcji okna. Drugim wydajnym poleceniem jest polecenie 1 z podzapytaniem. Najgorszą wydajności cechuje się polecenie 2 z join.

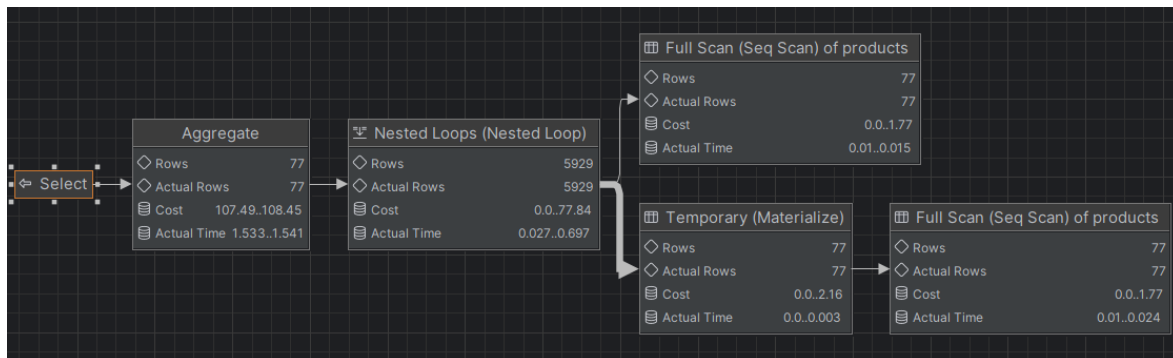
## b) PostgreSQL

### Polecenie 1



Cost=1.97

### Polecenie 2



Cost=107.49

### Polecenie 3



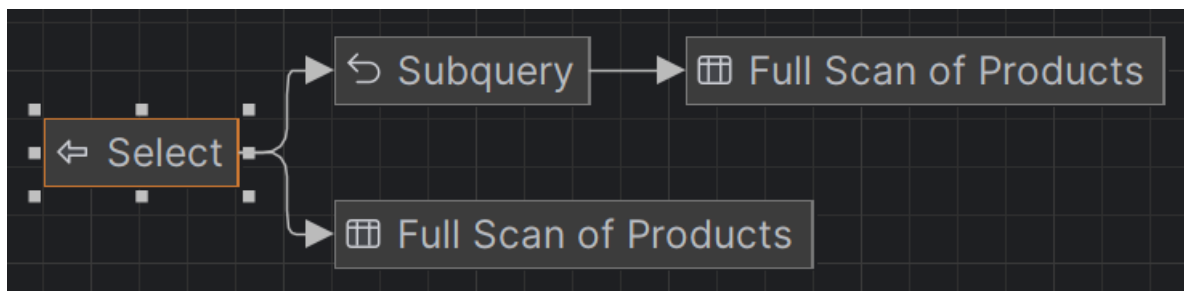
Cost=0.0...2

Porównanie:

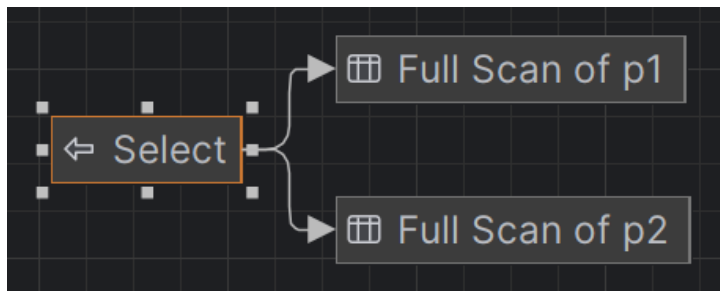
Najmniejszy koszt Cost=0.0...2 uzyskano dla polecenia 3 z wykorzystaniem funkcji okna. Kolejny wynik Cost=1.97 uzyskano dla polecenia 1 z podzapytaniem. 3 najgorszy wynik Cost=107.49 uzyskano dla polecenia 2 z użyciem join. Z obserwacji wynika, że najwydajniejszym poleceniem jest polecenie 3 z wykorzystaniem funkcji okna. Drugim wydajnym poleceniem jest polecenie 1 z podzapytaniem. Najgorszą wydajności cechuje się polecenie 3 z join.

## c) SQLite

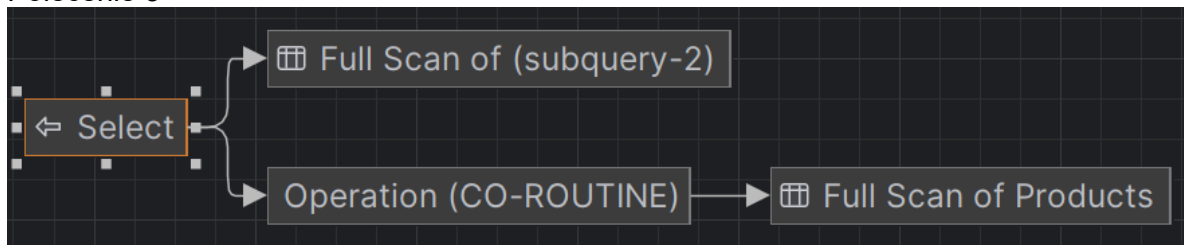
### Polecenie 1



Polecenie 2



Polecenie 3



Porównując wykonania na serwerach Postgres i SQLite, dla polecenia z użyciem podzapytania oraz joina czasy i w ogólności wykresy wyglądają podobnie, mamy taką samą liczbę skanowa tabeli, z której korzystamy. W przypadku funkcji okna wygląda to inaczej. Z analizy wynika, że sqlite rozbija w analizie wykonanie funkcji okna, poprzez co mamy dwa full scan, jedno z nich z co-routine ( podobne do podprogramu/współbieżne). Jednak czasy wykonania ostatecznie są podobne – ok. 100ms.

## Zadanie 4

Baza: Northwind, tabela products

Napisz polecenie, które zwraca: id produktu, nazwę produktu, cenę produktu, średnią cenę produktów w kategorii, do której należy dany produkt. Wyświetl tylko pozycje (produkty) których cena jest większa niż średnia cena.

Napisz polecenie z wykorzystaniem podzapytania, join'a oraz funkcji okna. Porównaj zapytania. Porównaj czasy oraz plany wykonania zapytań.

Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSQL, SQLite)

Polecenie 1 z wykorzystaniem podzapytania:

```

with tmp ( ProductID , ProductName,UnitPrice, AvgPrice)
as
(
    select
        p1.ProductID,
        p1.ProductName,
        p1.UnitPrice,
        (select avg(UnitPrice) as avg_price
         from Products p2
         where p2.CategoryID = p1.CategoryID
         group by CategoryID)
        from Products as p1
)
select *
from tmp
where AvgPrice < UnitPrice

```

Polecenie 2 z wykorzystaniem join'a

```

select
    p1.ProductID,
    p1.ProductName,
    p1.UnitPrice,
    avg(p2.UnitPrice) as avgprice
from Products as p1
join Products as p2 on p1.CategoryID = p2.CategoryID
group by p1.ProductID,
    p1.ProductName,
    p1.UnitPrice
having p1.UnitPrice > avg(p2.UnitPrice)

```

Polecenie 3 z wykorzystaniem funkcji okna.

```

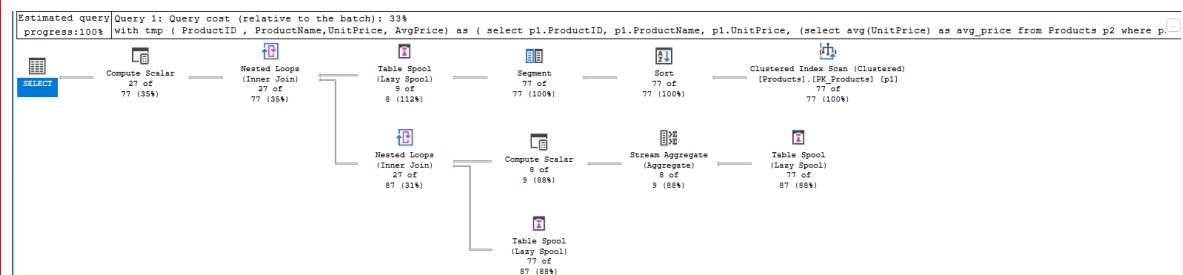
with tmp ( ProductID , ProductName,UnitPrice, AvgPrice)
as(
select
    p1.ProductID,
    p1.ProductName,
    p1.UnitPrice,
    avg(UnitPrice) over(partition by CategoryID) as AvgPrice
from Products as p1
)
select *
from tmp
where AvgPrice <UnitPrice

```

Porównanie czasów oraz planów wykonania zapytań

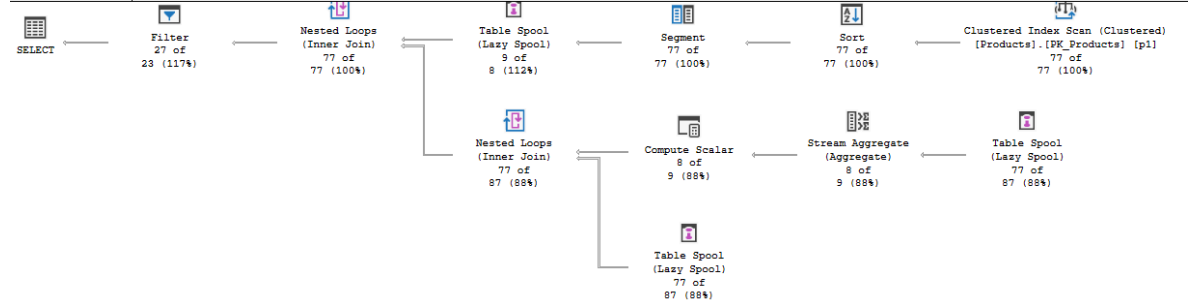
a) MS SQL Server

Polecenie 1



## Polecenie 2

Estimated query progress:100% Query 2: Query cost (relative to the batch): 33%  
/\* funkcja okna \*/;with tmp ( ProductID , ProductName,UnitPrice, AvgPrice) as( select p1.ProductID, p1.ProductName, p1.UnitPrice, a

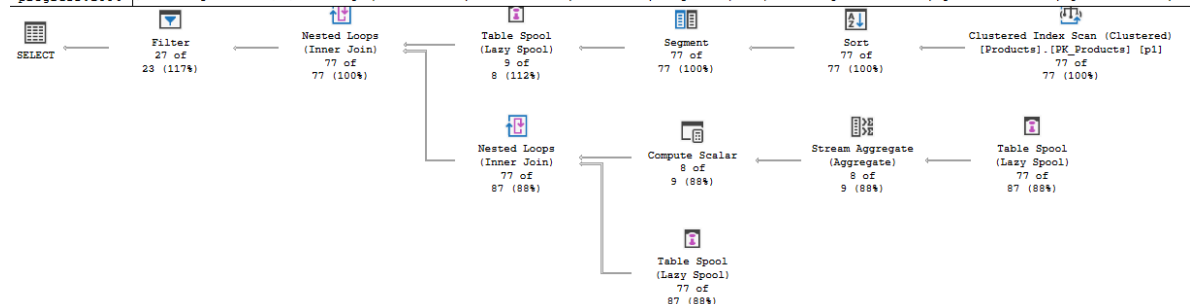


Estimated Subtree Cost: 0,0170431

Query cost (relative to the batch): 33%

## Polecenie 3

Estimated query progress:100% Query 3: Query cost (relative to the batch): 33%  
/\* funkcja okna \*/;with tmp ( ProductID , ProductName,UnitPrice, AvgPrice) as( select p1.ProductID, p1.ProductName, p1.UnitPrice, a



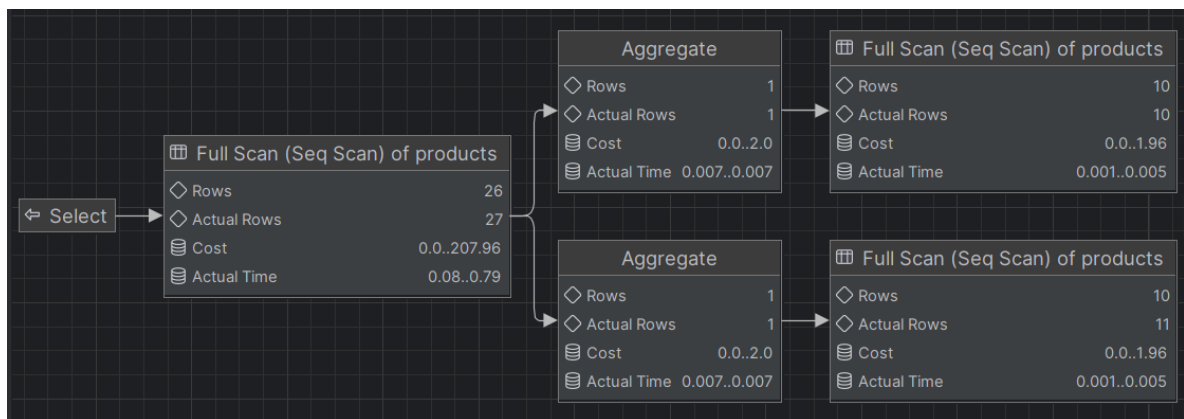
Estimated Subtree Cost: 0,0170431

Query cost (relative to the batch): 33%

## Porównanie:

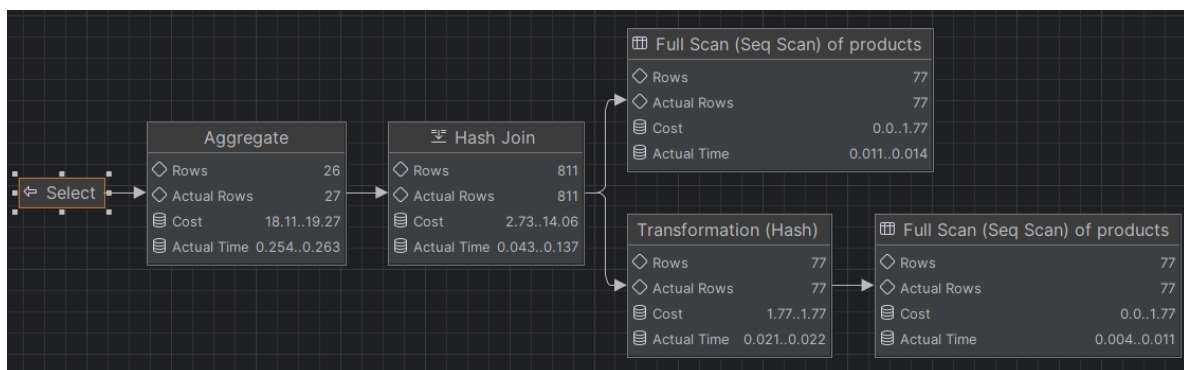
Najmniejszy koszt Estimated Subtree Cost=0,0170431 uzyskano dla polecenia 2 z wykorzystaniem join oraz polecenia 3 z wykorzystaniem funkcji okna. 3 najgorszy wynik Estimated Subtree Cost= 0,0170508 uzyskano dla polecenia 1 z użyciem podzapytania. Należy zaznaczyć, że w tym przypadku różnice są niewielkie. Świadczy o tym też Query cost (relative to the batch) pokazujący udziały poszczególnych poleceń w koszcie uruchomienia 3 poleceń naraz. Dla wszystkich poleceń jest taki sam (33%). Z obserwacji wynika, że najwydajniejszymi poleceniami są polecenie 2 z wykorzystaniem join oraz polecenie 3 z wykorzystaniem funkcji okna. Najgorszą wydajności cechuje się polecenie 1 z podzapytaniem.

b) PostgreSQL  
Polecenie 1



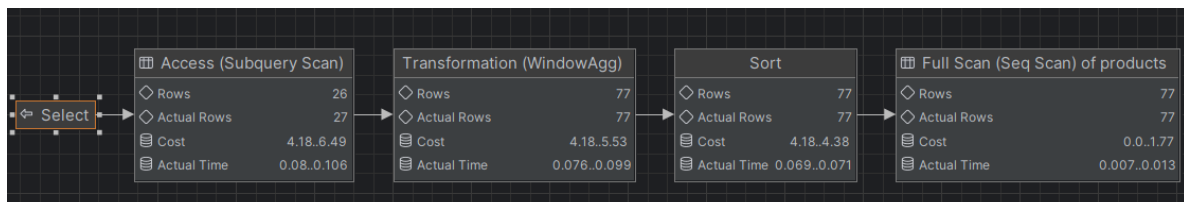
Cost=207.96

### Polecenie 2



Cost=18.11

### Polecenie 3

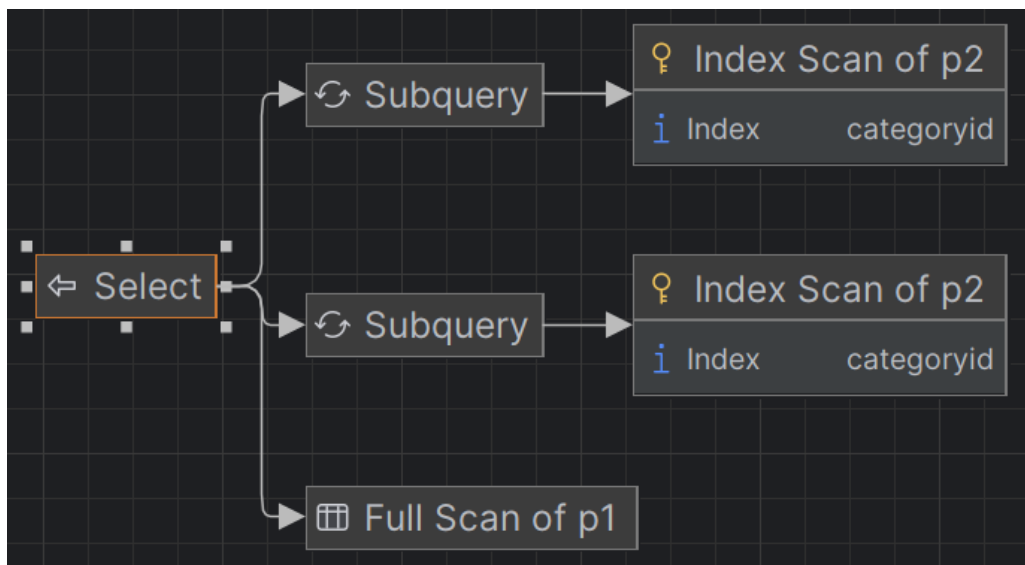


Cost=4.18

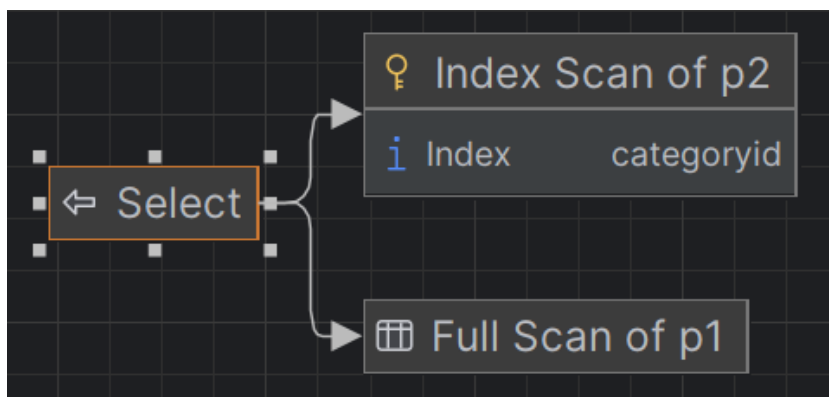
### Porównanie

Najmniejszy koszt Cost=0.0...207 uzyskano dla polecenia 1 z wykorzystaniem podzapytania. Kolejny wynik Cost=4.18 uzyskano dla polecenia 3 z funkcją okna. 3 najgorszy wynik Cost=18.11 uzyskano dla polecenia 2 z użyciem join. Z obserwacji wynika, że najwydajniejszym poleceniem jest polecenie 1 z wykorzystaniem podzapytania. Drugim wydajnym poleceniem jest polecenie 3 z funkcją okna. Najgorszą wydajności cechuje się polecenie 3 z join.

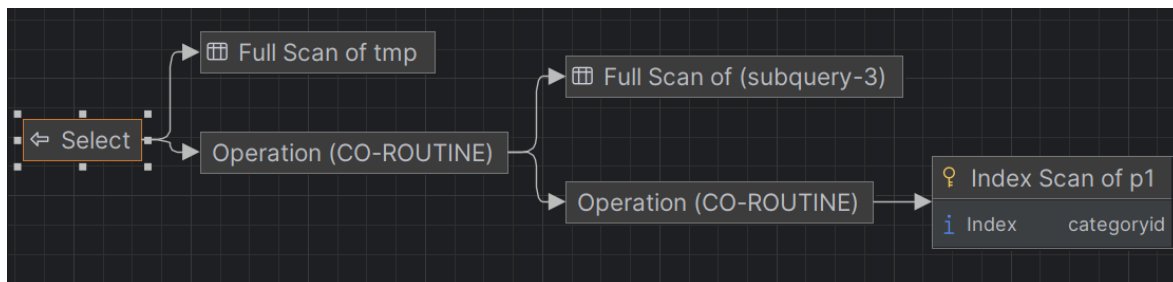
c) SQLite  
Polecenie 1



Polecenie 2



Polecenie 3



W wykonaniach na serwerze postgres zauważalny jest dużo większy koszt dla wersji z wykorzystaniem podzapytania – osiąga on aż ok. 207. Jest to prawdopodobnie spowodowane ilością wykonanych full scan i agregowania tych wyników. W SQLite analiza wygląda podobnie – wykonanie jest zbliżone. Dla wersji z join z diagramu dla Postgres widzimy, że korzysta on z hash, możliwe że dzięki temu taka wersja jest szybsza od wersji wcześniejszej – koszt ok. 18. Jeśli chodzi o wykonanie z funkcją okna jest ono najszybsze dla wszystkich serwerów, przy czym w ssms ta różnica nie jest aż tak duża a także koszty wykonania są podobne.



## Zadanie 5 - przygotowanie

Baza: Northwind

Tabela products zawiera tylko 77 wiersz. Warto zaobserwować działanie na większym zbiorze danych.

Wygeneruj tabelę zawierającą kilka milionów (kilkaset tys.) wierszy

Stwórz tabelę o następującej strukturze:

Skrypt dla SQL Sriver

```
create table product_history(  
    id int identity(1,1) not null,  
    productid int,  
    productname varchar(40) not null,  
    supplierid int null,  
    categoryid int null,  
    quantityperunit varchar(20) null,  
    unitprice decimal(10,2) null,  
    quantity int,  
    value decimal(10,2),  
    date date,  
    constraint pk_product_history primary key clustered  
        (id asc )  
)
```

Wygeneruj przykładowe dane:

Dla 30000 iteracji, tabela będzie zawierała nieco ponad 2mln wierszy (dostostu ograniczenie do możliwości swojego komputera)

Skrypt dla SQL Sriver

```
declare @i int  
set @i = 1  
while @i <= 30000  
begin  
    insert product_history  
        select productid, ProductName, SupplierID, CategoryID,  
            QuantityPerUnit, round(RAND()*unitprice + 10,2),  
            cast(RAND() * productid + 10 as int), 0,  
            dateadd(day, @i, '1940-01-01')  
        from products  
    set @i = @i + 1;  
end;  
  
update product_history  
set value = unitprice * quantity
```

```
where 1=1;
```

### Skrypt dla Postgresql

```
create table product_history(  
    id int generated always as identity not null  
        constraint pkproduct_history  
            primary key,  
    productid int,  
    productname varchar(40) not null,  
    supplierid int null,  
    categoryid int null,  
    quantityperunit varchar(20) null,  
    unitprice decimal(10,2) null,  
    quantity int,  
    value decimal(10,2),  
    date date  
);
```

Wygeneruj przykładowe dane:

### Skrypt dla Postgresql

```
do $$  
begin  
    for cnt in 1..30000 loop  
        insert into product_history(productid, productname, supplierid,  
            categoryid, quantityperunit,  
            unitprice, quantity, value, date)  
        select productid, productname, supplierid, categoryid,  
            quantityperunit,  
            round((random()*unitprice + 10)::numeric,2),  
            cast(random() * productid + 10 as int), 0,  
            cast('1940-01-01' as date) + cnt  
        from products;  
    end loop;  
end; $$;  
  
update product_history  
set value = unitprice * quantity  
where 1=1;
```

Wykonaj polecenia: select count(\*) from product\_history, potwierdzające wykonanie zadania

Wynik polecenia dla SQL Server

| Results          |         | Messages |
|------------------|---------|----------|
| (No column name) |         |          |
| 1                | 2310000 |          |

Wynik polecenia dla Postgresql

| count |         |
|-------|---------|
| 1     | 2310000 |

## Zadanie 6

Baza: Northwind, tabela product\_history

To samo co w zadaniu 3, ale dla większego zbioru danych

Napisz polecenie, które zwraca: id pozycji, id produktu, nazwę produktu, cenę produktu, średnią cenę produktów w kategorii do której należy dany produkt. Wyświetl tylko pozycje (produkty) których cena jest większa niż średnia cena.

Napisz polecenie z wykorzystaniem podzapytania, join'a oraz funkcji okna. Porównaj zapytania. Porównaj czasy oraz plany wykonania zapytań.

Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSQL, SQLite)

Polecenie 1 z wykorzystaniem podzapytania:

```
;with tmp ( ProductID , ProductName,UnitPrice, AvgPrice)
as
(
    select distinct
        p1.ProductID,
        p1.ProductName,
        p1.UnitPrice,
        ( select avg(UnitPrice) as avg_price
          from product_history p2
         where p2.CategoryID = p1.CategoryID group by CategoryID
        )
    from product_history as p1
)
select *
from tmp
where AvgPrice < UnitPrice
```

Polecenie 2 z wykorzystaniem join'a

```
Select distinct
    p1.ProductID,
    p1.ProductName,
    p1.UnitPrice,
    avg(p2.UnitPrice) as avgprice
from product_history as p1
join product_history as p2 on p1.CategoryID = p2.CategoryID
group by p1.ProductID, p1.ProductName,p1.UnitPrice
having p1.UnitPrice > avg(p2.UnitPrice)
```

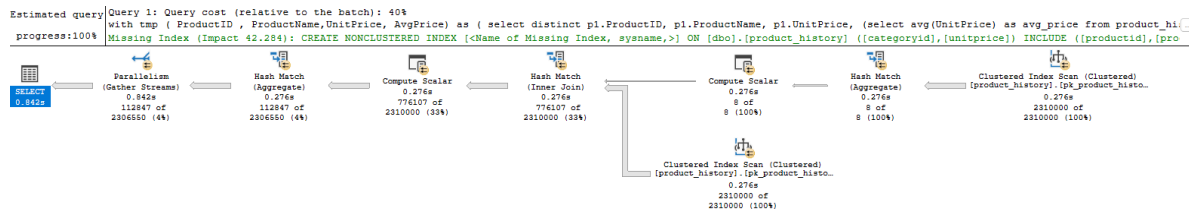
Polecenie 3 z wykorzystaniem funkcji okna.

```
;with tmp ( ProductID , ProductName,UnitPrice, AvgPrice)
as(
select distinct
    p1.ProductID,
    p1.ProductName,
    p1.UnitPrice,
    avg(UnitPrice) over(partition by CategoryID) as AvgPrice
from product_history as p1
)
select *
from tmp
where AvgPrice < UnitPrice
```

Porównanie czasów oraz planów wykonania zapytań

a) MS SQL Server

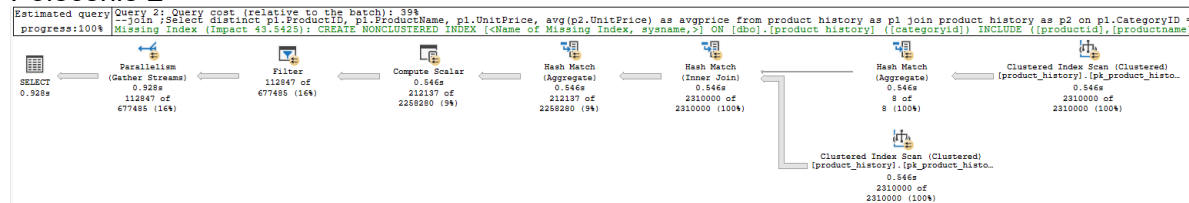
Polecenie 1



Estimated Subtree Cost: 42,4808

Query cost (relative to the batch): 40%

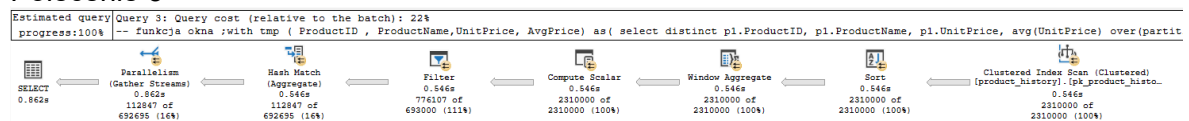
Polecenie 2



Estimated Subtree Cost: 41,2529

Query cost (relative to the batch): 39%

Polecenie 3



Estimated Subtree Cost: 23,2506

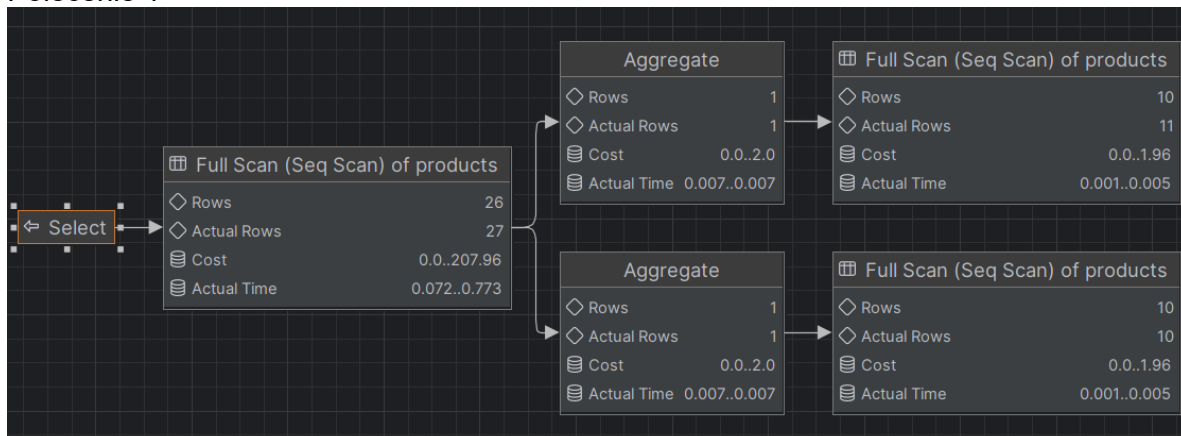
Query cost (relative to the batch): 22%

Porównanie:

Najmniejszy koszt Estimated Subtree Cost=23,2506 uzyskano dla polecenia 3 z wykorzystaniem funkcji okna. 2 wynik Estimated Subtree Cost=41,25293 uzyskano dla polecenia 2 z join. Najgorszy wynik Estimated Subtree Cost= 42,4808 uzyskano dla polecenia 1 z użyciem podzapytania. Należy zaznaczyć, że w tym przypadku różnice są niewielkie. Świadczy o tym też Query cost (relative to the batch) pokazujący udziały poszczególnych poleceń w koszcie uruchomienia 3 poleceń naraz. Dla polecenia 1 i 2 jest podobny odpowiednio 40% i 39% Z obserwacji wynika, że najwydajniejszym poleceniem jest polecenie 3 z wykorzystaniem funkcji okna. Najgorszą wydajnością cechuje się polecenie 1 z podzapytaniem.

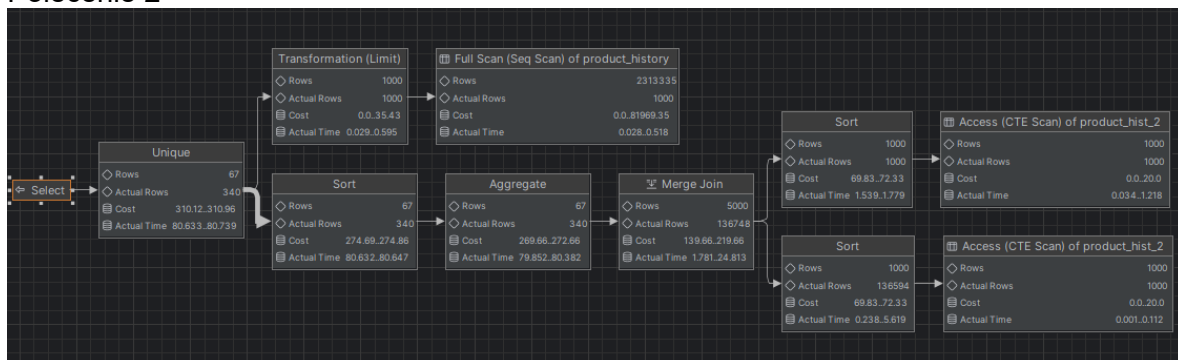
## b) PostgreSQL

### Polecenie 1



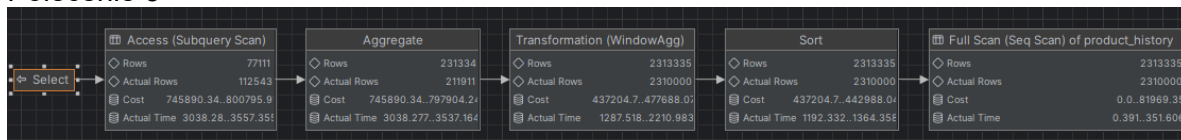
Cost=0.0..207

### Polecenie 2



Cost=310.12

### Polecenie 3



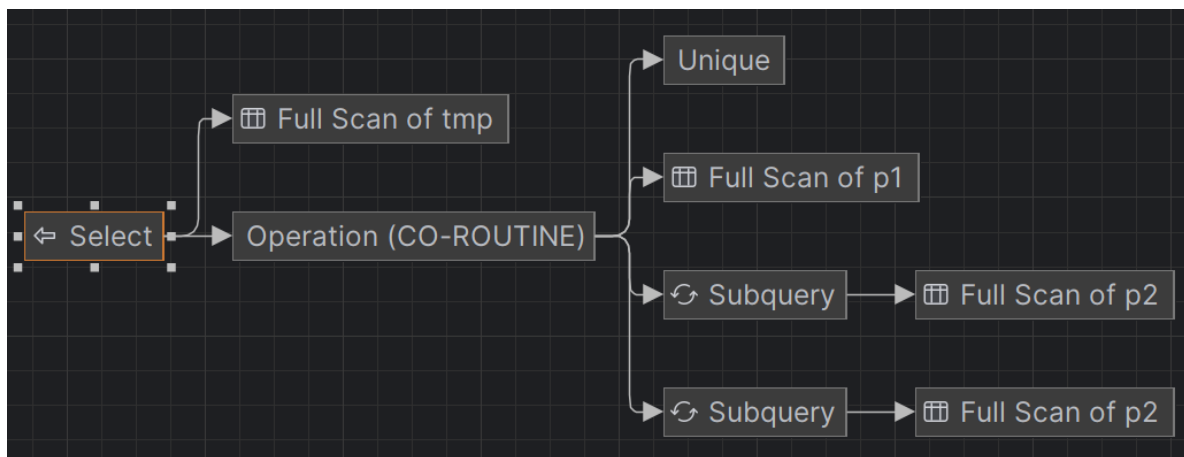
Cost=745890.34

## Porównanie

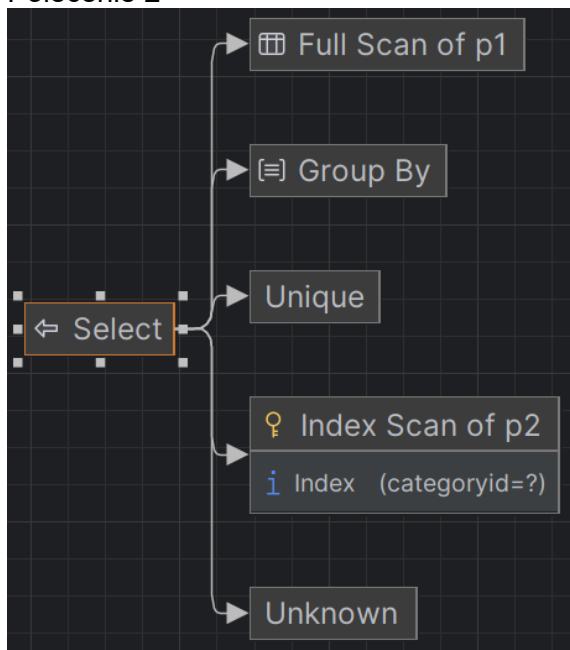
Najmniejszy koszt Cost=0.0...207 uzyskano dla polecenia 1 z wykorzystaniem podzapytania. Kolejny wynik 310.12 uzyskano dla polecenia 2 z joinem. 3 najgorszy wynik Cost= 745890.34 uzyskano dla polecenia 3 z użyciem przesuwanego okna. Z obserwacji wynika, że najwydajniejszym poleceniem jest polecenie 1 z wykorzystaniem podzapytania. Drugim wydajnym poleceniem jest polecenie 2 z join. Zaskakująco najgorszą wydajności cechuje się polecenie 3 z przesuwanym oknem.

## c) SQLite

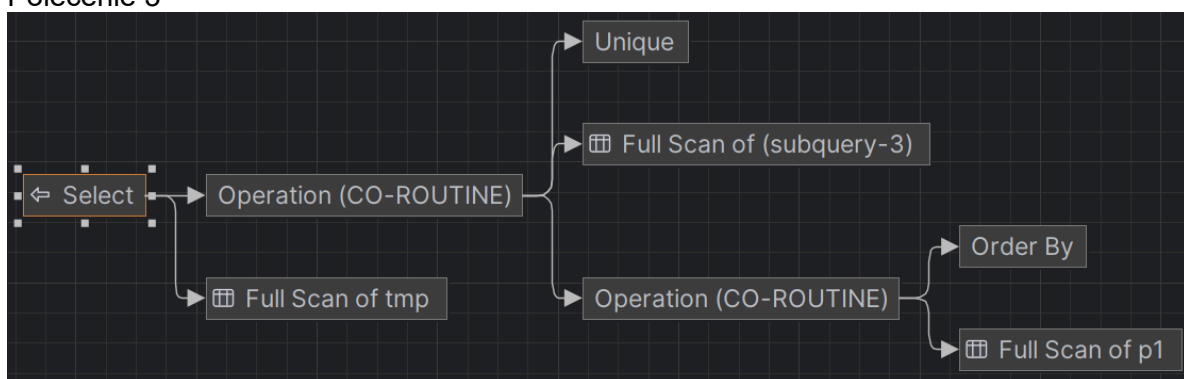
### Polecenie 1



Polecenie 2



Polecenie 3



Porównując wykonania na wszystkich serwerach można zauważyć znów inną strukturę w analizie dla funkcji okna na serwerze SQLite – jest ona rozbita zapewne na poszczególne operacje. Dla serwerów SQLite i Postgres widać dużo gorszą wydajność jeśli chodzi o zapytania z wykorzystaniem join lub podzapytania.

## Zadanie 7

Baza: Northwind, tabela product\_history

Lekka modyfikacja poprzedniego zadania

Napisz polecenie, które zwraca: id pozycji, id produktu, nazwę produktu, cenę produktu oraz

- średnią cenę produktów w kategorii do której należy dany produkt.
- łączną wartość sprzedaży produktów danej kategorii (suma dla pola value)
- średnią cenę danego produktu w roku którego dotyczy dana pozycja
- łączną wartość sprzedaży produktów danej kategorii (suma dla pola value)

Napisz polecenie z wykorzystaniem podzapytania, join'a oraz funkcji okna. Porównaj zapytania. W przypadku funkcji okna spróbuj użyć klauzuli WINDOW.

Porównaj czasy oraz plany wykonania zapytań.

Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSQL, SQLite)

Polecenie 1 z wykorzystaniem podzapytania

```
;select
    p1.id,
    p1.productid,
    p1.productname,
    p1.unitprice,
    ( select avg(unitprice)
      from product_history
     where categoryid = p1.categoryid
     group by categoryid
    ) as avgprice,
    (
      select avg(value)
      from product_history
     where categoryid = p1.categoryid
     group by categoryid
    ) as sumprice,
    (
      select avg(unitprice) as avg_price_year
      from product_history
     where year(p1.date) = year(date) and p1.productid = productid
     group by year(date), productid
    ) as avgpriceyearly
from product_history as p1
```

Polecenie 2 z wykorzystaniem join'a

Próba wykonania tego zapytania bez użycia CTE, kończyła się niepowodzeniem – długi czas wykonania + przekroczenie pamięci – ze względu na ilość rekordów w tabeli product\_history.

```
;with tmp ( categoryId , avgprice, sumprice )
as(
select
    categoryid,
    avg(unitprice) as avgprice,
    sum(value) as sumprice
  from product_history
  group by categoryid
),
tmp2 (productid , year_date ,avgpriceyearly )
```

```

as(
select
    productid,
    year(date),
    avg(unitprice) as avgpriceyearly
from product_history as p1
group by productid , year(date)
)
select p1.id,
    p1.productid,
    p1.productname,
    t.avgprice,
    t.sumprice
from product_history as p1
join tmp t on p1.categoryid = t.categoryId
join tmp2 t2 on p1.productid = t2.productid and year(p1.date) = t2.year_date

```

Polecenie 3 z wykorzystaniem funkcji okna.

```

select distinct
    p1.id,
    p1.productid,
    p1.productname,
    p1.unitprice,
    avg(UnitPrice) over(partition by CategoryId) as avgprice,
    sum(p1.value) over(partition by CategoryId) as sumprice,
    avg(p1.unitprice) over(partition by ProductId , year(date))
as avgpriceyearly
from product_history as p1

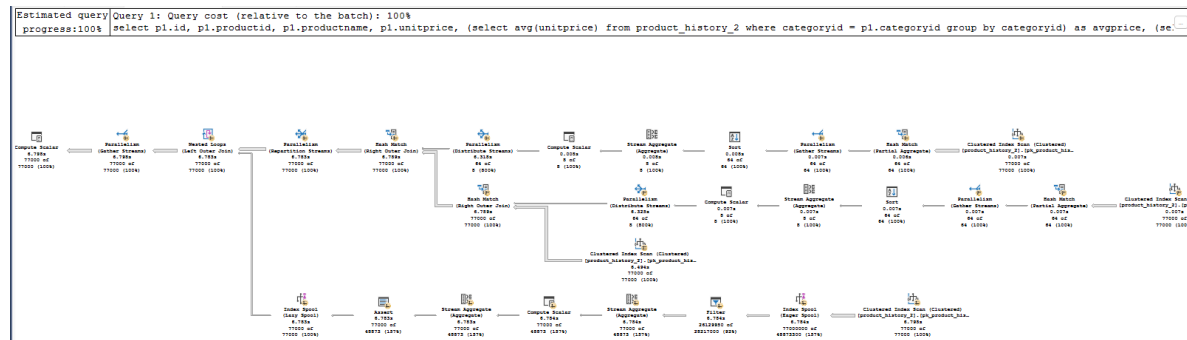
```

Porównanie czasów oraz planów wykonania zapytań

a) MS SQL Server

Polecenie 1

Zbyt długo się liczyło, więc stworzono tymczasową tabelę - product\_history\_2 dla 1000 rekordów



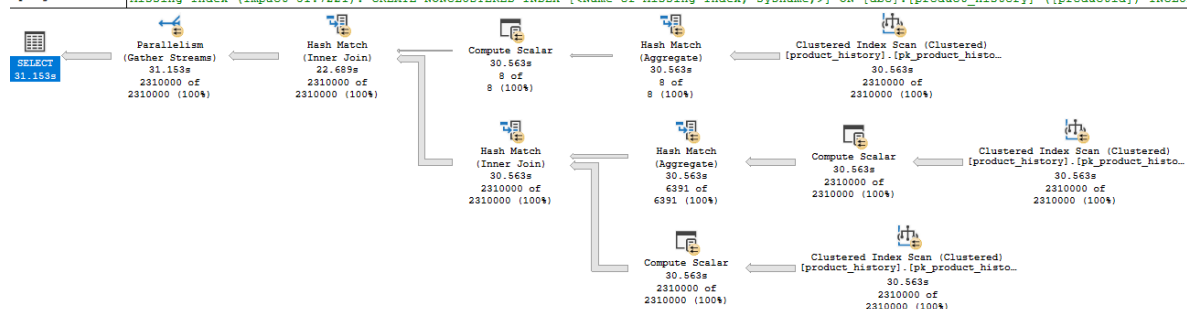
Estimated Subtree Cost: 154,29

Polecenie 2

Estimated query Query 1: Query cost (relative to the batch): 100%

progress:100%

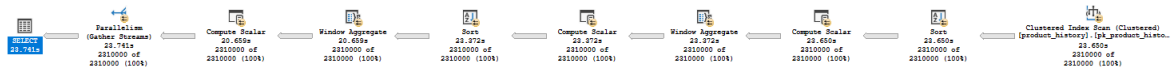
Missing Index (Impact 31.7221): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname>] ON [dbo].[product\_history] ([productid]) INCLU





Estimated Subtree Cost: 60,5111

### Polecenie 3



Estimated Subtree Cost: 41,3068

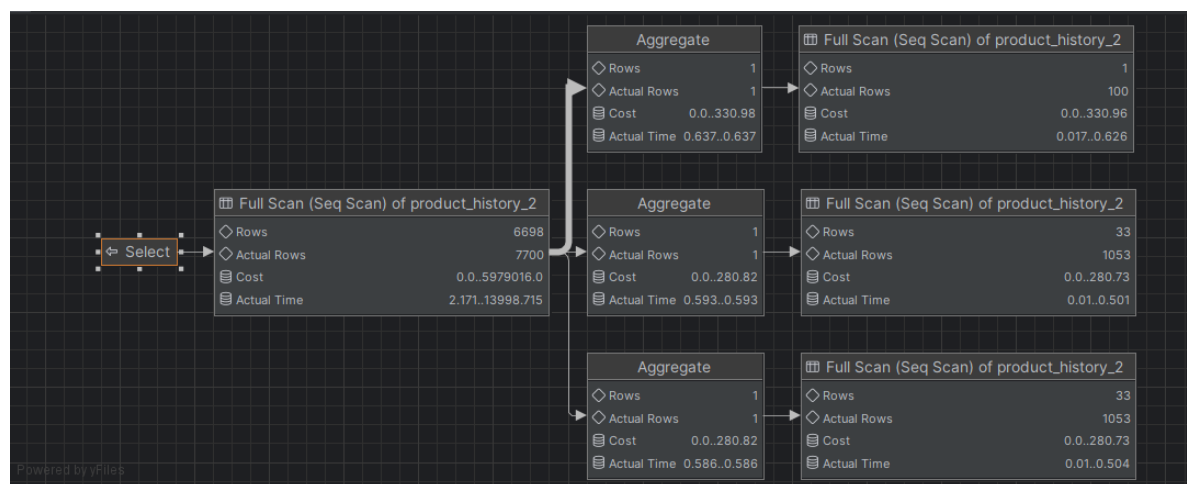
### Porównanie

Najmniejszy koszt Estimated Subtree Cost=41,3068 uzyskano dla polecenia 3 z wykorzystaniem funkcji okna. 2 wynik Estimated Subtree Cost=60,5111uzyskano dla polecenia 2 z join. Najgorszy wynik Estimated Subtree Cost= 154,29 uzyskano dla polecenia 1 z użyciem podzapytania dla product\_history\_2 z 1000 rekordów. Z obserwacji wynika, że najwydajniejszym poleceniem jest polecenie 3 z wykorzystaniem funkcji okna. Najgorszą wydajnością cechuje się polecenie 1 z podzapytaniem.

### b) PostgreSQL

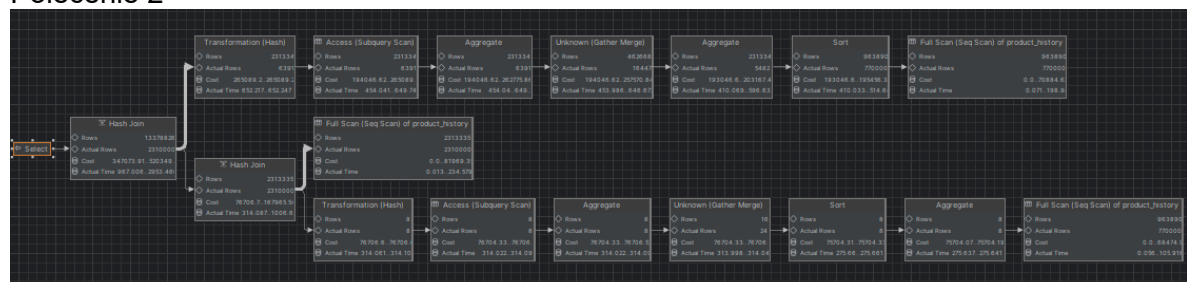
#### Polecenie 1

Cost=brak danych, ponieważ zbyt długo się liczyło. Stworzono tymczasową tabelę product\_history\_2 dla 100 rekordów



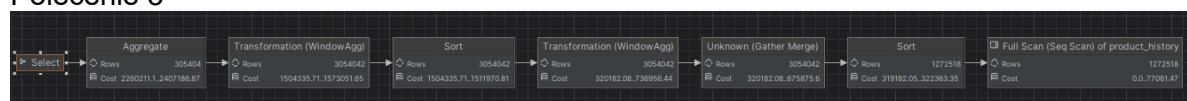
Cost = 5979016.0

#### Polecenie 2



Cost=347073,91

#### Polecenie 3



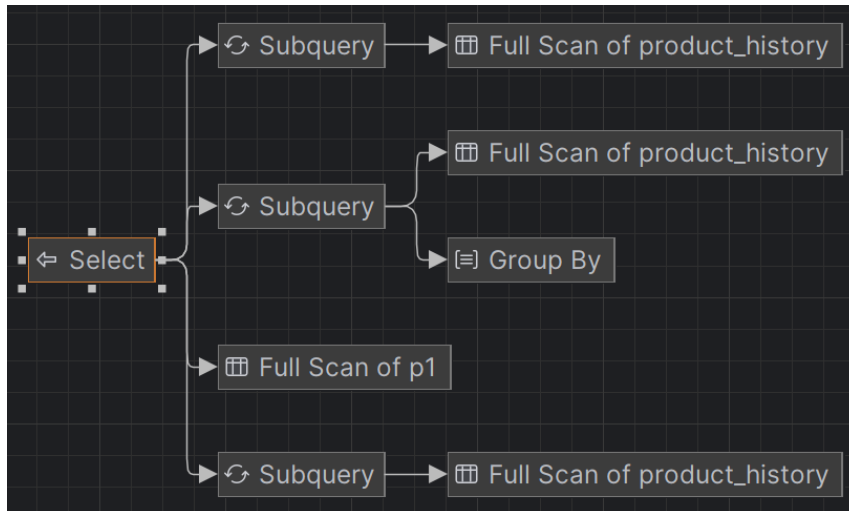
Cost=2260211,1

### Porównanie

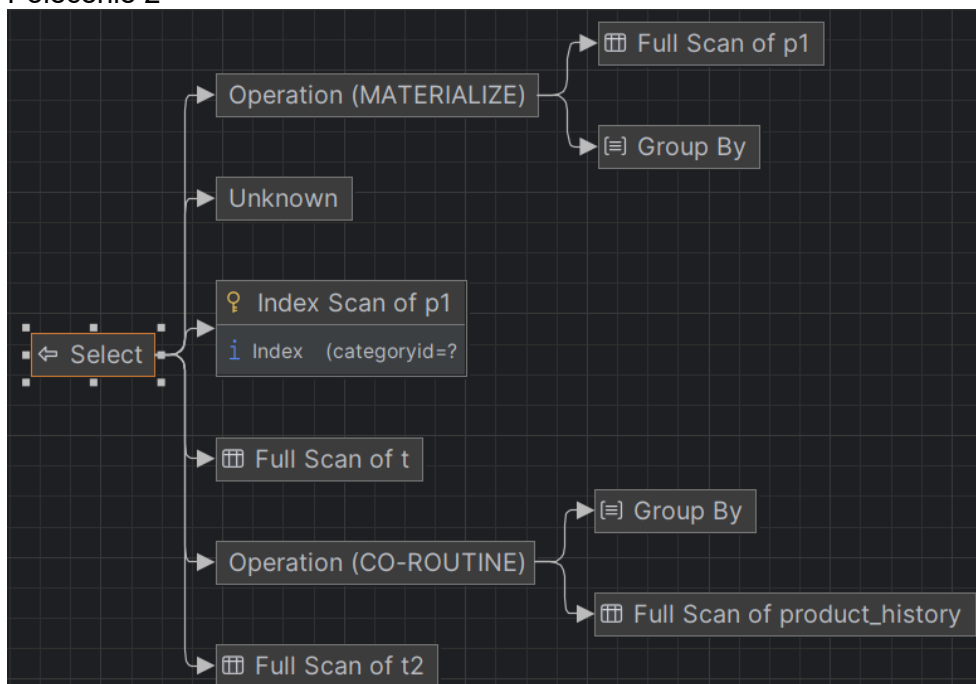
Najmniejszy koszt  $\text{Cost}=347073,91$  uzyskano dla polecenia 2 z wykorzystaniem join oraz polecenia 3 z funkcją okna. 3 najgorszy wynik  $\text{Cost}=0.0...59$  dla `product_history_2` z 100 rekordami ( dla oryginału był zbyt długi czas liczenia) uzyskano dla polecenia 1 z użyciem podzapytania. Z obserwacji wynika, że najwydajniejszym poleceniem jest polecenie 2 z join oraz polecenie 3 z funkcją okna. Najgorszą wydajności cechuje się polecenie 1 z podzapytaniem.

#### c) SQLite

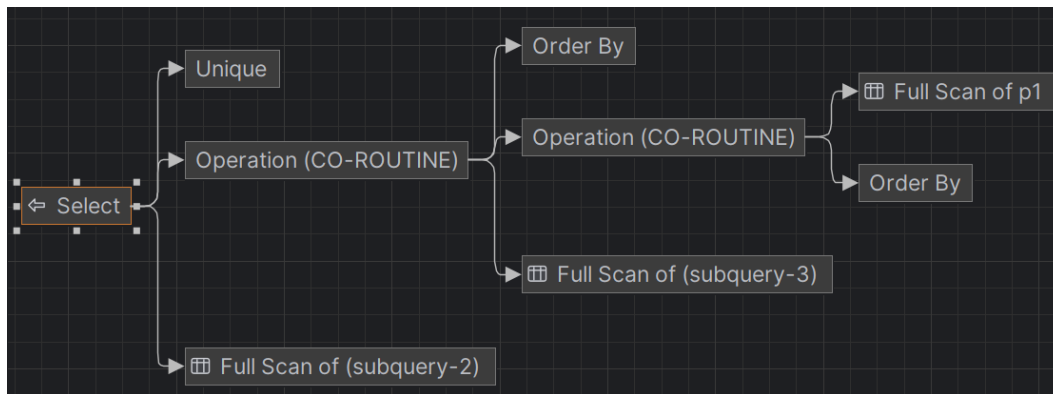
##### Polecenie 1



##### Polecenie 2



##### Polecenie 3



Dla wszystkich serwerów najgorszej wypada wersja z wykorzystaniem podzapytania. Co ciekawe pomimo bardziej złożonej struktury operacji dla wersji z joinem w przypadku Postgres i SQLite. Wpływ na to mają zapewne wielokrotne przetwarzania całej tabeli – co ciekawe 3 w przypadku postgres, a 4 w przypadku SQLite i SSMS.

## Zadanie 8 - obserwacja

Funkcje rankingu, `row_number()`, `rank()`, `dense_rank()`

Wykonaj polecenie, zaobserwuj wynik. Porównaj funkcje `row_number()`, `rank()`, `dense_rank()`

```

select productid, productname, unitprice, categoryid,
       row_number() over(partition by categoryid order by unitprice desc)
                           as rowno,
       rank() over(partition by categoryid order by unitprice desc)
                           as rankprice,
       dense_rank() over(partition by categoryid order by unitprice desc)
                           as denserankprice
from products;
  
```

Wynik wykonania polecenia

| Results |           | Messages                     |           |            |       |           |                |
|---------|-----------|------------------------------|-----------|------------|-------|-----------|----------------|
|         | productid | productname                  | unitprice | categoryid | rowno | rankprice | denserankprice |
| 1       | 38        | Côte de Blaye                | 263,50    | 1          | 1     | 1         | 1              |
| 2       | 43        | Iph Coffee                   | 46,00     | 1          | 2     | 2         | 2              |
| 3       | 2         | Chang                        | 19,00     | 1          | 3     | 3         | 3              |
| 4       | 1         | Chai                         | 18,00     | 1          | 4     | 4         | 4              |
| 5       | 39        | Chartreuse verte             | 18,00     | 1          | 5     | 4         | 4              |
| 6       | 35        | Steeleye Stout               | 18,00     | 1          | 6     | 4         | 4              |
| 7       | 76        | Lakkalikööri                 | 18,00     | 1          | 7     | 4         | 4              |
| 8       | 70        | Outback Lager                | 15,00     | 1          | 8     | 8         | 5              |
| 9       | 67        | Laughing Lumberjack Lager    | 14,00     | 1          | 9     | 9         | 6              |
| 10      | 34        | Sasquatch Ale                | 14,00     | 1          | 10    | 9         | 6              |
| 11      | 75        | Rhönbräu Klosterbier         | 7,75      | 1          | 11    | 11        | 7              |
| 12      | 24        | Guaraná Fantástica           | 4,50      | 1          | 12    | 12        | 8              |
| 13      | 63        | Veggie-spread                | 43,90     | 2          | 1     | 1         | 1              |
| 14      | 8         | Northwoods Cranberry Sauce   | 40,00     | 2          | 2     | 2         | 2              |
| 15      | 61        | Sirup d'érable               | 28,50     | 2          | 3     | 3         | 3              |
| 16      | 6         | Grandma's Boysenberry Spread | 25,00     | 2          | 4     | 4         | 4              |
| 17      | 4         | Chef Anton's Cajun Seasoning | 22,00     | 2          | 5     | 5         | 5              |
| 18      | 5         | Chef Anton's Gumbo Mix       | 21,35     | 2          | 6     | 6         | 6              |

row\_number() – przypisuje każdemu rekordowi unikalny numer wiersza w kolumnie rowno. Numer wiersza jest resetowany dla każdej partycji, w przypadku użycia partition by. W kolumnie rowno partycja odbywa się po klasach produktów.

rank() – służy do nadania każdemu rekordowi unikalnej rangi na podstawie określonej wartości w tym wypadku wartości z kolumny unitprice. Jeśli rekordy mają tę samą wartość np. unitprice równe 18,00 , funkcja przypisze tę samą rangę tym rekordom, pomijając kolejną rangę. Efekt jest widoczny w kolumnie rankprice.

dense\_rank() – w przeciwieństwie do rank() ta funkcja nie pomija żadnej rangi. Oznacza to, że jeśli zostaną znalezione identyczne rekordy, gdzie np. unitprice jest równe 18,00, funkcja przypisze tę samą rangę tym rekordom, ale nie pominie kolejnej rangi.

## Zadanie

Spróbuj uzyskać ten sam wynik bez użycia funkcji okna

```
SELECT
  p.productid,
  p.productname,
  p.unitprice,
  p.categoryid,
  (SELECT COUNT(*)
   FROM products p4
   WHERE p4.categoryid = p.categoryid AND p4.unitprice >= p.unitprice) as rowno,
  (SELECT COUNT(*) + 1
   FROM products p2
   WHERE p2.categoryid = p.categoryid AND p2.unitprice > p.unitprice) as rankprice,
  (SELECT COUNT(DISTINCT p3.unitprice)
   FROM products p3
   WHERE p3.categoryid = p.categoryid AND p3.unitprice >= p.unitprice) as denserankprice
FROM
  products p
ORDER BY
  p.categoryid, p.unitprice DESC;
```

Wynik:

|    | productid | productname                  | unitprice | categoryid | rowno | rankprice | denserankprice |
|----|-----------|------------------------------|-----------|------------|-------|-----------|----------------|
| 1  | 38        | Côte de Blaye                | 263,50    | 1          | 1     | 1         | 1              |
| 2  | 43        | Ipoh Coffee                  | 46,00     | 1          | 2     | 2         | 2              |
| 3  | 2         | Chang                        | 19,00     | 1          | 3     | 3         | 3              |
| 4  | 1         | Chai                         | 18,00     | 1          | 7     | 4         | 4              |
| 5  | 39        | Chartreuse verte             | 18,00     | 1          | 7     | 4         | 4              |
| 6  | 35        | Steeleye Stout               | 18,00     | 1          | 7     | 4         | 4              |
| 7  | 76        | Lakkalikööri                 | 18,00     | 1          | 7     | 4         | 4              |
| 8  | 70        | Outback Lager                | 15,00     | 1          | 8     | 8         | 5              |
| 9  | 67        | Laughing Lumberjack Lager    | 14,00     | 1          | 10    | 9         | 6              |
| 10 | 34        | Sasquatch Ale                | 14,00     | 1          | 10    | 9         | 6              |
| 11 | 75        | Rhönbräu Klosterbier         | 7,75      | 1          | 11    | 11        | 7              |
| 12 | 24        | Guaraná Fantástica           | 4,50      | 1          | 12    | 12        | 8              |
| 13 | 63        | Vegie-spread                 | 43,90     | 2          | 1     | 1         | 1              |
| 14 | 8         | Northwoods Cranberry Sauce   | 40,00     | 2          | 2     | 2         | 2              |
| 15 | 61        | Sirop d'érable               | 28,50     | 2          | 3     | 3         | 3              |
| 16 | 6         | Grandma's Boysenberry Spread | 25,00     | 2          | 4     | 4         | 4              |
| 17 | 4         | Chef Anton's Cajun Seasoning | 22,00     | 2          | 5     | 5         | 5              |
| 18 | 5         | Chef Anton's Gumbo Mix       | 21,35     | 2          | 6     | 6         | 6              |

## Zadanie 9

Baza: Northwind, tabela product\_history

Dla każdego produktu, podaj 4 najwyższe ceny tego produktu w danym roku. Zbiór wynikowy powinien zawierać:

rok

id produktu

nazwę produktu

cenę

datę (datę uzyskania przez produkt takiej ceny)

pozycję w rankingu

Uporządkuj wynik wg roku, nr produktu, pozycji w rankingu

Polecenie 1

```
with r as (
    select
        year(date) as year_date,
        productid,
        productname,
        unitprice,
        date,
        row_number() over(partition by productid, year(date) order by unitprice
desc) as pricerank
    from product_history
)
select year_date, productid, productname, unitprice, date, pricerank from r
where pricerank <= 4
order by year_date, productid, pricerank;
```

Wynik:

|    | year_date | productid | productname                  | unitprice | date       | pricerank |
|----|-----------|-----------|------------------------------|-----------|------------|-----------|
| 1  | 1940      | 1         | Chai                         | 27.98     | 1940-07-13 | 1         |
| 2  | 1940      | 1         | Chai                         | 27.95     | 1940-11-17 | 2         |
| 3  | 1940      | 1         | Chai                         | 27.92     | 1940-06-20 | 3         |
| 4  | 1940      | 1         | Chai                         | 27.87     | 1940-06-21 | 4         |
| 5  | 1940      | 2         | Chang                        | 28.98     | 1940-07-13 | 1         |
| 6  | 1940      | 2         | Chang                        | 28.95     | 1940-11-17 | 2         |
| 7  | 1940      | 2         | Chang                        | 28.92     | 1940-06-20 | 3         |
| 8  | 1940      | 2         | Chang                        | 28.86     | 1940-06-21 | 4         |
| 9  | 1940      | 3         | Aniseed Syrup                | 19.99     | 1940-07-13 | 1         |
| 10 | 1940      | 3         | Aniseed Syrup                | 19.97     | 1940-11-17 | 2         |
| 11 | 1940      | 3         | Aniseed Syrup                | 19.96     | 1940-06-20 | 3         |
| 12 | 1940      | 3         | Aniseed Syrup                | 19.93     | 1940-06-21 | 4         |
| 13 | 1940      | 4         | Chef Anton's Cajun Seasoning | 31.98     | 1940-07-13 | 1         |
| 14 | 1940      | 4         | Chef Anton's Cajun Seasoning | 31.94     | 1940-11-17 | 2         |
| 15 | 1940      | 4         | Chef Anton's Cajun Seasoning | 31.90     | 1940-06-20 | 3         |
| 16 | 1940      | 4         | Chef Anton's Cajun Seasoning | 31.84     | 1940-06-21 | 4         |
| 17 | 1940      | 5         | Chef Anton's Gumbo Mix       | 31.33     | 1940-07-13 | 1         |

Spróbuj uzyskać ten sam wynik bez użycia funkcji okna, porównaj wyniki, czasy i plany zapytań. Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSQL, SQLite)

#### Polecenie 2

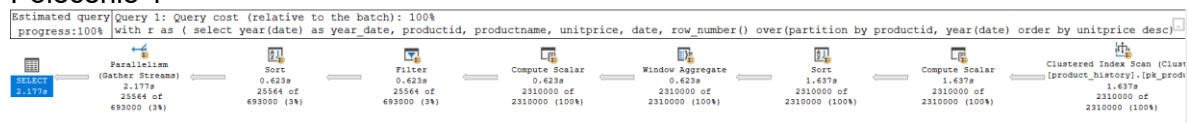
```
with r as (
    select
        year(p1.date) as year_date,
        p1.productid,
        p1.productname,
        p1.unitprice,
        p1.date,
        count(distinct p2.unitprice) as pricerank
    from product_history as p1
    inner join product_history as p2 on year(p1.date) = year(p2.date)
        and p1.productid = p2.productid
        and p2.unitprice >= p1.unitprice
    group by year(p1.date), p1.productid, p1.productname, p1.unitprice, p1.date
)
select year_date, productid, productname, unitprice, date, pricerank from r
where pricerank <= 4
order by year_date, productid, pricerank
```

|    | year_date | productid | productname                  | unitprice | date       | pricerank |
|----|-----------|-----------|------------------------------|-----------|------------|-----------|
| 1  | 1940      | 1         | Chai                         | 27.98     | 1940-07-13 | 1         |
| 2  | 1940      | 1         | Chai                         | 27.95     | 1940-11-17 | 2         |
| 3  | 1940      | 1         | Chai                         | 27.92     | 1940-06-20 | 3         |
| 4  | 1940      | 1         | Chai                         | 27.87     | 1940-06-21 | 4         |
| 5  | 1940      | 2         | Chang                        | 28.98     | 1940-07-13 | 1         |
| 6  | 1940      | 2         | Chang                        | 28.95     | 1940-11-17 | 2         |
| 7  | 1940      | 2         | Chang                        | 28.92     | 1940-06-20 | 3         |
| 8  | 1940      | 2         | Chang                        | 28.86     | 1940-06-21 | 4         |
| 9  | 1940      | 3         | Aniseed Syrup                | 19.99     | 1940-07-13 | 1         |
| 10 | 1940      | 3         | Aniseed Syrup                | 19.97     | 1940-11-17 | 2         |
| 11 | 1940      | 3         | Aniseed Syrup                | 19.96     | 1940-06-20 | 3         |
| 12 | 1940      | 3         | Aniseed Syrup                | 19.93     | 1940-06-21 | 4         |
| 13 | 1940      | 4         | Chef Anton's Cajun Seasoning | 31.98     | 1940-07-13 | 1         |
| 14 | 1940      | 4         | Chef Anton's Cajun Seasoning | 31.94     | 1940-11-17 | 2         |
| 15 | 1940      | 4         | Chef Anton's Cajun Seasoning | 31.90     | 1940-06-20 | 3         |
| 16 | 1940      | 4         | Chef Anton's Cajun Seasoning | 31.84     | 1940-06-21 | 4         |

Porównanie czasów oraz planów wykonania zapytań

a) MS SQL Server

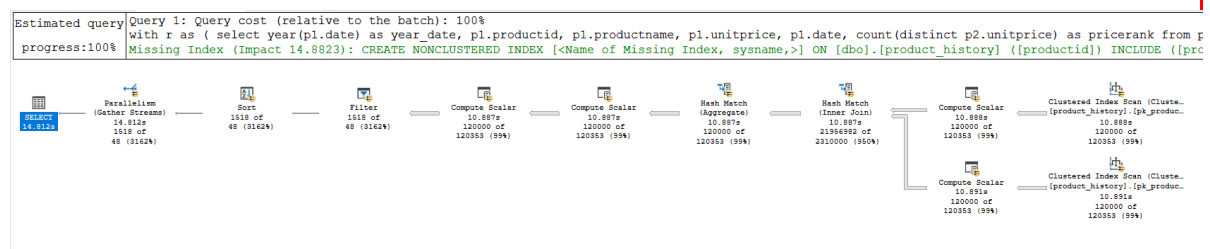
Polecenie 1



Estimated Subtree Cost: 31,6059

Polecenie 2

Polecenie 2 bez funkcji okna dla wszystkich rekordów z tabeli product\_history wykonywało się zbyt długo by dokonać analizy. Zastosowano więc ograniczenie do productid < 5.

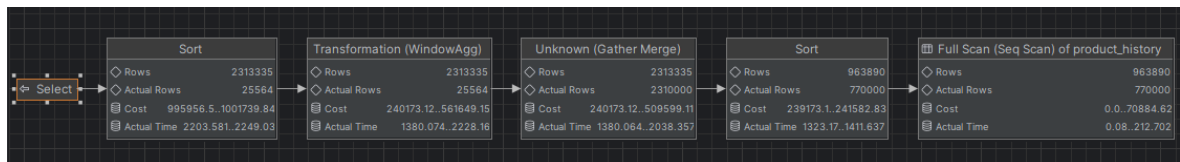


Porównanie

Na powyższych planach widać dużą różnicę w wykonaniu zadania, na korzyść funkcji okna. Największy wpływ na koszt zapytania ma część z join, która dodatkowo traci na wydajności przez brak odpowiednich indeksów, mamy jedynie clustered index. Czas wykonania wynosi 14sekund dla jedynie 4 id produktów. Gdzie dla funkcji okna mamy wykonanie ok 2-3 sekund.

b) PostgreSQL

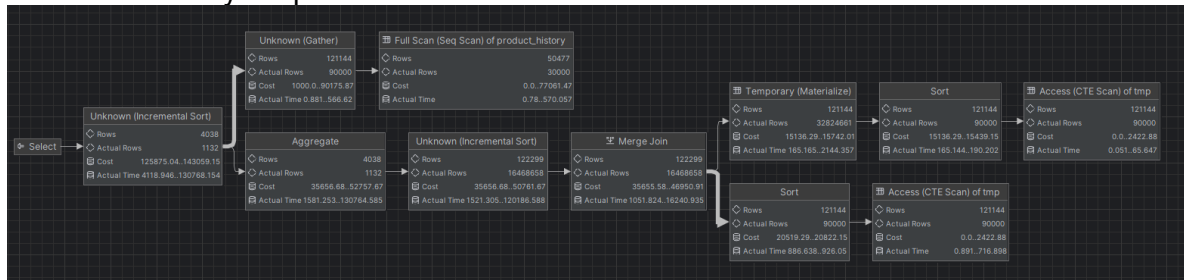
Polecenie 1



Cost=995956.5

## Polecenie 2

Cost=brak danych (Zbyt długo się liczyło), więc ograniczono tabelę początkową tak by zawierała rekordy dla productId<4



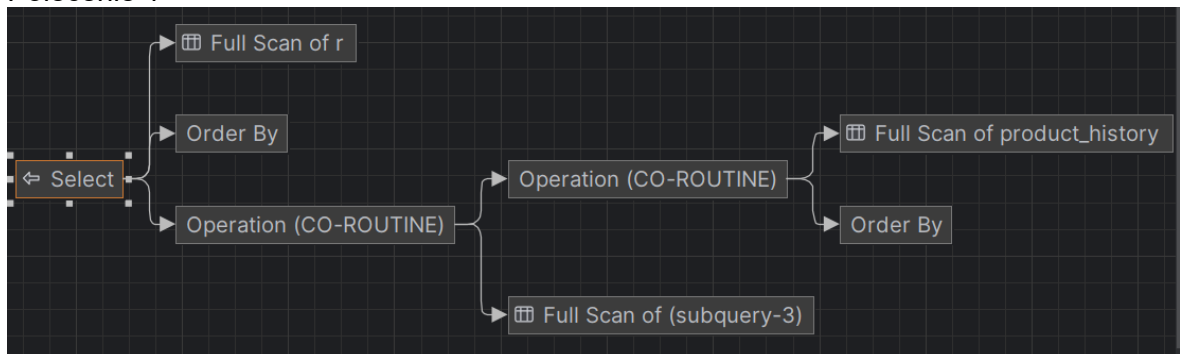
Cost = 143059

## Porównanie

Polecenie 1 z funkcją okna miało Cost=995956.5 i było wydajniejsze od polecenia 2 bez funkcji okna, gdzie Cost=brak danych (Zbyt długo się liczyło)

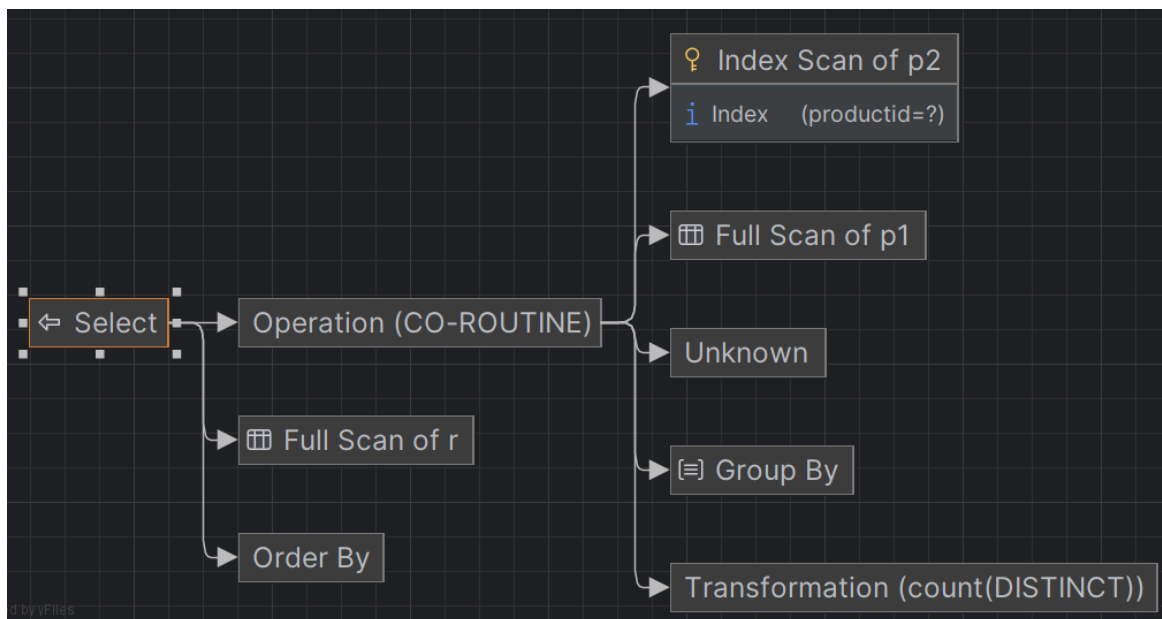
## c) SQLite

### Polecenie 1



### Polecenie 2





Na bazach Postgres i SQL Server konieczne było stworzenie tymczasowej tabeli zawierającej na starcie mniejszą ilość produktów by wersja zapytania bez funkcji okna się wykonała. A i tak czasy wykonania okazywały się gorsze. W przypadku SQLite nie zauważono aż takich rozbieżności – udało się wykonać 2 zapytanie bez ograniczania rozmiaru tabeli początkowej.

## Zadanie 10 - obserwacja

Funkcje lag(), lead()

Wykonaj polecenia, zaobserwuj wynik. Jak działają funkcje lag(), lead()

```

select productid, productname, categoryid, date, unitprice,
       lag(unitprice) over (partition by productid order by date)
       as previousprodprice,
       lead(unitprice) over (partition by productid order by date)
       as nextprodprice
from product_history
where productid = 1 and year(date) = 2022
order by date;

with t as (select productid, productname, categoryid, date, unitprice,
       lag(unitprice) over (partition by productid
       order by date) as previousprodprice,
       lead(unitprice) over (partition by productid
       order by date) as nextprodprice
       from product_history
       )
select * from t
where productid = 1 and year(date) = 2022
order by date;
```

Wynik polecenia 1 – wystąpiły dwie wartości NULL, jedna w pierwszym wierszu kolumny previousprodprice, a druga wartość w ostatnim wierszu kolumny nextprodprice. 2 wiersz w kolumnie previousprodprice będzie miał wartość 1 wiersza kolumny unitprice. 1 wiersz w kolumnie nextprodprice będzie miał wartość 2 wiersza kolumny unitprice.

|    | productid | productname | categoryid | date       | unitprice | previousprodprice | nextprodprice |
|----|-----------|-------------|------------|------------|-----------|-------------------|---------------|
| 1  | 1         | Chai        | 1          | 2022-01-01 | 27.69     | NULL              | 24.01         |
| 2  | 1         | Chai        | 1          | 2022-01-02 | 24.01     | 27.69             | 26.72         |
| 3  | 1         | Chai        | 1          | 2022-01-03 | 26.72     | 24.01             | 17.00         |
| 4  | 1         | Chai        | 1          | 2022-01-04 | 17.00     | 26.72             | 22.05         |
| 5  | 1         | Chai        | 1          | 2022-01-05 | 22.05     | 17.00             | 12.68         |
| 6  | 1         | Chai        | 1          | 2022-01-06 | 12.68     | 22.05             | 26.49         |
| 7  | 1         | Chai        | 1          | 2022-01-07 | 26.49     | 12.68             | 16.34         |
| 8  | 1         | Chai        | 1          | 2022-01-08 | 16.34     | 26.49             | 20.80         |
| 9  | 1         | Chai        | 1          | 2022-01-09 | 20.80     | 16.34             | 13.81         |
| 10 | 1         | Chai        | 1          | 2022-01-10 | 13.81     | 20.80             | 19.99         |
| 11 | 1         | Chai        | 1          | 2022-01-11 | 19.99     | 13.81             | 10.33         |
| 12 | 1         | Chai        | 1          | 2022-01-12 | 10.33     | 19.99             | 24.02         |
| 13 | 1         | Chai        | 1          | 2022-01-13 | 24.02     | 10.33             | 23.88         |
| 14 | 1         | Chai        | 1          | 2022-01-14 | 23.88     | 24.02             | 12.85         |
| 15 | 1         | Chai        | 1          | 2022-01-15 | 12.85     | 23.88             | 14.81         |
| 16 | 1         | Chai        | 1          | 2022-01-16 | 14.81     | 12.85             | 15.21         |
| 17 | 1         | Chai        | 1          | 2022-01-17 | 15.21     | 14.81             | 12.38         |
| 18 | 1         | Chai        | 1          | 2022-01-18 | 12.38     | 15.21             | 26.67         |
| 19 | 1         | Chai        | 1          | 2022-01-19 | 26.67     | 12.38             | 16.43         |

Wynik polecenia 2 – zaobserwowano tylko jedną wartość NULL w ostatnim wierszu w kolumnie nextprodprice. Wartość NULL nie została zaobserwowana w kolumnie previousprice. 2 wiersz w kolumnie previousprodprice będzie miał wartość 1 wiersza kolumny unitprice. 1 wiersz w kolumnie nextprodprice będzie miał wartość 2 wiersza kolumny unitprice.

|    | productid | productname | categoryid | date       | unitprice | previousprodprice | nextprodprice |
|----|-----------|-------------|------------|------------|-----------|-------------------|---------------|
| 32 | 1         | Chai        | 1          | 2022-02-01 | 12.69     | 10.26             | 26.61         |
| 33 | 1         | Chai        | 1          | 2022-02-02 | 26.61     | 12.69             | 16.69         |
| 34 | 1         | Chai        | 1          | 2022-02-03 | 16.69     | 26.61             | 23.13         |
| 35 | 1         | Chai        | 1          | 2022-02-04 | 23.13     | 16.69             | 24.22         |
| 36 | 1         | Chai        | 1          | 2022-02-05 | 24.22     | 23.13             | 23.19         |
| 37 | 1         | Chai        | 1          | 2022-02-06 | 23.19     | 24.22             | 19.14         |
| 38 | 1         | Chai        | 1          | 2022-02-07 | 19.14     | 23.19             | 10.12         |
| 39 | 1         | Chai        | 1          | 2022-02-08 | 10.12     | 19.14             | 16.35         |
| 40 | 1         | Chai        | 1          | 2022-02-09 | 16.35     | 10.12             | 11.21         |
| 41 | 1         | Chai        | 1          | 2022-02-10 | 11.21     | 16.35             | 16.81         |
| 42 | 1         | Chai        | 1          | 2022-02-11 | 16.81     | 11.21             | 22.11         |
| 43 | 1         | Chai        | 1          | 2022-02-12 | 22.11     | 16.81             | 18.79         |
| 44 | 1         | Chai        | 1          | 2022-02-13 | 18.79     | 22.11             | 12.81         |
| 45 | 1         | Chai        | 1          | 2022-02-14 | 12.81     | 18.79             | 21.99         |
| 46 | 1         | Chai        | 1          | 2022-02-15 | 21.99     | 12.81             | 19.77         |
| 47 | 1         | Chai        | 1          | 2022-02-16 | 19.77     | 21.99             | 27.56         |
| 48 | 1         | Chai        | 1          | 2022-02-17 | 27.56     | 19.77             | 19.32         |
| 49 | 1         | Chai        | 1          | 2022-02-18 | 19.32     | 27.56             | 19.62         |
| 50 | 1         | Chai        | 1          | 2022-02-19 | 19.62     | 19.32             | NULL          |

lag() - umożliwia dostęp do danych z poprzedniego wiersza w tym samym zestawie wyników bez użycia jakichkolwiek złączeń SQL.

lead() - pozwala na dostęp do danych z następnego wiersza w tym samym zestawie wyników bez użycia jakichkolwiek złączeń SQL.

#### Zadanie

Spróbuj uzyskać ten sam wynik bez użycia funkcji okna, porównaj wyniki, czasy i plany zapytań. Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSQL, SQLite)

Polecenie 1 bez funkcji okna

```
select
    p1.productid,
    p1.productname,
    p1.categoryid,
    p1.date,
    p1.unitprice,
    p2.unitprice as previousprodprice,
    p3.unitprice as extprodprice
from product_history as p1
inner join product_history as p2
on p1.productid = p2.productid and p2.date = (
    select max(date)
    from product_history
    where productid = p1.productid and date < p1.date)
inner join product_history as p3
on p2.productid = p3.productid and p3.date = (
    select min(date)
    from product_history
    where productid = p1.productid and date > p1.date)
where p1.productid = 1 and year(p1.date) = 2022
order by p1.productid, p1.date;
```

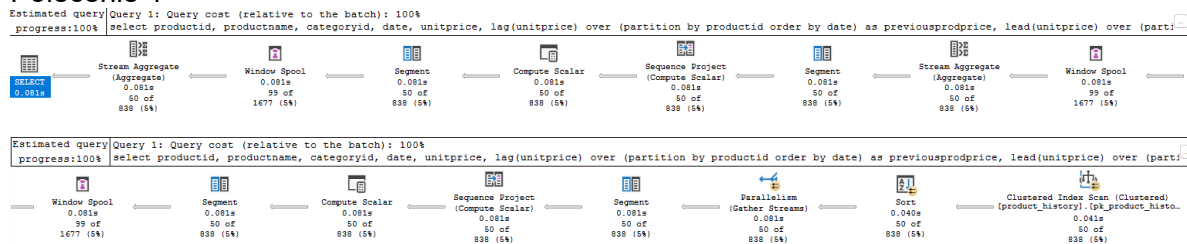
Polecenie 2 bez funkcji okna

```
with t as (
    select
        p1.productid,
        p1.productname,
        p1.categoryid,
        p1.date,
        p1.unitprice,
        p2.unitprice as previousprodprice,
        p3.unitprice as extprodprice
    from product_history as p1
    inner join product_history as p2
    on p1.productid = p2.productid and p2.date = (
        select max(date)
        from product_history
        where productid = p1.productid and date < p1.date)
    inner join product_history as p3
    on p2.productid = p3.productid and p3.date = (
        select min(date)
        from product_history
        where productid = p1.productid and date > p1.date)
)
select * from t
where productid = 1 and year(date) = 2022
order by date
```

Porównanie czasów oraz planów wykonania zapytań

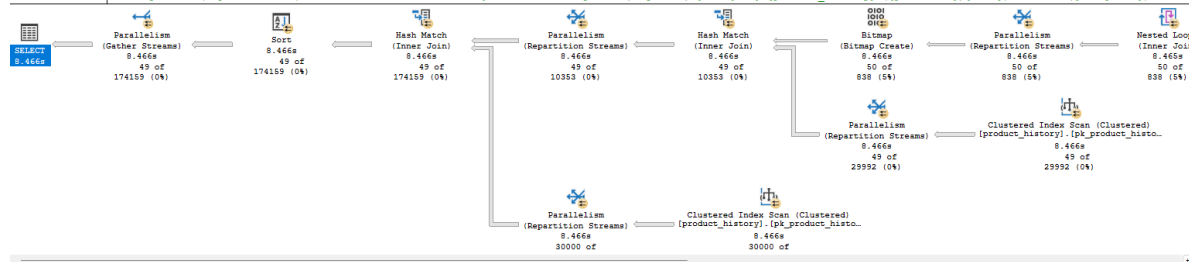
a) MS SQL Server

Polecenie 1



## Polecenie 1 – bez funkcji okna

Estimated query cost (relative to the batch): 100%  
 select p1.productid, p1.productname, p1.categoryid, p1.date, p1.unitprice, p2.unitprice as previousprodprice, p3.unitprice as extprodprice from product\_history as p1 inner join product\_history as p2 on (p1.productid = p2.productid and p1.date < p2.date) inner join product\_history as p3 on (p1.productid = p3.productid and p1.date = p3.date) include ((unitprice))



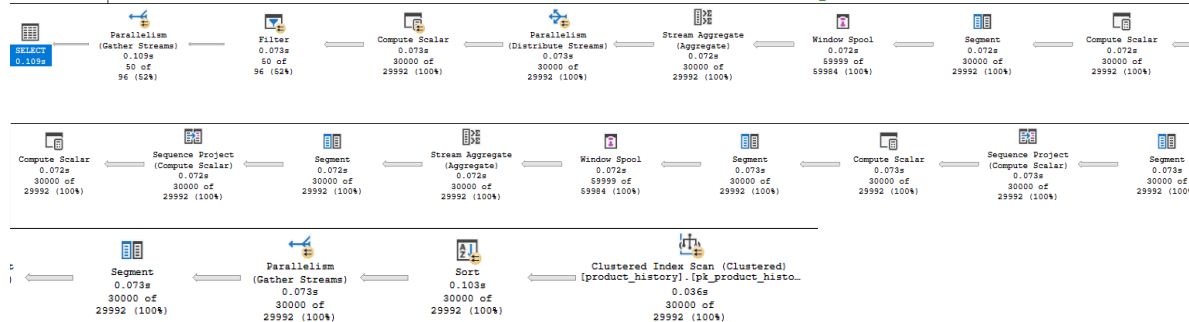
Estimated Subtree Cost: 187,7726

## Porównanie

Polecenie 1 z funkcją okna ma Estimated Subtree Cost=19,358 i jest wydajniejsze od swojej wersji bez funkcji okna, która ma Estimated Subtree Cost:=187,7726

## Polecenie 2

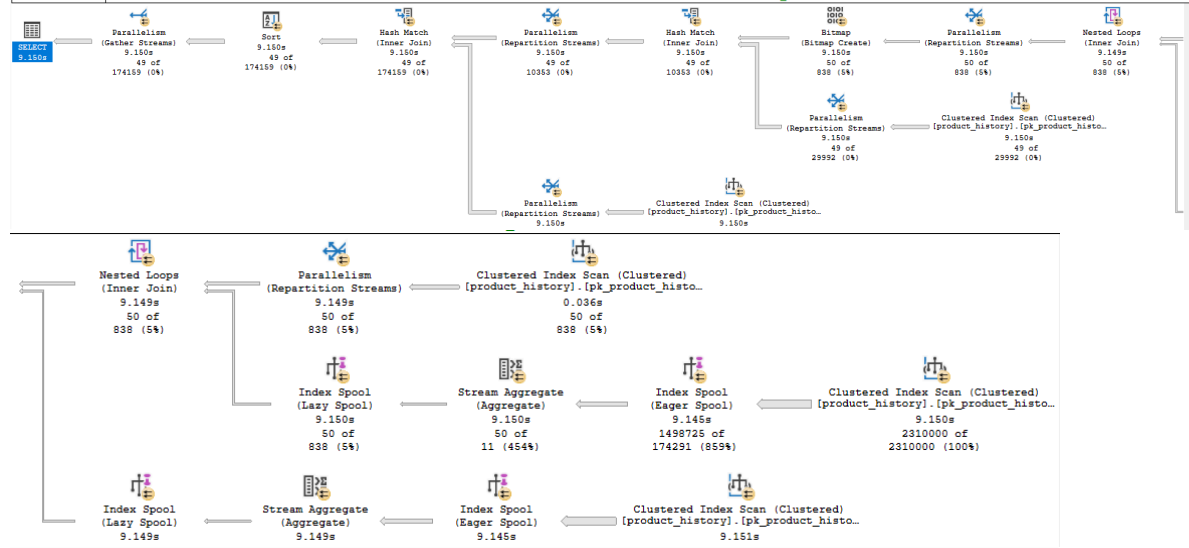
Estimated query cost (relative to the batch): 100%  
 with t as (select productid, productname, categoryid, date, unitprice, lag(unitprice) over (partition by productid order by date) as previousprodprice, lead(unitprice) over (partition by productid order by date) as extprodprice from product\_history) select t.productid, t.productname, t.categoryid, t.date, t.unitprice, t.previousprodprice, t.extprodprice from t



Estimated Subtree Cost: 20,1187

## Polecenie 2 – bez funkcji okna

Estimated query cost (relative to the batch): 100%  
 select p1.productid, p1.productname, p1.categoryid, p1.date, p1.unitprice, p2.unitprice as previousprodprice, p3.unitprice as extprodprice from product\_history as p1 inner join product\_history as p2 on (p1.productid = p2.productid and p1.date < p2.date) inner join product\_history as p3 on (p1.productid = p3.productid and p1.date = p3.date) include ((unitprice))



Estimated Subtree Cost: 194,668

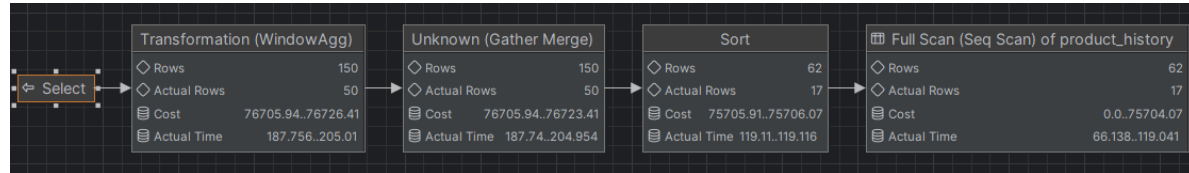
## Porównanie

Polecenie 2 z funkcją okna ma Estimated Subtree Cost=20,1187 i jest wydajniejsze od swojej wersji bez funkcji okna, która ma Estimated Subtree Cost:=194,668

swojej wersji bez funkcji okna, która ma Estimated Subtree Cost:= 194,668

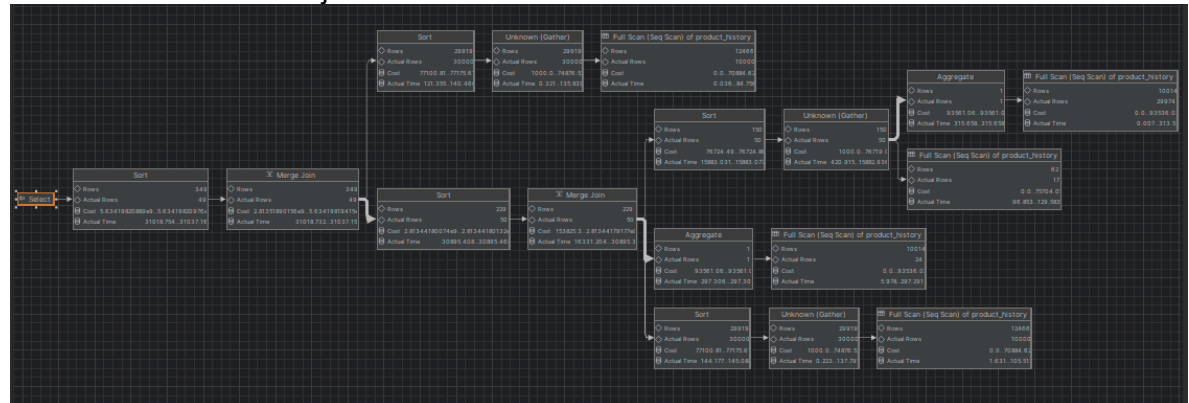
b) PostgreSQL

Polecenie 1



Cost:78705.94

Polecenie 1 – bez funkcji okna

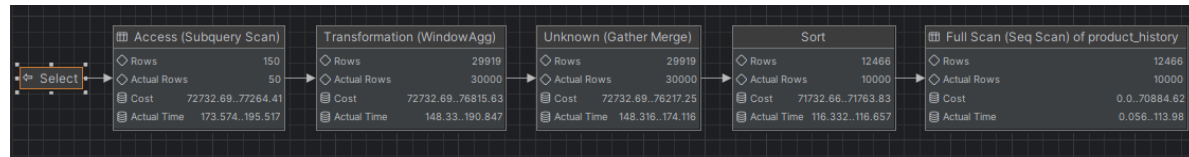


Cost:5634198...5.63

Porównanie

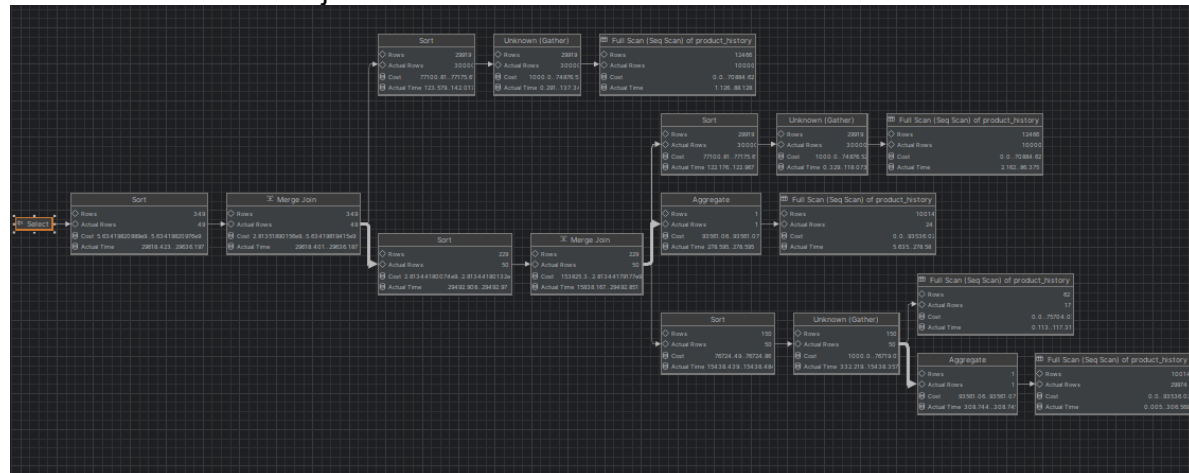
Polecenie 1 z funkcją okna ma Cost=78705.94 i jest wydajniejsze od swojej wersji bez funkcji okna, która ma Cost=5634198...5.63

Polecenie 2



Cost:72732.69

Polecenie 2 – bez funkcji okna



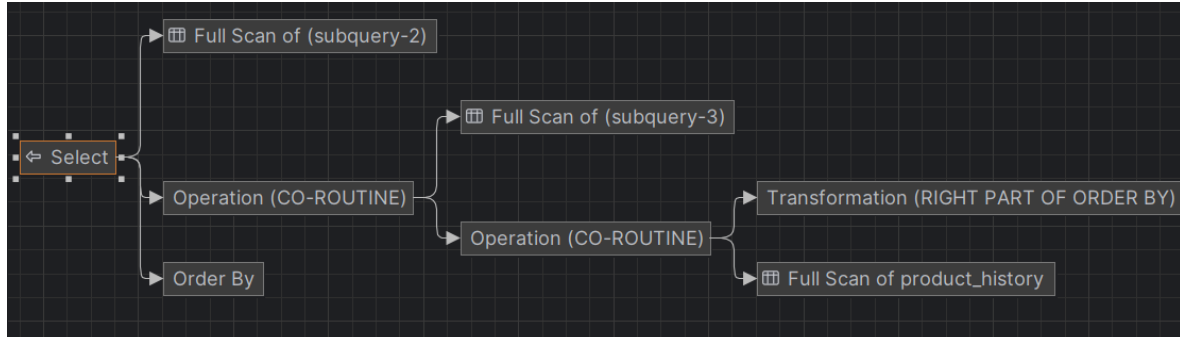
Cost:563419820..5.63

## Porównanie

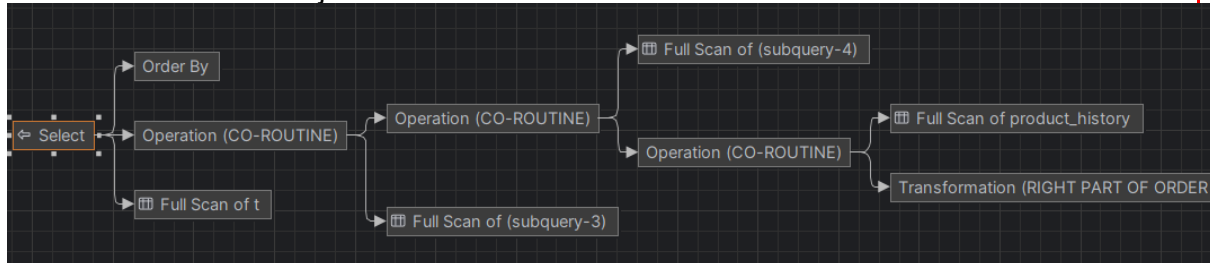
Polecenie 2 z funkcją okna ma Cost=72732.69 i jest wydajniejsze od swojej wersji bez funkcji okna, która ma Cost=5634198...5.63

### c) SQLite

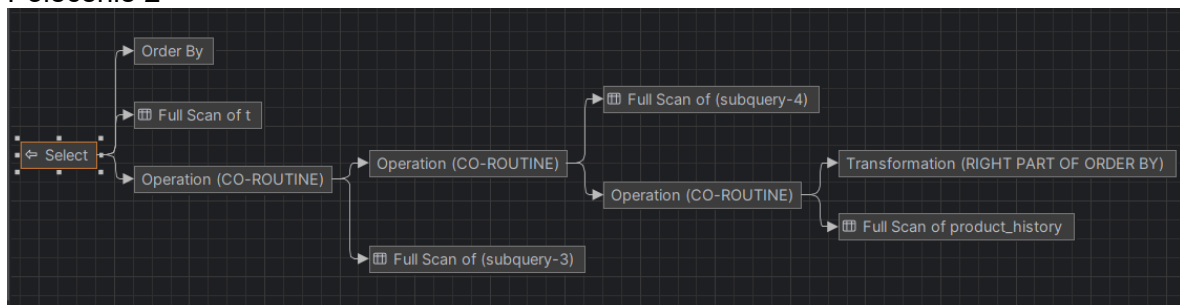
#### Polecenie 1



#### Polecenie 1 – bez funkcji okna



#### Polecenie 2



#### Polecenie 2 – bez funkcji okna

[2024-03-01 19:57:15] Database returned plan in unsupported format: unexpected PARTIAL != INDEX

Komunikat "Database returned plan in unsupported format: unexpected PARTIAL != INDEX" wskazuje na to, że baza danych zwróciła plan wykonania w nieobsługiwanym formacie. To może wynikać z różnic w implementacji bazy danych lub błędzie w działaniu samej bazy danych.

W diagramach powyżej widać dużą różnicę w realizacji/prezentacji realizacji zapytań za pomocą funkcji okna dla sqlite w stosunku do postgres i SQL server. Gdzie kolejne operacje wykonują się po sobie w odpowiedniej kolejności, natomiast w sqlite mamy rozgałęzienia bardzo podobne do tych, które pojawiają się dla diagramów powstałych z wersji rozwiązania z join lub podzapytaniem.

## Zadanie 11

Baza: Northwind, tabele customers, orders, order details

Napisz polecenie które wyświetla inf. o zamówieniach

Zbiór wyników powinien zawierać:

nazwę klienta, nr zamówienia,

datę zamówienia,

wartość zamówienia (wraz z opłatą za przesyłkę),

nr poprzedniego zamówienia danego klienta,

datę poprzedniego zamówienia danego klienta,

wartość poprzedniego zamówienia danego klienta.

```
select  c.CompanyName,
        o.OrderID,
        o.OrderDate,
        sum(od.Quantity * od.UnitPrice) + o.Freight as OrderValue,
        lag(o.OrderID) over (partition by o.CustomerId order by o.OrderId ) as
PrevOrderID,
        lag(o.OrderDate) over (partition by o.CustomerId order by
o.OrderID ) as PrevOrderDate,
        lag(sum(od.Quantity * od.UnitPrice) + o.Freight) over (partition by
o.CustomerId order by o.OrderId ) as PrevOrderValue
from Customers c
join Orders o on o.CustomerID = c.CustomerID
join [Order Details] od on od.OrderID = o.OrderID
group by c.CompanyName,
        o.OrderID,
        o.OrderDate,
        o.Freight,
        o.CustomerID
```

Wyniki:

|    | CompanyName                        | OrderID | OrderDate               | OrderValue | PrevOrderID | PrevOrderDate           | PrevOrderValue |
|----|------------------------------------|---------|-------------------------|------------|-------------|-------------------------|----------------|
| 1  | Alfreds Futterkiste                | 10643   | 1997-08-25 00:00:00.000 | 1115.46    | NULL        | NULL                    | NULL           |
| 2  | Alfreds Futterkiste                | 10692   | 1997-10-03 00:00:00.000 | 939.02     | 10643       | 1997-08-25 00:00:00.000 | 1115.46        |
| 3  | Alfreds Futterkiste                | 10702   | 1997-10-13 00:00:00.000 | 353.94     | 10692       | 1997-10-03 00:00:00.000 | 939.02         |
| 4  | Alfreds Futterkiste                | 10835   | 1998-01-15 00:00:00.000 | 920.53     | 10702       | 1997-10-13 00:00:00.000 | 353.94         |
| 5  | Alfreds Futterkiste                | 10952   | 1998-03-16 00:00:00.000 | 531.62     | 10835       | 1998-01-15 00:00:00.000 | 920.53         |
| 6  | Alfreds Futterkiste                | 11011   | 1998-04-09 00:00:00.000 | 961.21     | 10952       | 1998-03-16 00:00:00.000 | 531.62         |
| 7  | Ana Trujillo Emparedados y helados | 10308   | 1996-09-18 00:00:00.000 | 90.41      | NULL        | NULL                    | NULL           |
| 8  | Ana Trujillo Emparedados y helados | 10625   | 1997-08-08 00:00:00.000 | 523.65     | 10308       | 1996-09-18 00:00:00.000 | 90.41          |
| 9  | Ana Trujillo Emparedados y helados | 10759   | 1997-11-28 00:00:00.000 | 331.99     | 10625       | 1997-08-08 00:00:00.000 | 523.65         |
| 10 | Ana Trujillo Emparedados y helados | 10926   | 1998-03-04 00:00:00.000 | 554.32     | 10759       | 1997-11-28 00:00:00.000 | 331.99         |
| 11 | Antonio Moreno Taquería            | 10365   | 1996-11-27 00:00:00.000 | 425.20     | NULL        | NULL                    | NULL           |
| 12 | Antonio Moreno Taquería            | 10507   | 1997-04-15 00:00:00.000 | 928.70     | 10365       | 1996-11-27 00:00:00.000 | 425.20         |
| 13 | Antonio Moreno Taquería            | 10535   | 1997-05-13 00:00:00.000 | 2172.14    | 10507       | 1997-04-15 00:00:00.000 | 928.70         |
| 14 | Antonio Moreno Taquería            | 10573   | 1997-06-19 00:00:00.000 | 2166.84    | 10535       | 1997-05-13 00:00:00.000 | 2172.14        |
| 15 | Antonio Moreno Taquería            | 10677   | 1997-09-22 00:00:00.000 | 960.93     | 10573       | 1997-06-19 00:00:00.000 | 2166.84        |
| 16 | Antonio Moreno Taquería            | 10682   | 1997-09-25 00:00:00.000 | 411.63     | 10677       | 1997-09-22 00:00:00.000 | 960.93         |
| 17 | Antonio Moreno Taquería            | 10856   | 1998-01-28 00:00:00.000 | 718.43     | 10682       | 1997-09-25 00:00:00.000 | 411.63         |

## Zadanie 12 - obserwacja

Funkcje first\_value(), last\_value()

Wykonaj polecenia, zaobserwuj wynik. Jak działają funkcje `first_value()`, `last_value()`. Skomentuj uzyskane wyniki. Czy funkcja `first_value` pokazuje w tym przypadku najdroższy produkt w danej kategorii, czy funkcja `last_value()` pokazuje najtańszy produkt? Co jest przyczyną takiego działania funkcji `last_value`. Co trzeba zmienić żeby funkcja `last_value` pokazywała najtańszy produkt w danej kategorii

```
select productid, productname, unitprice, categoryid,  
       first_value(productname) over (partition by categoryid  
                                     order by unitprice desc) first,  
       last_value(productname) over (partition by categoryid  
                                    order by unitprice desc) last  
from products  
order by categoryid, unitprice desc;
```

First value pokazuje najdroższy produkt, funkcja `last_value` pokazuje najtańszy produkt ale z zakresu produktów nie tańszych niż produkt w danym rekordzie, stąd z kolejnymi rekordami wartości `last_value` ulegają zmianie.

Wersja, w której `last_value` pokazuje najtańszy produkt w danej kategorii:

```
select productid,  
       productname,  
       unitprice,  
       categoryid,  
       first_value(productname) over (partition by categoryid  
                                     order by unitprice desc) first,  
       last_value(productname) over (partition by categoryid  
                                    order by unitprice desc  
                                    RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) last  
from products  
order by categoryid, unitprice desc ;
```

Wyniki:



|    | productid | productname       | unitprice | categoryid | first         | last               |
|----|-----------|-------------------|-----------|------------|---------------|--------------------|
| 1  | 38        | Côte de Blaye     | 263,50    | 1          | Côte de Blaye | Guaraná Fantástica |
| 2  | 43        | Ipoh Coffee       | 46,00     | 1          | Côte de Blaye | Guaraná Fantástica |
| 3  | 2         | Chang             | 19,00     | 1          | Côte de Blaye | Guaraná Fantástica |
| 4  | 1         | Chai              | 18,00     | 1          | Côte de Blaye | Guaraná Fantástica |
| 5  | 39        | Chartreuse verte  | 18,00     | 1          | Côte de Blaye | Guaraná Fantástica |
| 6  | 35        | Steeleye Stout    | 18,00     | 1          | Côte de Blaye | Guaraná Fantástica |
| 7  | 76        | Lakkalikööri      | 18,00     | 1          | Côte de Blaye | Guaraná Fantástica |
| 8  | 70        | Outback Lager     | 15,00     | 1          | Côte de Blaye | Guaraná Fantástica |
| 9  | 67        | Laughing Lum...   | 14,00     | 1          | Côte de Blaye | Guaraná Fantástica |
| 10 | 34        | Sasquatch Ale     | 14,00     | 1          | Côte de Blaye | Guaraná Fantástica |
| 11 | 75        | Rhönbräu Klo...   | 7,75      | 1          | Côte de Blaye | Guaraná Fantástica |
| 12 | 24        | Guaraná Fant...   | 4,50      | 1          | Côte de Blaye | Guaraná Fantástica |
| 13 | 63        | Vegie-spread      | 43,90     | 2          | Vegie-spread  | Aniseed Syrup      |
| 14 | 8         | Northwoods Cr...  | 40,00     | 2          | Vegie-spread  | Aniseed Syrup      |
| 15 | 61        | Sirop d'érable    | 28,50     | 2          | Vegie-spread  | Aniseed Syrup      |
| 16 | 6         | Grandma's Bo...   | 25,00     | 2          | Vegie-spread  | Aniseed Syrup      |
| 17 | 4         | Chef Anton's C... | 22,00     | 2          | Vegie-spread  | Aniseed Syrup      |

#### Zadanie

Spróbuj uzyskać ten sam wynik bez użycia funkcji okna, porównaj wyniki, czasy i plany zapytań. Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSQL, SQLite)

```

select p.productid,
       p.productname,
       p.unitprice,
       p.categoryid,
       p_min.ProductName as MinProductName ,
       p_max.ProductName as MaxProductName
from products p
join (
    select CategoryId, min(UnitPrice) as minPrice, max(UnitPrice) as maxPrice
    from products
    group by CategoryID
) as x on 1 = 1
join products p_min on p_min.UnitPrice = x.minPrice
                    and p_min.CategoryID = x.CategoryID
                    and p.CategoryID = p_min.CategoryID
join products p_max on p_max.UnitPrice = x.maxPrice
                    and p_max.CategoryID = x.CategoryID
                    and p.CategoryID = p_max.CategoryID
order by p.categoryid , p.unitprice desc;

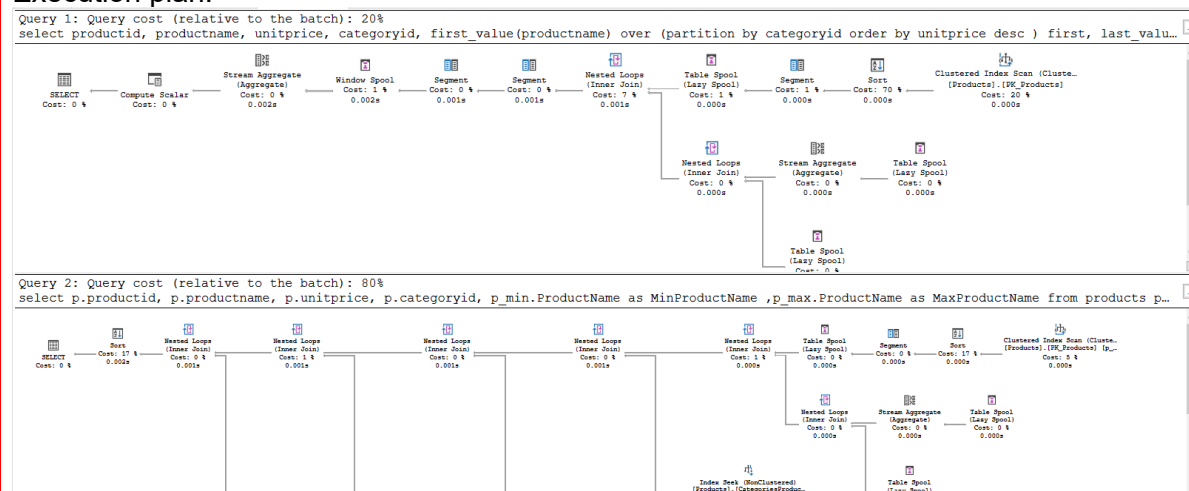
```

Wynik:

|    | productid | productname                  | unitprice | categoryid | MinProductName     | MaxProductName |
|----|-----------|------------------------------|-----------|------------|--------------------|----------------|
| 1  | 38        | Côte de Blaye                | 263,50    | 1          | Guaraná Fantástica | Côte de Blaye  |
| 2  | 43        | Ipoh Coffee                  | 46,00     | 1          | Guaraná Fantástica | Côte de Blaye  |
| 3  | 2         | Chang                        | 19,00     | 1          | Guaraná Fantástica | Côte de Blaye  |
| 4  | 1         | Chai                         | 18,00     | 1          | Guaraná Fantástica | Côte de Blaye  |
| 5  | 39        | Chartreuse verte             | 18,00     | 1          | Guaraná Fantástica | Côte de Blaye  |
| 6  | 76        | Lakkalikööri                 | 18,00     | 1          | Guaraná Fantástica | Côte de Blaye  |
| 7  | 35        | Steeleye Stout               | 18,00     | 1          | Guaraná Fantástica | Côte de Blaye  |
| 8  | 70        | Outback Lager                | 15,00     | 1          | Guaraná Fantástica | Côte de Blaye  |
| 9  | 67        | Laughing Lumberjack Lager    | 14,00     | 1          | Guaraná Fantástica | Côte de Blaye  |
| 10 | 34        | Sasquatch Ale                | 14,00     | 1          | Guaraná Fantástica | Côte de Blaye  |
| 11 | 75        | Rhönbräu Klosterbier         | 7,75      | 1          | Guaraná Fantástica | Côte de Blaye  |
| 12 | 24        | Guaraná Fantástica           | 4,50      | 1          | Guaraná Fantástica | Côte de Blaye  |
| 13 | 63        | Vegie-spread                 | 43,90     | 2          | Aniseed Syrup      | Vegie-spread   |
| 14 | 8         | Northwoods Cranberry Sauce   | 40,00     | 2          | Aniseed Syrup      | Vegie-spread   |
| 15 | 61        | Sirop d'érable               | 28,50     | 2          | Aniseed Syrup      | Vegie-spread   |
| 16 | 6         | Grandma's Boysenberry Spread | 25,00     | 2          | Aniseed Syrup      | Vegie-spread   |
| 17 | 4         | Chef Anton's Cajun Seasoning | 22,00     | 2          | Aniseed Syrup      | Vegie-spread   |

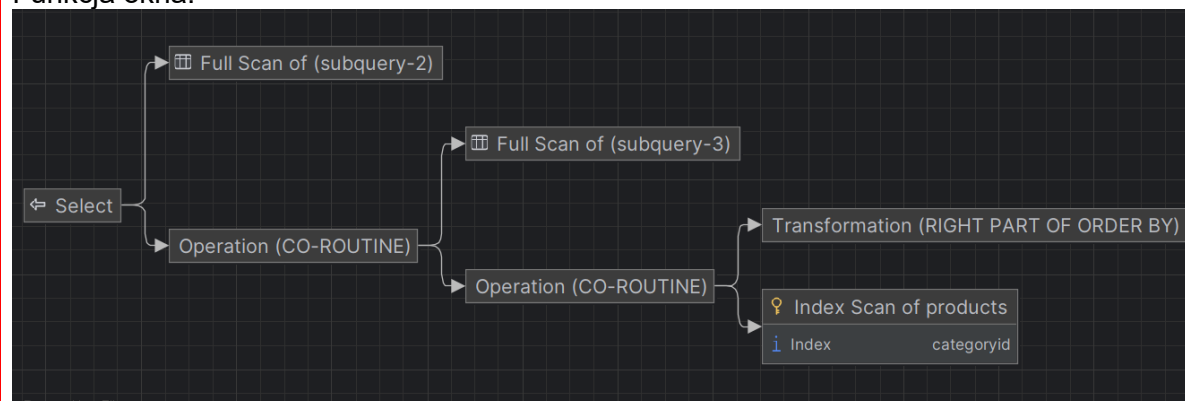
## Analiza T-SQL

### Execution plan:



## SQLite

### Funkcja okna:

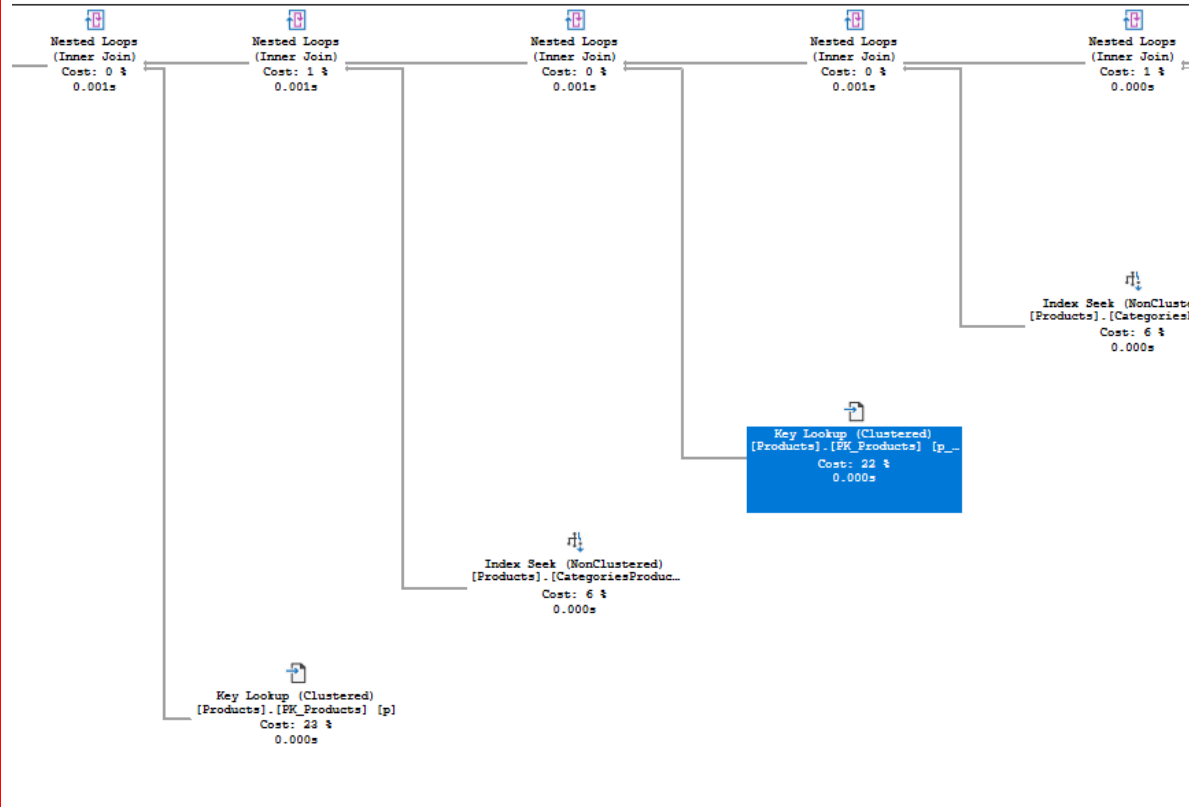


### Bez funkcji okna:

### Czas:



ilości zasobów, procentowo 20 do 80. Największy koszt w przypadku tego zapytania powodują wykonania join i sprawdzanie indeksów – co jest spodziewane:



## Zadanie 13

Baza: Northwind, tabele orders, order details

Napisz polecenie które wyświetla inf. o zamówieniach

Zbiór wyników powinien zawierać:

Id klienta,

nr zamówienia,

datę zamówienia,

wartość zamówienia (wraz z opłatą za przesyłkę),

dane zamówienia klienta o najniższej wartości w danym miesiącu

nr zamówienia o najniższej wartości w danym miesiącu

datę tego zamówienia

wartość tego zamówienia

dane zamówienia klienta o najwyższej wartości w danym miesiącu

nr zamówienia o najniższej wartości w danym miesiącu

datę tego zamówienia

wartość tego zamówienia

```

;with tmp (CustomerID ,CompanyName,OrderID, OrderDate ,MinOrderId,MaxOrderId)
as(
    select c.CustomerID ,
           c.CompanyName,
           o.OrderID,
           o.OrderDate ,
           first_value(o.OrderID)over (PARTITION BY c.CustomerID, YEAR(OrderDate),
                                       MONTH(OrderDate) order by sum(od.Quantity * od.UnitPrice) + o.Freight)as MinOrderId,
           first_value(o.OrderID) over (PARTITION BY c.CustomerID , YEAR(OrderDate), MONTH(OrderDate)
                                       order by sum(od.Quantity * od.UnitPrice) + o.Freight desc) as MaxOrderId
    from Customers c
    join Orders o on o.CustomerID = c.CustomerID
    join [Order Details] od on od.OrderID = o.OrderID
    group by c.CustomerId , c.CompanyName , o.OrderID, o.OrderDate, o.Freight
),
order_values(OrderId ,OrderDate, OrderValue)
as(
    select o.OrderID,o.OrderDate, sum(od.Quantity * od.UnitPrice) + o.Freight
    from Orders o
    join [Order Details] od on od.OrderID = o.OrderID
    group by o.OrderID, o.OrderDate ,o.Freight
)
select x.CompanyName,
       o.OrderID,
       o.OrderDate,
       x.MinOrderId,
       o_min.OrderDate as MinOrderDate,
       o_min.OrderValue as MinOrderValue,
       x.MaxOrderId,
       o_max.OrderDate as MaxOrderDate,
       o_max.OrderValue as MinOrderValue
from tmp x
join order_values o on o.OrderID = x.OrderID
join order_values o_min on o_min.OrderID = x.MinOrderId
join order_values o_max on o_max.OrderID = x.MaxOrderId
order by CompanyName ,orderDate

```

Wynik:

|    | CompanyName                        | OrderID | OrderDate               | MinOrderId | MinOrderDate            | MinOrderValue | MaxOrderId | MaxOrderDate            | MinOrderValue |
|----|------------------------------------|---------|-------------------------|------------|-------------------------|---------------|------------|-------------------------|---------------|
| 1  | Alfreds Futterkiste                | 10643   | 1997-08-25 00:00:00.000 | 10643      | 1997-08-25 00:00:00.000 | 1115.46       | 10643      | 1997-08-25 00:00:00.000 | 1115.46       |
| 2  | Alfreds Futterkiste                | 10692   | 1997-10-03 00:00:00.000 | 10702      | 1997-10-13 00:00:00.000 | 353.94        | 10692      | 1997-10-03 00:00:00.000 | 939.02        |
| 3  | Alfreds Futterkiste                | 10702   | 1997-10-13 00:00:00.000 | 10702      | 1997-10-13 00:00:00.000 | 353.94        | 10692      | 1997-10-03 00:00:00.000 | 939.02        |
| 4  | Alfreds Futterkiste                | 10835   | 1998-01-15 00:00:00.000 | 10835      | 1998-01-15 00:00:00.000 | 920.53        | 10835      | 1998-01-15 00:00:00.000 | 920.53        |
| 5  | Alfreds Futterkiste                | 10952   | 1998-03-16 00:00:00.000 | 10952      | 1998-03-16 00:00:00.000 | 531.62        | 10952      | 1998-03-16 00:00:00.000 | 531.62        |
| 6  | Alfreds Futterkiste                | 11011   | 1998-04-09 00:00:00.000 | 11011      | 1998-04-09 00:00:00.000 | 961.21        | 11011      | 1998-04-09 00:00:00.000 | 961.21        |
| 7  | Ana Trujillo Emparedados y helados | 10308   | 1996-09-18 00:00:00.000 | 10308      | 1996-09-18 00:00:00.000 | 90.41         | 10308      | 1996-09-18 00:00:00.000 | 90.41         |
| 8  | Ana Trujillo Emparedados y helados | 10625   | 1997-08-08 00:00:00.000 | 10625      | 1997-08-08 00:00:00.000 | 523.65        | 10625      | 1997-08-08 00:00:00.000 | 523.65        |
| 9  | Ana Trujillo Emparedados y helados | 10759   | 1997-11-28 00:00:00.000 | 10759      | 1997-11-28 00:00:00.000 | 331.99        | 10759      | 1997-11-28 00:00:00.000 | 331.99        |
| 10 | Ana Trujillo Emparedados y helados | 10926   | 1998-03-04 00:00:00.000 | 10926      | 1998-03-04 00:00:00.000 | 554.32        | 10926      | 1998-03-04 00:00:00.000 | 554.32        |
| 11 | Antonio Moreno Taquería            | 10365   | 1996-11-27 00:00:00.000 | 10365      | 1996-11-27 00:00:00.000 | 425.20        | 10365      | 1996-11-27 00:00:00.000 | 425.20        |
| 12 | Antonio Moreno Taquería            | 10507   | 1997-04-15 00:00:00.000 | 10507      | 1997-04-15 00:00:00.000 | 928.70        | 10507      | 1997-04-15 00:00:00.000 | 928.70        |
| 13 | Antonio Moreno Taquería            | 10535   | 1997-05-13 00:00:00.000 | 10535      | 1997-05-13 00:00:00.000 | 2172.14       | 10535      | 1997-05-13 00:00:00.000 | 2172.14       |
| 14 | Antonio Moreno Taquería            | 10573   | 1997-06-19 00:00:00.000 | 10573      | 1997-06-19 00:00:00.000 | 2166.84       | 10573      | 1997-06-19 00:00:00.000 | 2166.84       |
| 15 | Antonio Moreno Taquería            | 10677   | 1997-09-22 00:00:00.000 | 10682      | 1997-09-25 00:00:00.000 | 411.63        | 10677      | 1997-09-22 00:00:00.000 | 960.93        |
| 16 | Antonio Moreno Taquería            | 10682   | 1997-09-25 00:00:00.000 | 10682      | 1997-09-25 00:00:00.000 | 411.63        | 10677      | 1997-09-22 00:00:00.000 | 960.93        |
| 17 | Antonio Moreno Taquería            | 10956   | 1998-01-28 00:00:00.000 | 10956      | 1998-01-28 00:00:00.000 | 719.42        | 10956      | 1998-01-28 00:00:00.000 | 719.42        |

## Zadanie 14

Baza: Northwind, tabela product\_history

Napisz polecenie które pokaże wartość sprzedaży każdego produktu narastająco od początku każdego miesiąca. Użyj funkcji okna

Zbiór wynikowy powinien zawierać:

id pozycji  
id produktu  
datę

wartość sprzedaży produktu w danym dniu

wartość sprzedaży produktu narastające od początku miesiąca

```
;with tmp(id , ProductID, Date, Value)
as
(
    select ph.id, ph.ProductID, ph.Date, sum(Value) OVER (PARTITION BY ph.ProductID, ph.date )
    from product_history ph
    group by ph.ProductID, ph.date , ph.value, ph.id
)
select t.id,
       t.ProductID,
       t.Date,
       t.Value as Value,
       sum(t.Value) OVER(PARTITION BY ProductId, MONTH(t.Date) ORDER BY ProductId ,Date  ROWS
                          UNBOUNDED PRECEDING) AS CumulativeValueMonth
from tmp t
group by t.ProductID, t.Date, t.Value, Id
order by id , ProductId ,Date
wynik:
```

|    | id | ProductID | Date       | Value  | CumulativeValueMonth |
|----|----|-----------|------------|--------|----------------------|
| 1  | 1  | 1         | 1940-01-02 | 158.50 | 158.50               |
| 2  | 2  | 2         | 1940-01-02 | 161.80 | 161.80               |
| 3  | 3  | 3         | 1940-01-02 | 132.50 | 132.50               |
| 4  | 4  | 4         | 1940-01-02 | 188.65 | 188.65               |
| 5  | 5  | 5         | 1940-01-02 | 186.34 | 186.34               |
| 6  | 6  | 6         | 1940-01-02 | 199.43 | 199.43               |
| 7  | 7  | 7         | 1940-01-02 | 237.00 | 237.00               |
| 8  | 8  | 8         | 1940-01-02 | 276.12 | 276.12               |
| 9  | 9  | 9         | 1940-01-02 | 498.48 | 498.48               |
| 10 | 10 | 10        | 1940-01-02 | 261.04 | 261.04               |
| 11 | 11 | 11        | 1940-01-02 | 218.79 | 218.79               |
| 12 | 12 | 12        | 1940-01-02 | 290.68 | 290.68               |
| 13 | 13 | 13        | 1940-01-02 | 167.30 | 167.30               |
| 14 | 14 | 14        | 1940-01-02 | 245.84 | 245.84               |
| 15 | 15 | 15        | 1940-01-02 | 210.56 | 210.56               |
| 16 | 16 | 16        | 1940-01-02 | 235.05 | 235.05               |
| 17 | 17 | 17        | 1940-01-02 | 340.20 | 340.20               |
| 18 | 18 | 18        | 1940-01-02 | 454.80 | 454.80               |
| 19 | 19 | 19        | 1940-01-02 | 207.84 | 207.84               |
| 20 | 20 | 20        | 1940-01-02 | 581.44 | 581.44               |
| 21 | 21 | 21        | 1940-01-02 | 212.00 | 212.00               |
| 22 | 22 | 22        | 1940-01-02 | 286.11 | 286.11               |

Czas wykonania w mssql: 33s

Spróbuj wykonać zadanie bez użycia funkcji okna. Spróbuj uzyskać ten sam wynik bez użycia funkcji okna, porównaj wyniki, czasy i plany zapytań. Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSQL, SQLite)

```

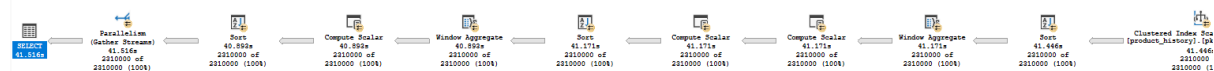
;with tmp(id , ProductID, Date, Value)
as
(
    select ph.id, ph.ProductID,ph.Date, sum(Value) OVER (PARTITION BY ph.ProductID, ph.date )
    from product_history ph
    group by ph.ProductID, ph.date , ph.value,ph.id
)
select t1.id,
       t1.ProductId,
       t1.Date,
       t1.Value,
       sum(isnull(t2.Value, t1.Value)) as CumulativeValueMonth
from tmp t1
left join tmp t2 on t1.ProductId = t2.ProductId
                  and Month(t2.Date) = Month(t1.Date)
                  and Year(t2.Date) = Year(t1.Date)
                  and t2.Date<=t1.Date
Group by t1.Date , t1.Value, t1.ProductId, t1.id
order by t1.id , t1.ProductId , t1.Date

```

|    | id | ProductId | Date       | Value  | CumulativeValueMonth |
|----|----|-----------|------------|--------|----------------------|
| 1  | 1  | 1         | 1940-01-02 | 233.70 | 233.70               |
| 2  | 2  | 2         | 1940-01-02 | 241.10 | 241.10               |
| 3  | 3  | 3         | 1940-01-02 | 174.30 | 174.30               |
| 4  | 4  | 4         | 1940-01-02 | 289.74 | 289.74               |
| 5  | 5  | 5         | 1940-01-02 | 284.35 | 284.35               |
| 6  | 6  | 6         | 1940-01-02 | 314.16 | 314.16               |
| 7  | 7  | 7         | 1940-01-02 | 387.36 | 387.36               |
| 8  | 8  | 8         | 1940-01-02 | 476.40 | 476.40               |
| 9  | 9  | 9         | 1940-01-02 | 984.36 | 984.36               |
| 10 | 10 | 10        | 1940-01-02 | 429.26 | 429.26               |
| 11 | 11 | 11        | 1940-01-02 | 332.67 | 332.67               |
| 12 | 12 | 12        | 1940-01-02 | 496.86 | 496.86               |
| 13 | 13 | 13        | 1940-01-02 | 202.44 | 202.44               |
| 14 | 14 | 14        | 1940-01-02 | 381.64 | 381.64               |
| 15 | 15 | 15        | 1940-01-02 | 301.14 | 301.14               |
| 16 | 16 | 16        | 1940-01-02 | 344.40 | 344.40               |
| 17 | 17 | 17        | 1940-01-02 | 584.40 | 584.40               |
| 18 | 18 | 18        | 1940-01-02 | 846.15 | 846.15               |
| 19 | 19 | 19        | 1940-01-02 | 269.28 | 269.28               |

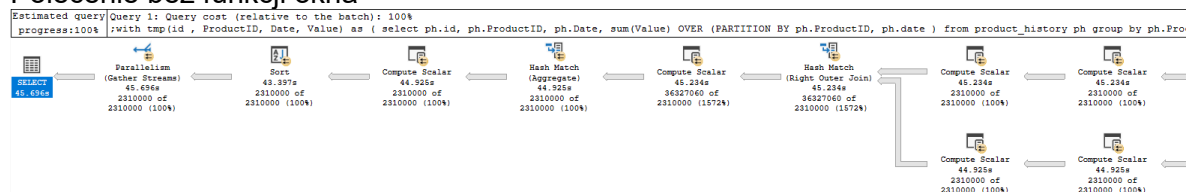
### a) SQL

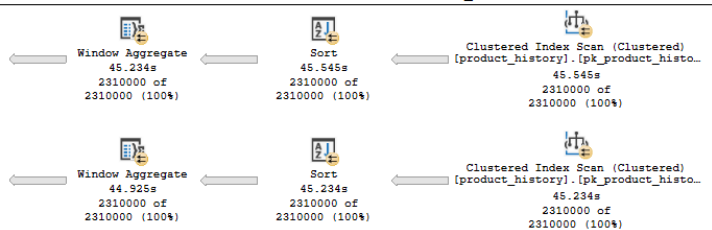
Polecenie z funkcją okna



Estimated Subtree Cost: 38,236

### Polecenie bez funkcji okna





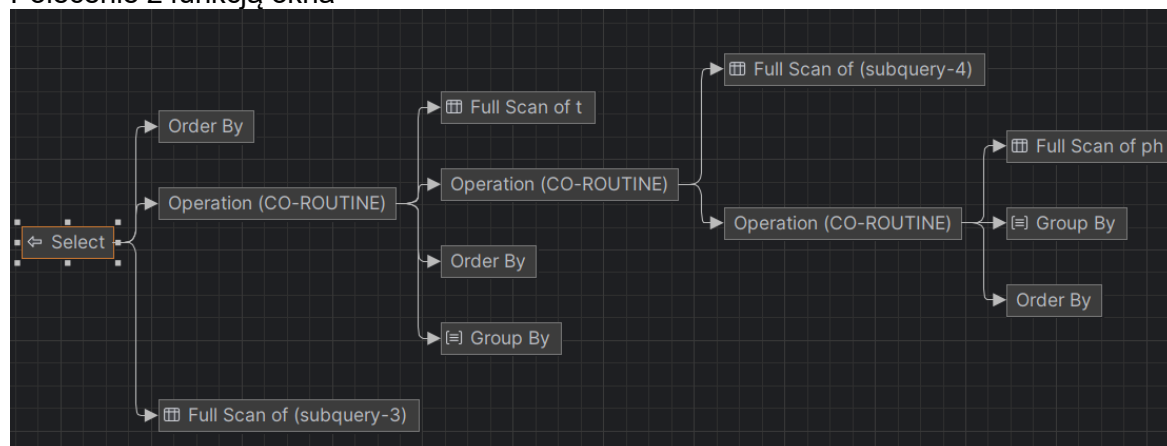
Estimated Subtree Cost: 65,3587

Porównanie

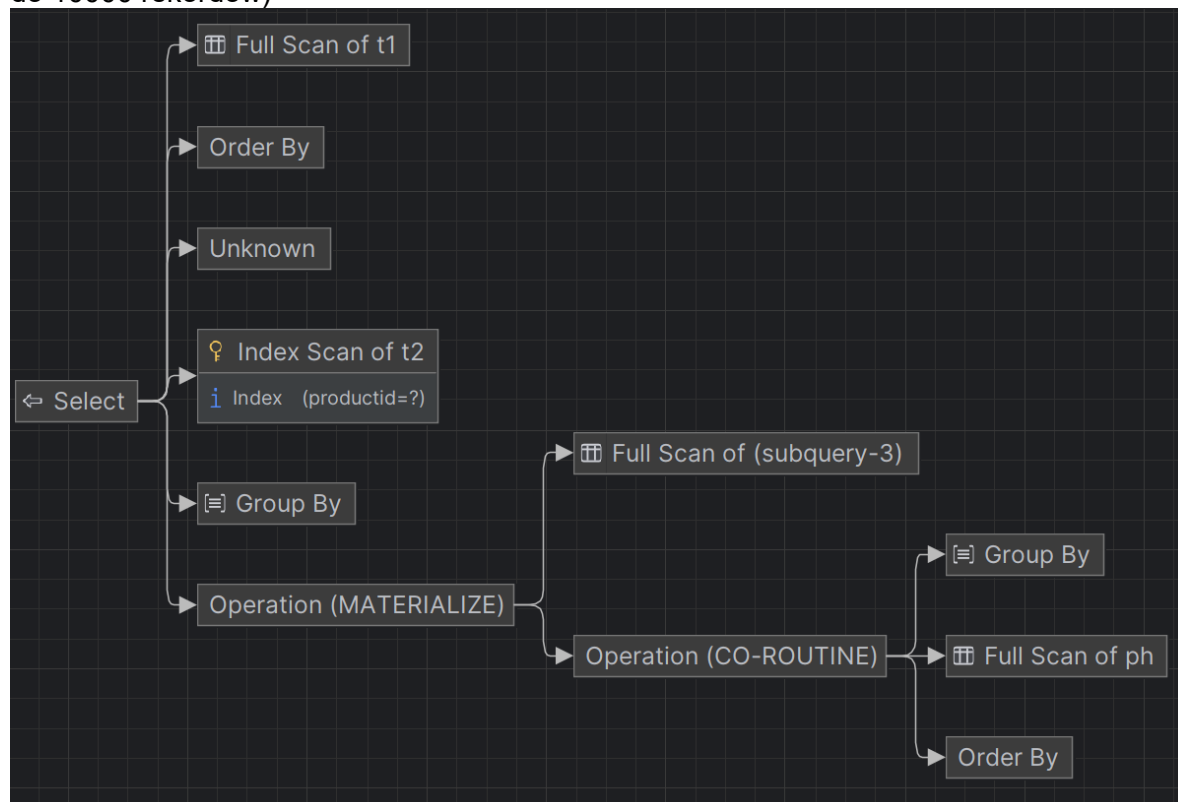
Polecenie z funkcją okna Estimated Subtree Cost: 38,236 jest wydajniejsze od polecenia bez funkcji okna Estimated Subtree Cost: 65,3587

b) Sqlite

Polecenie z funkcją okna

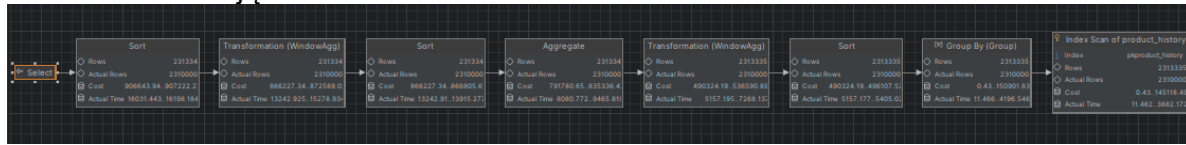


Polecenie bez funkcji okna ( z powodów wydajnościowych ograniczono tabelę wyjściową do 10000 rekordów)



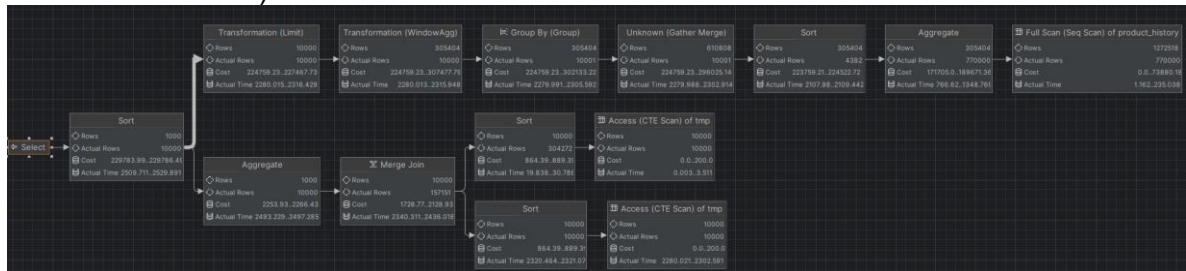


### c) Postgres: Polecenie z funkcją okna



Cost: 90664394.907222.2

Polecenie bez funkcji okna ( z powodów wydajnościowych ograniczono tabelę wyjściową do 10000 rekordów)



Brak możliwości wykonania zapytania w sensownym czasie dla wersji bez funkcji okna znów pokazuje ich przewagę nad innymi rozwiązaniami. Dla wersji zapytania gdzie została ograniczona liczba rekordów do 10 tys, zapytanie dla serwera sqlite wykonało się szybciej. Jeśli chodzi o diagramy wykonania widać, że w porównaniu do np. Postgres dla serwera SQLite są one bardziej rozgałęzione i rozłożone, wynika z nich też że wykonuje się tam więcej niezależnych skanowań tabel.

## Zadanie 15

Wykonaj kilka "własnych" przykładowych analiz. Czy są jeszcze jakieś ciekawe/przydatne funkcje okna (z których nie korzystałeś w ćwiczeniu)? Spróbuj ich użyć w zaprezentowanych przykładach.

### PERCENT\_RANK()

Pokazuje jaką część rekordów z wyłączeniem obecnego, ma daną wartość mniejszą od wartości dla rekordu dla którego zwracana jest wartość

Przykładowe zapytanie:

```
;with order_values(YearOrderDate, MonthOrderDate, OrderValue)
as(
    select YEAR(o.OrderDate),
           MONTH(o.OrderDate),
           sum(od.Quantity * od.UnitPrice) + sum(o.Freight)
    from Orders o
    join [Order Details] od on od.OrderID = o.OrderID
    group by YEAR(OrderDate), MONTH(o.OrderDate)
)
select *, CUME_DIST() over (order by OrderValue) as percentOfLessOrEq
from order_values
```

Wyniki:

|    | YearOrderDate | MonthOrderDate | OrderValue | percent_rank  |
|----|---------------|----------------|------------|---------------|
| 1  | 1998          | 5              | 22473,41   | 0             |
| 2  | 1996          | 9              | 30943,37   | 0,04545454... |
| 3  | 1996          | 8              | 30957,83   | 0,09090909... |
| 4  | 1996          | 7              | 34192,98   | 0,13636363... |
| 5  | 1997          | 6              | 44602,03   | 0,18181818... |
| 6  | 1997          | 2              | 46306,64   | 0,22727272... |
| 7  | 1997          | 3              | 46597,08   | 0,27272727... |
| 8  | 1996          | 10             | 46626,89   | 0,31818181... |
| 9  | 1997          | 11             | 51953,82   | 0,36363636... |
| 10 | 1996          | 11             | 55689,35   | 0,40909090... |
| 11 | 1997          | 8              | 59668,25   | 0,45454545... |
| 12 | 1996          | 12             | 59959,61   | 0,5           |
| 13 | 1997          | 7              | 64086,30   | 0,54545454... |
| 14 | 1997          | 4              | 65676,78   | 0,59090909... |
| 15 | 1997          | 5              | 69095,20   | 0,63636363... |
| 16 | 1997          | 9              | 70667,78   | 0,68181818... |
| 17 | 1997          | 1              | 73715,30   | 0,72727272... |
| 18 | 1997          | 10             | 84376,10   | 0,77272727... |
| 19 | 1997          | 12             | 88435,54   | 0,81818181... |
| 20 | 1998          | 2              | 115103,03  | 0,86363636... |
| 21 | 1998          | 1              | 119882,27  | 0,90909090... |
| 22 | 1998          | 3              | 125938,04  | 0,95454545... |
| 23 | 1998          | 4              | 154817,09  | 1             |

### CUME\_DIST()

Pokazuje jaką część rekordów ma daną wartość mniejszą lub równą od wartości dla rekordu dla którego zwracana jest wartość

Przykładowe zapytanie:

```

;with order_values(YearOrderDate, MonthOrderDate, OrderValue)
as(
    select YEAR(o.OrderDate),
           MONTH(o.OrderDate),
           sum(od.Quantity * od.UnitPrice) + sum(o.Freight)
    from Orders o
    join [Order Details] od on od.OrderID = o.OrderID
    group by YEAR(OrderDate) , MONTH(o.OrderDate)
)
select * , CUME_DIST() over (order by OrderValue) as percentOfLessOrEq
from order_values

```

Wyniki:

|    | YearOrderDate | MonthOrderDate | OrderValue | cume_dist          |
|----|---------------|----------------|------------|--------------------|
| 1  | 1998          | 5              | 22473.41   | 0,0434782608695652 |
| 2  | 1996          | 9              | 30943,37   | 0,0869565217391304 |
| 3  | 1996          | 8              | 30957,83   | 0,130434782608696  |
| 4  | 1996          | 7              | 34192,98   | 0,173913043478261  |
| 5  | 1997          | 6              | 44602,03   | 0,217391304347826  |
| 6  | 1997          | 2              | 46306,64   | 0,260869565217391  |
| 7  | 1997          | 3              | 46597,08   | 0,304347826086957  |
| 8  | 1996          | 10             | 46626,89   | 0,347826086956522  |
| 9  | 1997          | 11             | 51953,82   | 0,391304347826087  |
| 10 | 1996          | 11             | 55689,35   | 0,434782608695652  |
| 11 | 1997          | 8              | 59668,25   | 0,478260869565217  |
| 12 | 1996          | 12             | 59959,61   | 0,521739130434783  |
| 13 | 1997          | 7              | 64086,30   | 0,565217391304348  |
| 14 | 1997          | 4              | 65676,78   | 0,608695652173913  |
| 15 | 1997          | 5              | 69095,20   | 0,652173913043478  |
| 16 | 1997          | 9              | 70667,78   | 0,695652173913043  |
| 17 | 1997          | 1              | 73715,30   | 0,739130434782609  |
| 18 | 1997          | 10             | 84376,10   | 0,782608695652174  |
| 19 | 1997          | 12             | 88435,54   | 0,826086956521739  |
| 20 | 1998          | 2              | 115103,03  | 0,869565217391304  |
| 21 | 1998          | 1              | 119882,27  | 0,91304347826087   |
| 22 | 1998          | 3              | 125938,04  | 0,956521739130435  |
| 23 | 1998          | 4              | 154817,09  | 1                  |

### Funkcja NTILE()

Umożliwia równomierne podzielenie danych na n grup i przypisanie każdej grupie numeru id. Jeśli liczba rekordów nie dzieli się dokładnie przez n, to różnice w liczbie rekordów między grupami nie przekraczają 1.

Przykład – prosty podział pracowników na 4 grupy:

```
select EmployeeId,
       LastName,
       FirstName,
       Title,
       NTILE(4) over (ORDER BY Title) as GroupID
from Employees
```

Wynik:

|   | EmployeeId | LastName  | FirstName | Title                    | GroupID |
|---|------------|-----------|-----------|--------------------------|---------|
| 1 | 8          | Callahan  | Laura     | Inside Sales Coordinator | 1       |
| 2 | 5          | Buchanan  | Steven    | Sales Manager            | 1       |
| 3 | 6          | Suyama    | Michael   | Sales Representative     | 1       |
| 4 | 7          | King      | Robert    | Sales Representative     | 2       |
| 5 | 9          | Dodsworth | Anne      | Sales Representative     | 2       |
| 6 | 1          | Davolio   | Nancy     | Sales Representative     | 3       |
| 7 | 3          | Leverling | Janet     | Sales Representative     | 3       |
| 8 | 4          | Peacock   | Margaret  | Sales Representative     | 4       |
| 9 | 2          | Fuller    | Andrew    | Vice President, Sales    | 4       |

# Punktacja

| zadanie | pkt |
|---------|-----|
| 1       | 0,5 |
| 2       | 0,5 |
| 3       | 1   |
| 4       | 1   |
| 5       | 0,5 |
| 6       | 2   |
| 7       | 2   |
| 8       | 0,5 |
| 9       | 2   |
| 10      | 1   |
| 11      | 2   |
| 12      | 1   |
| 13      | 2   |
| 14      | 2   |
| 15      | 2   |
| razem   | 20  |