

SQL - Funkcje okna (Window functions)

Lab 1-2

Imię i nazwisko:

Jacek Budny, Mateusz Kleszcz

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 miejsca oznaczone jako:

```
-- wyniki ...
```

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
 - SSMS - Microsoft SQL Managment Studio
 - DDataGrip lub DBeaver
- Przykładowa baza Northwind
 - W wersji dla każdego z wymienionych serwerów

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>

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?

Grupowanie danych zwraca zagregowane wyniki, podczas gdy funkcje okna zwracają wyniki osobno dla wszystkich kolumn. Czas wykonania zapytania dla funkcji okna jak i dla grupowania danych jest zbliżony.

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?

Funkcja okna wykonuje się po klauzuli where, dlatego zwraca średnią cenę produktów jedynie z id mniejszym niż 10. Jeżeli użyjemy podzapytania, to średnia zostanie policzona z całej tabeli, a następnie zostaną wyświetlone produkty z id mniejszym niż 10.

Napisz polecenie równoważne

- i. z wykorzystaniem funkcji okna. Napisz polecenie równoważne

```
select top 9 p.productid, p.ProductName, p.unitprice,  
            avg(unitprice) over () as avgprice  
from products p  
order by ProductID
```

- ii. z wykorzystaniem podzapytania

```

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

```

Zadanie 3

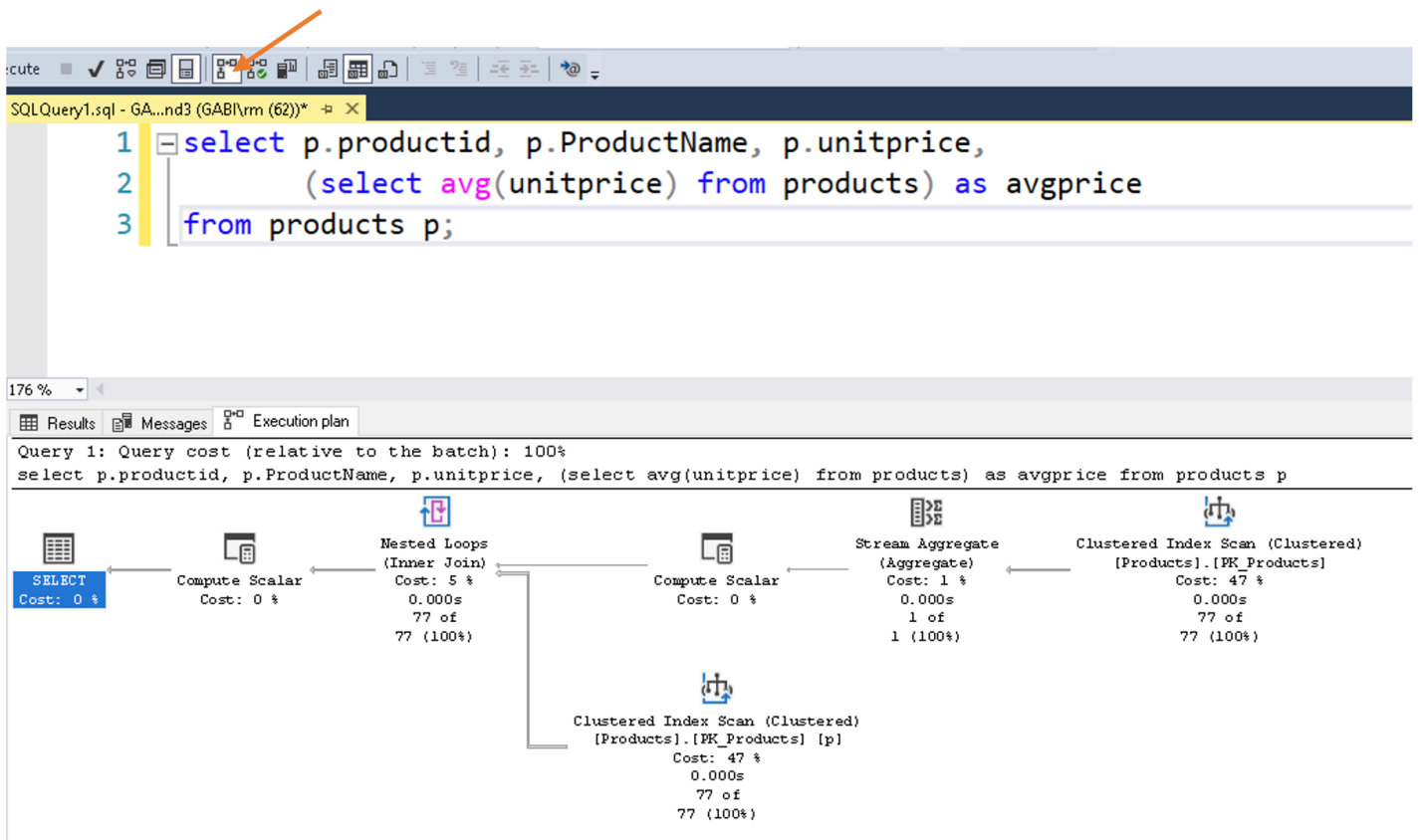
Baza: Northwind, tabela: products

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


```
select p.ProductID, p.ProductName, p.UnitPrice,  
       (select avg(UnitPrice) from products) as avgprice  
from products p
```

join - 110ms - 156ms

```
select p.ProductID, p.ProductName, p.UnitPrice,  
       (select avg(UnitPrice) from products) as avgprice  
from products p  
cross join (select avg(UnitPrice) as avgprice from products) as prod
```

okna - 82ms - 112ms

```
select p.ProductID, p.ProductName, p.UnitPrice, avg(unitprice) over () as avgprice  
from products p;
```

Powyższe zapytania testowaliśmy w obrębie MS SQL Server.

Czasy wykonania poszczególnych zapytań różnią się na tyle nieznacznie, że może być to błąd pomiaru.

Największe czasy uzyskujemy przy użyciu zapytania z JOIN'em.

Jest to też zapytanie najbardziej skomplikowane do napisania.

Przetestowaliśmy również działanie funkcji okna w 3 SZBD.

Czasy różniły się znacznie: dla MS SQL Server uzyskaliśmy czas 130ms, w Postgresie 80ms i w SQLite 68ms.

Po przeanalizowaniu zapytań za pomocą "Explain Plain" w poszczególnych systemach, zauważyliśmy na diagramie, że MS SQL Server wykonuje znacznie więcej operacji w ramach wywołania funkcji (m. in. ma operację INNER JOIN, skanowanie indeksów i agregację), dlatego może być najwolniejsza.

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)

podzapytanie -79ms

```
select p.ProductID, p.ProductName, p.UnitPrice,  
       (select avg(unitprice) from Products where p.CategoryID = CategoryID) as avgprice  
from Products p  
where p.UnitPrice > (select avg(unitprice) from Products where p.CategoryID = CategoryID)
```

join - 74ms

```
SELECT p.productid, p.ProductName, p.unitprice, avgprice  
FROM products p  
LEFT JOIN (SELECT CategoryID, avg(unitprice) AS avgprice FROM products group by CategoryID) AS prod  
ON prod.CategoryID = p.CategoryID  
WHERE p.UnitPrice > avgprice
```

okna - 82ms

```
with t as (  
    select p.ProductID, p.ProductName, p.UnitPrice,  
           avg(unitprice) over (partition by CategoryID) as avgprice from products p  
) select * from t where t.UnitPrice > t.avgprice
```

Wszystkie serwery wykonały podzapytania w identycznym czasie. Porównanie czasu dla tak małych danych nie ma większego sensu. Każda z funkcji wymagała użycia sumarycznie przynajmniej 2 zapytań, co wynika z tego, że warunek, który sprawdzamy, był zawarty w klauzuli WHERE, która wykona się jako pierwsza.

Tym razem nie zaobserwowaliśmy różnic w czasie pomiędzy różnymi SZBD

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 Server

```

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 Sserver

```

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

Dla MS SQL Servera - 1s414ms

Dla Postgres - 354ms

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)

- Podzapytanie

```
select ph.id, ph.productid, ph.ProductName, ph.unitprice,
(select avg(unitprice) from product_history where CategoryID = ph.CategoryID) as avgprice
from product_history ph
where ph.UnitPrice > (select avg(unitprice) from product_history where CategoryID = ph.CategoryID);
```

- join

```
select ph1.id, ph1.productid, ph1.ProductName, ph1.unitprice, avg(ph2.UnitPrice) as avgPrice
from product_history ph1
join product_history ph2
on ph2.CategoryID = ph1.CategoryID
group by ph1.id, ph1.productid, ph1.ProductName, ph1.unitprice
having ph1.unitprice > avg(ph2.UnitPrice);
```

- okna

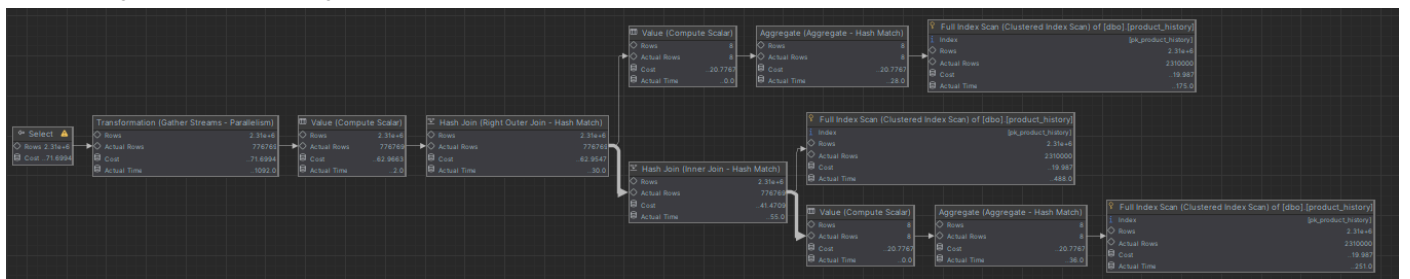
```
with new_product AS (
select ph.id, ph.productid, ph.ProductName, ph.unitprice, AVG(ph.UnitPrice) over (Partition by CategoryID
from product_history ph
)
select nph1.id, nph1.productid, nph1.ProductName, nph1.unitprice, nph1.avgprice
from new_product nph1
where nph1.unitprice > nph1.avgprice
```

Wnioski:

Wynik wykonania zapytania:

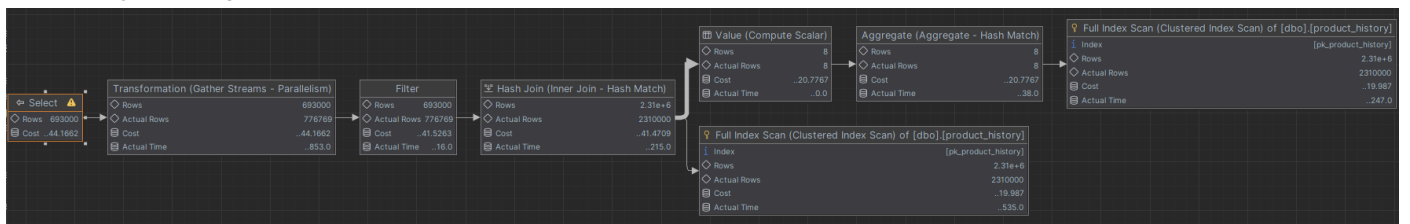
	id	productid	ProductName	unitprice	avgprice
1	24599	36	Inlagd Sill	26.90	20.224992
2	24600	37	Gravad lax	33.13	20.224992
3	24601	38	Côte de Blaye	244.40	28.776081
4	24603	40	Boston Crab Meat	26.37	20.224992
5	24605	42	Singaporean Hokkien Fried Mee	22.45	20.011166
6	24606	43	Ipon Coffee	50.92	28.776081
7	24607	44	Gula Malacca	27.30	21.401616
8	24609	46	Spegesild	20.67	20.224992
9	24612	49	Maxilaku	27.79	22.438569
10	24613	50	Valkoinen suklaa	24.46	22.438569
11	24614	51	Manjimup Dried Apples	57.15	26.003033
12	24616	53	Perth Pasties	39.18	36.699749
13	24619	56	Gnocchi di nonna Alice	43.80	20.011166
14	24620	57	Ravioli Angelo	27.35	20.011166
15	24621	58	Escargots de Bourgogne	21.79	20.224992
16	24622	59	Raclette Courdavault	58.93	24.203494
17	24623	60	Camembert Pierrot	40.25	24.203494
18	24624	61	Sirop d'érable	35.35	21.401616
19	24625	62	Tarte au sucre	53.86	22.438569
20	24626	63	Veggie-spread	49.05	21.401616
21	24627	64	Wimmers gute Semmelknödel	39.58	20.011166
22	24628	65	Louisiana Fiery Hot Pepper Sauce	28.73	21.401616

Plan zapytania z podzapytaniem na SQL Server:



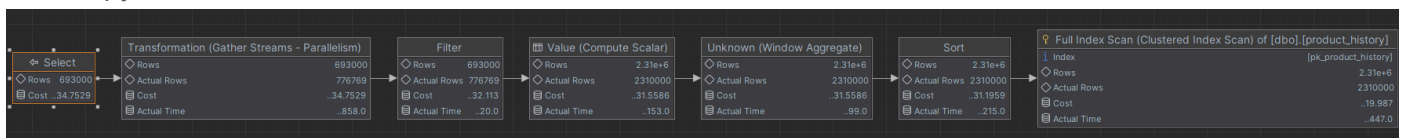
Koszt 71.70

Plan zapytanie z join na SQL Server



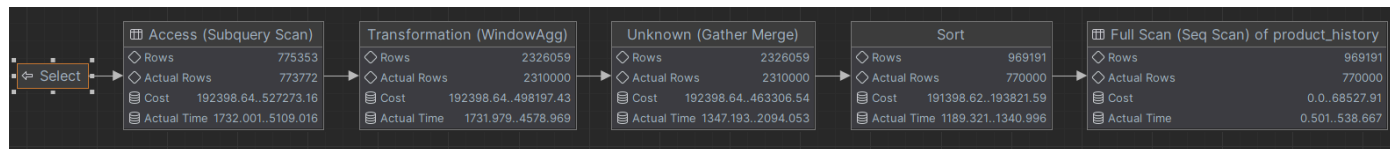
Koszt: 44.17

Plan zapytania z oknem na SQL Serer

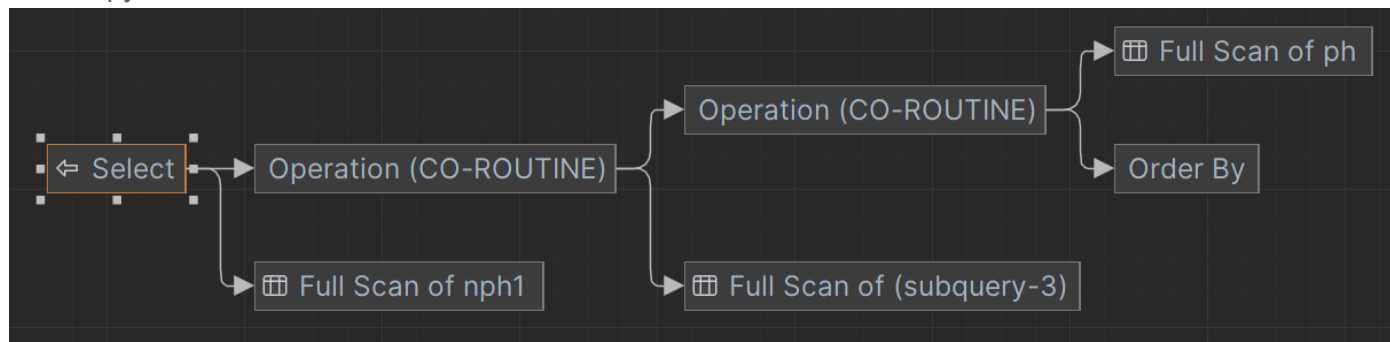


Koszt: 34.75

Plan zapytania z oknem na PostgreSQL:



Plan zapytania z oknem na SQLite:



Dla zapytań i join'ów zarówno w Postgresie, jak i SQLite, wyniki nie były zwracane przez długi czas (przekraczający kilka minut), dlatego zdecydowaliśmy przerwać ich przetwarzanie. Co ciekawe, SQL Server zwrócił wszystkie wyniki już po sekundzie. Postanowiliśmy stworzyć nową tabelę zawierającą tylko 30 000 wierszy (w przeciwieństwie do ponad 2 000 000 wierszy w oryginalnej tabeli product_history). Na nowej tabeli czasy wykonania były nadal znacznie większe niż na SQL Serverze.

Dla funkcji okna wszystkie bazy danych testowaliśmy na tabeli z pierwotną liczbą rekordów. Jednakże, jeśli chodzi o funkcje okna, różnica między SQLite a PostgreSQL jest niewielka, a czasy wynoszą około 4 sekund, podczas gdy dla SQLite czas wykonania wynosił około 1 sekundę.

Pomimo większej złożoności planu w SQL Server, czas wykonania dla każdej komendy był znacząco krótszy.

Dla podzapytania z oknem widać, że Full scan index wykonywany jest tylko jeden raz.

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)

- podzapytania, na postgresie trzeba zamienić year na extract(year from ...)

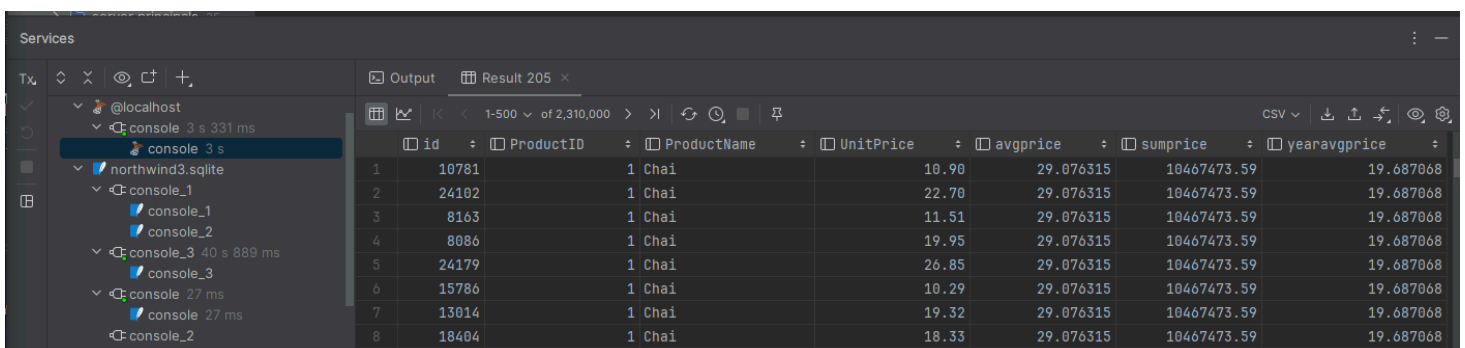
```
select p.id, p.ProductID, p.ProductName, p.UnitPrice,
       (select avg(UnitPrice) from product_history where CategoryID = p.CategoryID) as avgprice,
       (select sum(UnitPrice) from product_history where CategoryID = p.CategoryID) as sumprice,
       (select avg(UnitPrice) from product_history
        where ProductID = p.ProductID and year(date) = year(p.date)) as yearavgprice
from product_history p
```

- join

```
select p.id, p.ProductID, p.ProductName, p.UnitPrice,
       avg(p2.UnitPrice) as avgprice,
       sum(p2.UnitPrice) as sumprice,
       avg(p3.UnitPrice) as avgYear
from product_history p
join product_history p2 on p2.CategoryID = p.CategoryID
join product_history p3 on p3.productid = p.productid and year(p.date) = year(p3.date)
group by p.id, p.ProductID, p.ProductName, p.UnitPrice
```

- okna

```
select p.id, p.ProductID, p.ProductName, p.UnitPrice,
       avg(unitprice) over (partition by CategoryID) as avgprice,
       sum(unitprice) over (partition by CategoryID) as sumprice,
       avg(unitprice) over (partition by ProductID, year(p.date)) as yearavgprice
from product_history p
```



The screenshot shows a database client interface with a sidebar on the left displaying the file structure of 'northwind3.sqlite'. The main window shows the 'Result 205' tab with a table of 8 rows. The table has columns: id, ProductID, ProductName, UnitPrice, avgprice, sumprice, and yearavgprice. The data is as follows:

id	ProductID	ProductName	UnitPrice	avgprice	sumprice	yearavgprice
1	10781	1 Chai	10.90	29.076315	10467473.59	19.687068
2	24102	1 Chai	22.70	29.076315	10467473.59	19.687068
3	8163	1 Chai	11.51	29.076315	10467473.59	19.687068
4	8086	1 Chai	19.95	29.076315	10467473.59	19.687068
5	24179	1 Chai	26.85	29.076315	10467473.59	19.687068
6	15786	1 Chai	10.29	29.076315	10467473.59	19.687068
7	13014	1 Chai	19.32	29.076315	10467473.59	19.687068
8	18404	1 Chai	18.33	29.076315	10467473.59	19.687068

Zapytanie przy użyciu podzapytań MS SQL Server - 3s226ms

Zapytanie przy użyciu podzapytań Postgresa - nie dało się wykonać w sensownym czasie

Zapytanie przy użyciu podzapytań SQLite - nie dało się wykonać w sensownym czasie

Zapytanie przy użyciu joina nie dało się przetworzyć na żadnym systemie

Zapytanie przy użyciu funkcji okna MS SQL Server - 3s331ms

Zapytanie przy użyciu funkcji okna Postgresa - 7s82ms

Zapytanie przy użyciu funkcji okna SQLite - 5s453ms

Dzięki powyższym wynikom jesteśmy w stanie zauważyć przewagę systemu MS SQL Server nad innymi.

Pomimo, że system ten sprawdzał się gorzej w przypadku mniej złożonych zapytań, tak w tym przypadku jako jedyny poradził sobie np. z podzapytaniami. Sprawdzał się on też najszybciej ze wszystkich systemów.

Funkcje okna, choć trochę wolniejsze od podzapytań, okazały się najbardziej niezawodnym, a przy tym łatwym do napisania sposobem.

Operacja z joinem jest już zbyt skomplikowana, aby za jej pomocą przetwarzać zapytania.

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;
```

`row_number` - numer wiersza

`rank` - ranking elementów, z uwzględnieniem remisów (1, 2, 2, 4)

`dense_rank` - ranking elementów, bez uwzględniania remisów (1, 2, 2, 3)

Zadanie

Spróbuj uzyskać ten sam wynik bez użycia funkcji okna

```

select productid, productname, unitprice, categoryid,
    (select count(*) from products p1
     where p.CategoryID = p1.CategoryID
     and (p.UnitPrice < p1.UnitPrice
     or (p.UnitPrice = p1.UnitPrice and p.ProductID > p1.ProductID)))
    + 1 as myrowno,
    (select count(*) from products p1
     where p1.CategoryID = p.CategoryID
     and p.UnitPrice < p1.UnitPrice)
    + 1 as myrankprice,
    (select count(*) from
        (select distinct UnitPrice
         from products p1
         where p1.CategoryID = p.CategoryID and p.UnitPrice < p1.UnitPrice) as less)
    + 1 as mydenserankprice
from products p order by CategoryID, UnitPrice desc, ProductID;

```

	productid	productname	unitprice	categoryid	myrowno	myrankprice	mydenserankprice
1	38	Côte de Blaye	263.5000	1	1	1	1
2	43	Ipoh Coffee	46.0000	1	2	2	2
3	2	Chang	19.0000	1	3	3	3
4	1	Chai	18.0000	1	4	4	4
5	35	Steeleye Stout	18.0000	1	5	4	4
6	39	Chartreuse verte	18.0000	1	6	4	4
7	76	Lakkalikööri	18.0000	1	7	4	4
8	70	Outback Lager	15.0000	1	8	8	5
9	34	Sasquatch Ale	14.0000	1	9	9	6
10	67	Laughing Lumberjack Lager	14.0000	1	10	9	6
11	75	Rhönbräu Klosterbier	7.7500	1	11	11	7
12	24	Guaraná Fantástica	4.5000	1	12	12	8
13	63	Vegie-spread	43.9000	2	1	1	1
14	8	Northwoods Cranberry Sauce	40.0000	2	2	2	2
15	61	Sirop d'érable	28.5000	2	3	3	3
16	6	Grandma's Boysenberry Spread	25.0000	2	4	4	4
17	4	Chef Anton's Cajun Seasoning	22.0000	2	5	5	5
18	5	Chef Anton's Gumbo Mix	21.3500	2	6	6	6
19	65	Louisiana Fiery Hot Pepper Sauce	21.0500	2	7	7	7
20	44	Gula Melanga	10.4500	2	8	8	8

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

```
with t as (
    select year(ph.Date) as year, p.ProductID, p.ProductName, ph.UnitPrice,
           rank() over (partition by p.ProductID, year(ph.Date)
                        order by ph.UnitPrice desc) as rank
    from Products p
    join product_history ph on ph.ProductID = p.ProductID
)
select * from t
where rank <= 4
order by Year, ProductID, rank;
```

	year	ProductID	ProductName	UnitPrice	rank
1	1940	1	Chai	27.93	1
2	1940	1	Chai	27.93	1
3	1940	1	Chai	27.89	3
4	1940	1	Chai	27.89	3
5	1940	2	Chang	28.93	1
6	1940	2	Chang	28.92	2
7	1940	2	Chang	28.89	3
8	1940	2	Chang	28.88	4
9	1940	3	Aniseed Syrup	19.96	1
10	1940	3	Aniseed Syrup	19.96	1
11	1940	3	Aniseed Syrup	19.94	3
12	1940	3	Aniseed Syrup	19.94	3
13	1940	4	Chef Anton's Cajun Seasoning	31.92	1
14	1940	4	Chef Anton's Cajun Seasoning	31.91	2
15	1940	4	Chef Anton's Cajun Seasoning	31.87	3
16	1940	4	Chef Anton's Cajun Seasoning	31.87	3
17	1940	5	Chef Anton's Gumbo Mix	31.27	1
18	1940	5	Chef Anton's Gumbo Mix	31.26	2
19	1940	5	Chef Anton's Gumbo Mix	31.22	3

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 t as (
    select year(ph.Date) as year, p.ProductID, p.ProductName, ph.UnitPrice,
        (select top 4 count (*) + 1
         from product_history ph1
         where p.ProductID = ph1.productid
              and year(ph.date) = year(ph1.date)
              and ph1.unitprice > ph.unitprice
        ) as rank
    from Products p
        join product_history ph on ph.ProductID = p.ProductID
)
select * from t
where rank <= 4
order by Year, ProductID, rank;

```

Po 30 krotnym :) zmniejszeniu tabeli wynikowej udało nam się uzyskać wynik 6s264ms dla MS SQL Servera (dla pozostałych SZBD nie udało się uzyskać wyniku nawet wtedy). Funkcje okna poradziły sobie z podanym zadaniem w 136ms. Oprócz tego rozwiązanie z funkcjami okna jest zdecydowanie dużo prostsze do napisania.

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;

```

Lag pokazuje wartość poprzedniego rekordu, lead następnego.

W drugim zapytaniu where zadziała dopiero po wykonaniu with i dlatego będziemy mieli wartość dla pierwszego produktu, w przypadku pierwszego zapytania where wykona się od razu i pierwszy rekord będzie miał wartość null. Obydwa zapytania zwracają null dla ostatniego rekordu.

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 ph.productid, ph.productname, ph.categoryid, ph.date, ph.unitprice,
       (select ph1.unitprice
        from product_history ph1
        where ph1.productid = ph.productid and ph1.date = dateadd(day, -1, ph.date)
       ) as previousprodprice,
       (select ph1.unitprice
        from product_history ph1
        where ph1.productid = ph.productid and ph1.date = dateadd(day, 1, ph.date)
       ) as nextprodprice
from product_history ph
where ph.productid = 1 and year(ph.date) = 2022
order by ph.date;

```

Services

Database

@localhost

console 9 s 217 ms

console 9 s

northwind3.sqlite

console_3 25 ms

console_3 25 ms

postgres@localhost

console 30 s 500 ms

console 30 s

Output

Result 24

Result 23

50 rows

CSV

Download

Copy

Refresh

uctid	productname	categoryid	date	unitprice	previousprodprice	nextprodprice
1	1 Chai		1 2022-01-01	12.40	26.34	12.23
2	1 Chai		1 2022-01-02	12.23	12.40	19.68
3	1 Chai		1 2022-01-03	19.68	12.23	10.77
4	1 Chai		1 2022-01-04	10.77	19.68	12.95
5	1 Chai		1 2022-01-05	12.95	10.77	22.20
6	1 Chai		1 2022-01-06	22.20	12.95	16.39
7	1 Chai		1 2022-01-07	16.39	22.20	15.40
8	1 Chai		1 2022-01-08	15.40	16.39	16.32
9	1 Chai		1 2022-01-09	16.32	15.40	26.71
10	1 Chai		1 2022-01-10	26.71	16.32	19.10
11	1 Chai		1 2022-01-11	19.10	26.71	19.88
12	1 Chai		1 2022-01-12	19.88	19.10	19.91
13	1 Chai		1 2022-01-13	19.91	19.88	10.82
14	1 Chai		1 2022-01-14	10.82	19.91	11.85
15	1 Chai		1 2022-01-15	11.85	10.82	17.73
16	1 Chai		1 2022-01-16	17.73	11.85	10.62
17	1 Chai		1 2022-01-17	10.62	17.73	26.56
18	1 Chai		1 2022-01-18	26.56	10.62	23.57

Zapytanie z funkcją okna wykonywało się 276ms, a zapytanie z selectem 9s217ms. Dodatkowo, wykorzystaliśmy tutaj fakt, że w kolumnie date znajdują się daty co 1 dzień, jeżeli którejś daty by brakowało to zapytanie nie zadziałałoby. Rozwiązaniem mogłoby być użycie funkcji okna row_number (ale jeżeli zakładamy że ich nie używamy, to musielibyśmy zrobić selecta z countem, co prawdopodobnie sprawiłoby że nie uzyskalibyśmy wyników w sensownym czasie)

Dla Postgresa i SQLitea zapytanie się nie wykonało nawet na zmniejszonym zbiorze wynikowym.

Zadanie 11

Baza: Northwind, tabele customers, orders, order details

Napisz polecenie które wyświetla inf. o zamówieniach

Zbiór wynikowy 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.

```

with t as(
    select C.ContactName, O.OrderID, O.OrderDate,
           sum(OD.UnitPrice * OD.Quantity * (1 - OD.Discount)) + O.Freight as OrderValue,
           lag(O.OrderID) over ( partition by C.CustomerID order by O.OrderDate) as PrevOrderID,
           lag(O.OrderDate) over ( partition by C.CustomerID order by O.OrderDate) as PrevOrderDate
    from Orders O
    join [Order Details] OD
    on O.OrderID = OD.OrderID
    join Customers C
    on O.CustomerID = C.CustomerID
    group by O.OrderID, O.Freight, O.OrderDate, C.CustomerID, C.ContactName)
select t.*, PD.OrderValue
from t
    left join (select O.OrderID,
                      sum(OD.UnitPrice * OD.Quantity * (1 - OD.Discount)) + O.Freight as OrderValue
                from Orders O
                join dbo.[Order Details] OD
                on O.OrderID = OD.OrderID
                group by O.OrderID, O.Freight) as PD
    on t.PrevOrderID = PD.OrderID

```

	ContactName	OrderID	OrderDate	t.OrderValue	PrevOrderID	PrevOrderDate	PD.OrderValue
1	Alexander Feuer	10277	1996-08-09 00:00:00.000	1326.5699877929687	<null>	<null>	<null>
2	Alexander Feuer	10575	1997-06-20 00:00:00.000	2274.7399938964845	10277	1996-08-09 00:00:00.000	1326.5699877929687
3	Alexander Feuer	10699	1997-10-09 00:00:00.000	114.58	10575	1997-06-20 00:00:00.000	2274.7399938964845
4	Alexander Feuer	10779	1997-12-16 00:00:00.000	1393.13	10699	1997-10-09 00:00:00.000	114.58
5	Alexander Feuer	10945	1998-03-12 00:00:00.000	255.22	10779	1997-12-16 00:00:00.000	1393.13
6	Ana Trujillo	10308	1996-09-18 00:00:00.000	90.40999923706055	<null>	<null>	<null>
7	Ana Trujillo	10625	1997-08-08 00:00:00.000	523.65	10308	1996-09-18 00:00:00.000	90.40999923706055
8	Ana Trujillo	10759	1997-11-28 00:00:00.000	331.99	10625	1997-08-08 00:00:00.000	523.65
9	Ana Trujillo	10926	1998-03-04 00:00:00.000	554.3200015258789	10759	1997-11-28 00:00:00.000	331.99
10	André Fonseca	10423	1997-01-23 00:00:00.000	1044.5	<null>	<null>	<null>
11	André Fonseca	10652	1997-09-01 00:00:00.000	325.97499908447264	10423	1997-01-23 00:00:00.000	1044.5
12	André Fonseca	10685	1997-09-29 00:00:00.000	834.8499984741211	10652	1997-09-01 00:00:00.000	325.97499908447264
13	André Fonseca	10709	1997-10-17 00:00:00.000	3634.8	10685	1997-09-29 00:00:00.000	834.8499984741211
14	André Fonseca	10734	1997-11-07 00:00:00.000	1499.9800061035157	10709	1997-10-17 00:00:00.000	3634.8
15	André Fonseca	10777	1997-12-15 00:00:00.000	227.01	10734	1997-11-07 00:00:00.000	1499.9800061035157
16	André Fonseca	10790	1997-12-22 00:00:00.000	750.73	10777	1997-12-15 00:00:00.000	227.01
17	André Fonseca	10959	1998-03-18 00:00:00.000	136.73	10790	1997-12-22 00:00:00.000	750.73
18	André Fonseca	11049	1998-04-24 00:00:00.000	281.93999847412107	10959	1998-03-18 00:00:00.000	136.73
19	Ann Devon	10364	1996-11-26 00:00:00.000	1021.97	<null>	<null>	<null>
20	Ann Devon	10400	1997-01-01 00:00:00.000	3146.93	10364	1996-11-26 00:00:00.000	1021.97
21	Ann Devon	10532	1997-05-09 00:00:00.000	870.8100061035157	10400	1997-01-01 00:00:00.000	3146.93
22	Ann Devon	10726	1997-11-03 00:00:00.000	671.56	10532	1997-05-09 00:00:00.000	870.8100061035157
23	Ann Devon	10987	1998-03-31 00:00:00.000	2957.48	10726	1997-11-03 00:00:00.000	671.56
24	Ann Devon	11024	1998-04-15 00:00:00.000	2041.1699975585936	10987	1998-03-31 00:00:00.000	2957.48
25	Ann Devon	11047	1998-04-24 00:00:00.000	864.495	11024	1998-04-15 00:00:00.000	2041.1699975585936
26	Ann Devon	11056	1998-04-28 00:00:00.000	4018.96	11047	1998-04-24 00:00:00.000	864.495
27	Annette Roulet	10350	1996-11-11 00:00:00.000	706.2499975585938	<null>	<null>	<null>
28	Annette Roulet	10358	1996-11-20 00:00:00.000	449.0399900817871	10350	1996-11-11 00:00:00.000	706.2499975585938

Do uzyskania dostęp do danych o poprzednim zamówieniu świetnie nadaje się funkcja lag()

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;
```

Można zauważyć że zapytanie w tej formie nie pokazuje poprawnie najtańszego produktu w danej kategorii. Wynika to z faktu że funkcje `last_value()`, oraz `first_value()` domyślnie wykorzystują zakres `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`, co w naszym przypadku powoduje niepoprawne wyświetlanie produktu najtańszego w danej kategorii. Poniżej poprawiona wersja zapytania:

```
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;
```

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)

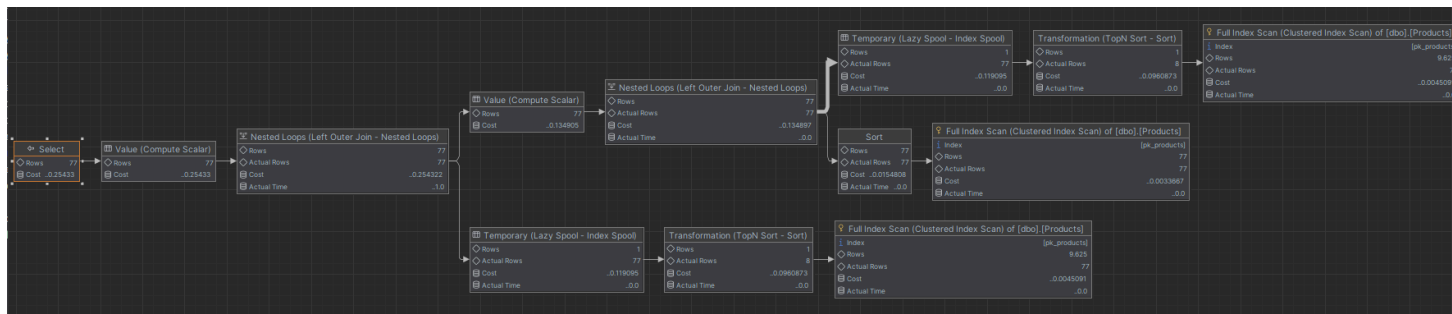
```

select productid, productname, unitprice, categoryid,
(select top 1 ProductName
from Products p2
where p2.CategoryID = p.CategoryID
order by UnitPrice desc) as last,
(select top 1 ProductName
from Products p2
where p2.CategoryID = p.CategoryID
order by UnitPrice) as first
from products p
order by categoryid, unitprice desc;

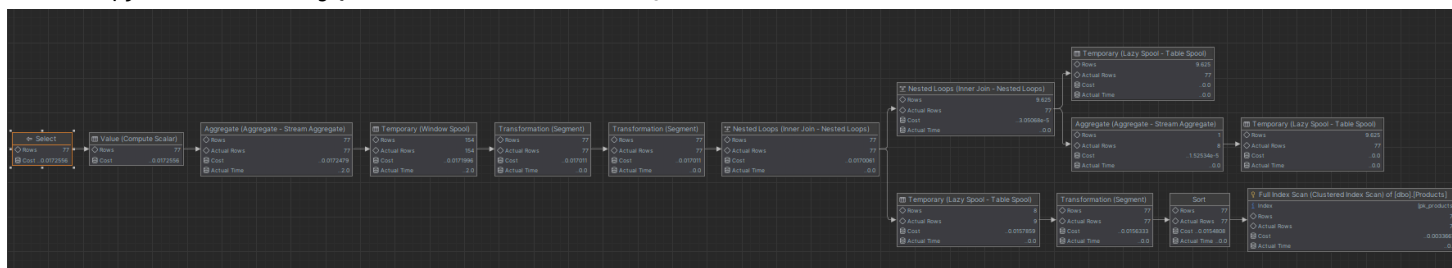
```

	productid	productname	unitprice	categoryid	last	first
1	38	Côte de Blaye	263.5000	1	Côte de Blaye	Guaraná Fantástica
2	43	Iphoh Coffee	46.0000	1	Côte de Blaye	Guaraná Fantástica
3	2	Chang	19.0000	1	Côte de Blaye	Guaraná Fantástica
4	1	Chai	18.0000	1	Côte de Blaye	Guaraná Fantástica
5	39	Chartreuse verte	18.0000	1	Côte de Blaye	Guaraná Fantástica
6	35	Steeleye Stout	18.0000	1	Côte de Blaye	Guaraná Fantástica
7	76	Lakkalikööri	18.0000	1	Côte de Blaye	Guaraná Fantástica
8	70	Outback Lager	15.0000	1	Côte de Blaye	Guaraná Fantástica
9	67	Laughing Lumberjack Lager	14.0000	1	Côte de Blaye	Guaraná Fantástica
10	34	Sasquatch Ale	14.0000	1	Côte de Blaye	Guaraná Fantástica
11	75	Rhönbräu Klosterbier	7.7500	1	Côte de Blaye	Guaraná Fantástica
12	24	Guaraná Fantástica	4.5000	1	Côte de Blaye	Guaraná Fantástica
13	63	Vegie-spread	43.9000	2	Vegie-spread	Aniseed Syrup
14	8	Northwoods Cranberry Sauce	40.0000	2	Vegie-spread	Aniseed Syrup
15	61	Sirop d'érable	28.5000	2	Vegie-spread	Aniseed Syrup
16	6	Grandma's Boysenberry Spread	25.0000	2	Vegie-spread	Aniseed Syrup
17	4	Chef Anton's Cajun Seasoning	22.0000	2	Vegie-spread	Aniseed Syrup
18	5	Chef Anton's Gumbo Mix	21.3500	2	Vegie-spread	Aniseed Syrup
19	65	Louisiana Fiery Hot Pepper Sauce	21.0500	2	Vegie-spread	Aniseed Syrup
20	44	Gula Malacca	19.4500	2	Vegie-spread	Aniseed Syrup
21	66	Louisiana Hot Spiced Okra	17.0000	2	Vegie-spread	Aniseed Syrup
22	15	Genen Shouyu	15.5000	2	Vegie-spread	Aniseed Syrup
23	77	Original Frankfurter grüne Soße	13.0000	2	Vegie-spread	Aniseed Syrup
24	3	Aniseed Syrup	10.0000	2	Vegie-spread	Aniseed Syrup
25	20	Sir Rodney's Marmalade	81.0000	3	Sir Rodney's Marmalade	Teatime Chocolate Biscuits

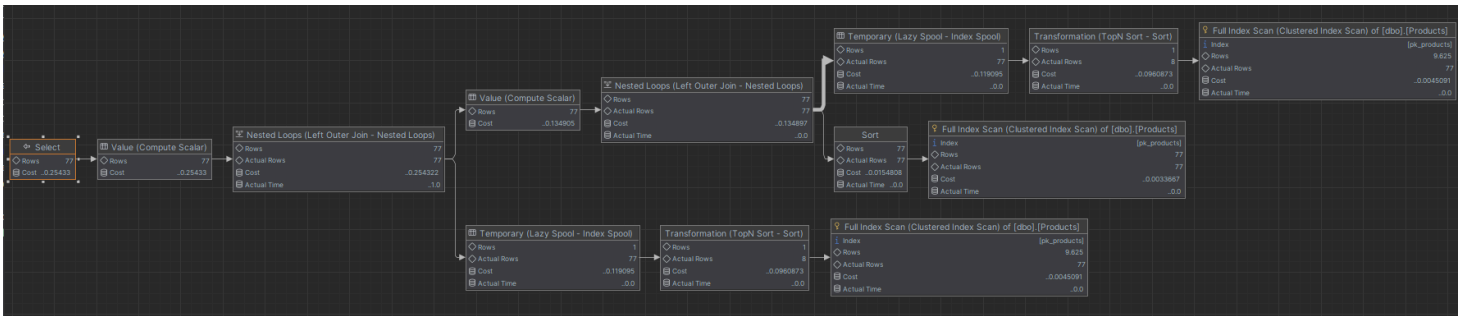
Plan zapytania z funkcją okna na serwerze PostgreSQL



Plan zapytania z funkcją okna na serwerze MS SQL



Plan zapytania bez funkcji okna na serwerze MS SQL



Porównując plany zapytań ciężko dojść do wniosku które zapytanie jest bardziej optymalne. Różnica zauważalna jest podczas porównania kosztów zapytań w wersji z funkcją okna jest to 0.0173 a w wersji bez funkcji okna 0.254. Widać więc że wykorzystanie funkcji okna jest dobrym wyborem, zmniejszającym koszt wykonania zapytania.

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 t as
(
select C.CustomerID, O.OrderID, O.OrderDate,
       sum(OD.UnitPrice * OD.Quantity * (1 - OD.Discount)) + O.Freight as OrderValue
from Orders O
join [Order Details] OD
on O.OrderID = OD.OrderID
join Customers C
on O.CustomerID = C.CustomerID
group by C.CustomerID, O.OrderID, O.OrderDate, O.Freight)
select *,
first_value(concat(t.OrderID, ' ', t.OrderDate, ' ', t.OrderValue))
over (partition by t.CustomerID, month(t.OrderDate) order by t.OrderValue) as MinValueOrder,
first_value(concat(t.OrderID, ' ', t.OrderDate, ' ', t.OrderValue))
over (partition by t.CustomerID, month(t.OrderDate) order by t.OrderValue desc) as MaxValueOrder
from t

```

	CustomerID	OrderID	OrderDate	OrderValue	MinValueOrder	MaxValueOrder
1	ALFKI	10835	1998-01-15 00:00:00.000	915.3300011444092	10835 Jan 15 1998 12:00AM 915.33	10835 Jan 15 1998 12:00AM 915.33
2	ALFKI	10952	1998-03-16 00:00:00.000	511.6199969482422	10952 Mar 16 1998 12:00AM 511.62	10952 Mar 16 1998 12:00AM 511.62
3	ALFKI	11011	1998-04-09 00:00:00.000	934.71	11011 Apr 9 1998 12:00AM 934.71	11011 Apr 9 1998 12:00AM 934.71
4	ALFKI	10643	1997-08-25 00:00:00.000	843.96	10643 Aug 25 1997 12:00AM 843.96	10643 Aug 25 1997 12:00AM 843.96
5	ALFKI	10692	1997-10-03 00:00:00.000	939.02	10702 Oct 13 1997 12:00AM 353.94	10692 Oct 3 1997 12:00AM 939.02
6	ALFKI	10702	1997-10-13 00:00:00.000	353.94	10702 Oct 13 1997 12:00AM 353.94	10692 Oct 3 1997 12:00AM 939.02
7	ANATR	10926	1998-03-04 00:00:00.000	554.3200015258789	10926 Mar 4 1998 12:00AM 554.32	10926 Mar 4 1998 12:00AM 554.32
8	ANATR	10625	1997-08-08 00:00:00.000	523.65	10625 Aug 8 1997 12:00AM 523.65	10625 Aug 8 1997 12:00AM 523.65
9	ANATR	10308	1996-09-18 00:00:00.000	90.40999923706055	10308 Sep 18 1996 12:00AM 90.41	10308 Sep 18 1996 12:00AM 90.41
10	ANATR	10759	1997-11-28 00:00:00.000	331.99	10759 Nov 28 1997 12:00AM 331.99	10759 Nov 28 1997 12:00AM 331.99
11	ANTON	10856	1998-01-28 00:00:00.000	718.43	10856 Jan 28 1998 12:00AM 718.43	10856 Jan 28 1998 12:00AM 718.43
12	ANTON	10507	1997-04-15 00:00:00.000	796.5125	10507 Apr 15 1997 12:00AM 796.513	10507 Apr 15 1997 12:00AM 796.513
13	ANTON	10535	1997-05-13 00:00:00.000	1956.4899908447267	10535 May 13 1997 12:00AM 1956.49	10535 May 13 1997 12:00AM 1956.49
14	ANTON	10573	1997-06-19 00:00:00.000	2166.84	10573 Jun 19 1997 12:00AM 2166.84	10573 Jun 19 1997 12:00AM 2166.84
15	ANTON	10677	1997-09-22 00:00:00.000	817.3950512695312	10682 Sep 25 1997 12:00AM 411.63	10677 Sep 22 1997 12:00AM 817.395
16	ANTON	10682	1997-09-25 00:00:00.000	411.63	10682 Sep 25 1997 12:00AM 411.63	10677 Sep 22 1997 12:00AM 817.395
17	ANTON	10365	1996-11-27 00:00:00.000	425.200001220703125	10365 Nov 27 1996 12:00AM 425.2	10365 Nov 27 1996 12:00AM 425.2
18	AROUT	10453	1997-02-21 00:00:00.000	433.0599969482422	10864 Feb 2 1998 12:00AM 285.04	10453 Feb 21 1997 12:00AM 433.06
19	AROUT	10864	1998-02-02 00:00:00.000	285.04	10864 Feb 2 1998 12:00AM 285.04	10453 Feb 21 1997 12:00AM 433.06
20	AROUT	10953	1998-03-16 00:00:00.000	4464.97	10920 Mar 3 1998 12:00AM 419.61	10953 Mar 16 1998 12:00AM 4464.97
21	AROUT	10920	1998-03-03 00:00:00.000	419.61	10920 Mar 3 1998 12:00AM 419.61	10953 Mar 16 1998 12:00AM 4464.97
22	AROUT	11016	1998-04-10 00:00:00.000	525.3	11016 Apr 10 1998 12:00AM 525.3	11016 Apr 10 1998 12:00AM 525.3
23	AROUT	10558	1997-06-04 00:00:00.000	2215.8700244140623	10558 Jun 4 1997 12:00AM 2215.87	10558 Jun 4 1997 12:00AM 2215.87
24	AROUT	10707	1997-10-16 00:00:00.000	1662.74	10707 Oct 16 1997 12:00AM 1662.74	10707 Oct 16 1997 12:00AM 1662.74
25	AROUT	10355	1996-11-15 00:00:00.000	521.95	10741 Nov 14 1997 12:00AM 238.96	10355 Nov 15 1996 12:00AM 521.95
26	AROUT	10743	1997-11-17 00:00:00.000	342.91998168945315	10741 Nov 14 1997 12:00AM 238.96	10355 Nov 15 1996 12:00AM 521.95
27	AROUT	10741	1997-11-14 00:00:00.000	238.96	10741 Nov 14 1997 12:00AM 238.96	10355 Nov 15 1996 12:00AM 521.95
28	AROUT	10768	1997-12-08 00:00:00.000	1623.32	10793 Dec 24 1997 12:00AM 195.62	10768 Dec 8 1997 12:00AM 1623.32

W celu skrócenia zapytania dwukrotnie wykorzystano funkcję first_value() zmieniając kolejność segregowanie

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 t as
    (select PH.id, PH.productid, PH.date, PH.value
     from product_history PH)
select t.*, sum(t.value)
       over(partition by t.productid, year(t.date), month(t.date) order by t.date) as ascVal
from t
```

	id	productid	date	value	ascVal
1	1	1	1940-01-02	109.60	109.60
2	78	1	1940-01-03	262.90	372.50
3	155	1	1940-01-04	278.60	651.10
4	232	1	1940-01-05	163.70	814.80
5	309	1	1940-01-06	221.90	1036.70
6	386	1	1940-01-07	117.00	1153.70
7	463	1	1940-01-08	258.30	1412.00
8	540	1	1940-01-09	129.80	1541.80
9	617	1	1940-01-10	107.40	1649.20
10	694	1	1940-01-11	225.60	1874.80
11	771	1	1940-01-12	191.90	2066.70
12	848	1	1940-01-13	127.10	2193.80
13	925	1	1940-01-14	111.90	2305.70
14	1002	1	1940-01-15	263.30	2569.00
15	1079	1	1940-01-16	214.70	2783.70
16	1156	1	1940-01-17	236.90	3020.60
17	1233	1	1940-01-18	173.00	3193.60
18	1310	1	1940-01-19	201.80	3395.40
19	1387	1	1940-01-20	211.30	3606.70
20	1464	1	1940-01-21	122.10	3728.80
21	1541	1	1940-01-22	243.80	3972.60
22	1618	1	1940-01-23	115.10	4087.70
23	1695	1	1940-01-24	130.60	4218.30
24	1772	1	1940-01-25	244.60	4462.90
25	1849	1	1940-01-26	176.90	4639.80
26	1926	1	1940-01-27	233.90	4873.70
27	2003	1	1940-01-28	228.70	5102.40
28	2080	1	1940-01-29	278.90	5381.30

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)

```
select PH.id, PH.productid, PH.date, PH.value,
       (select sum(PH2.value)
        from product_history PH2
        where PH2.date <= PH.date
          and PH2.productid = PH.productid
          and year(PH2.date) = year(PH.date)
          and month(PH2.date) = month(PH.date))
from product_history PH
order by PH.productid, PH.date
```

Zapytanie bez funkcji okna wykonuje się w znacznie dłuższym czasie. Różnicę widać w koszcie wykonania zapytań, 8112 bez wykorzystania funkcji okna oraz 26 wykorzystując funkcję okna. Pokazuje to przydatność tego mechanizmu.

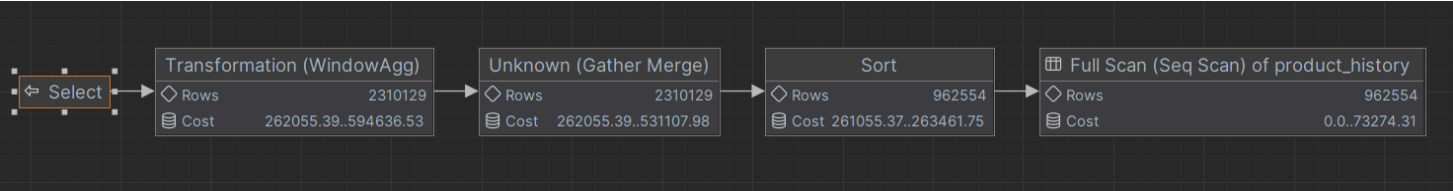
	id	productid	date	value	<anonymous>
1	1	1	1940-01-02	109.60	109.60
2	78	1	1940-01-03	262.90	372.50
3	155	1	1940-01-04	278.60	651.10
4	232	1	1940-01-05	163.70	814.80
5	309	1	1940-01-06	221.90	1036.70
6	386	1	1940-01-07	117.00	1153.70
7	463	1	1940-01-08	258.30	1412.00
8	540	1	1940-01-09	129.80	1541.80
9	617	1	1940-01-10	107.40	1649.20
10	694	1	1940-01-11	225.60	1874.80
11	771	1	1940-01-12	191.90	2066.70
12	848	1	1940-01-13	127.10	2193.80
13	925	1	1940-01-14	111.90	2305.70
14	1002	1	1940-01-15	263.30	2569.00
15	1079	1	1940-01-16	214.70	2783.70
16	1156	1	1940-01-17	236.90	3020.60
17	1233	1	1940-01-18	173.00	3193.60
18	1310	1	1940-01-19	201.80	3395.40
19	1387	1	1940-01-20	211.30	3606.70
20	1464	1	1940-01-21	122.10	3728.80
21	1541	1	1940-01-22	243.80	3972.60
22	1618	1	1940-01-23	115.10	4087.70
23	1695	1	1940-01-24	130.60	4218.30
24	1772	1	1940-01-25	244.60	4462.90
25	1849	1	1940-01-26	176.90	4639.80
26	1926	1	1940-01-27	233.90	4873.70
27	2003	1	1940-01-28	228.70	5102.40

Porównując plany zapytań można dojść do dwóch wniosków.

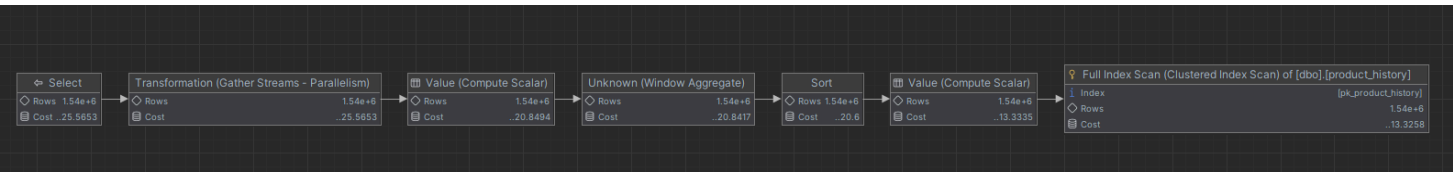
- Plany nie różnią się znacznie między różnymi serwerami.
- Zapytania z funkcją okna mają znacznie prostszy plan, w porównaniu do zapytań bez funkcji okna.

Czas wykonania zapytań był najkrótszy na serwerze MS SQL. Dodatkowo

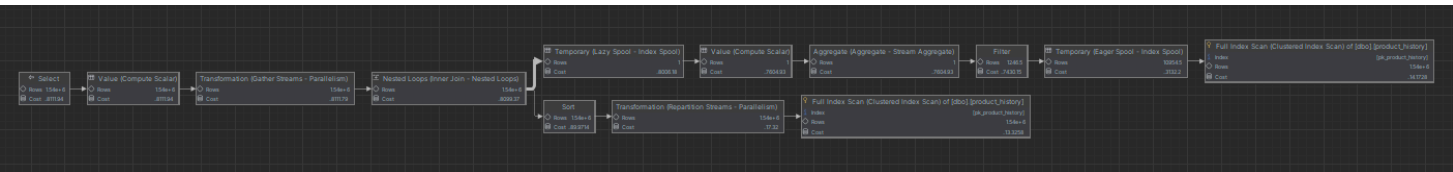
Plan zapytania z funkcją okna na serwerze PostgreSQL



Plan zapytania z funkcją okna na serwerze MS SQL



Plan zapytania bez funkcji okna na serwerze MS SQL



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.

```

select
    c.contactname, o.ORDERID, o.ORDERDATE,
    (SUM(od.unitprice * od.quantity * (1 - od.discount)) + o.freight) as OrderValue,
    ntile(4) over (
        partition by YEAR(o.ORDERDATE), MONTH(o.ORDERDATE)
        order by (sum(od.unitprice * od.quantity * (1 - od.discount)) + o.freight)) as Quartile
from
    orders o
    join
    customers c on o.CUSTOMERID = c.CUSTOMERID
    join
    [order details] od on o.ORDERID = od.ORDERID
group by
    o.ORDERID, o.ORDERDATE, o.FREIGHT, c.CONTACTNAME

```

Zapytanie to zwraca nazwę klienta, numer zamówienia, datę zamówienia, wartość zamówienia oraz kwartył, w którym znajduje się wartość zamówienia dla danego miesiąca

	contactname	ORDERID	ORDERDATE	OrderValue	Quartile
1	Francisco Chang	10259	1996-07-18 00:00:00.000	104.04999923706055	1
2	Pirkko Koskitalo	10266	1996-07-26 00:00:00.000	372.28999755859377	1
3	Bernardo Batista	10261	1996-07-19 00:00:00.000	451.05	1
4	Paul Henriot	10248	1996-07-04 00:00:00.000	472.38	1
5	Paula Parente	10256	1996-07-15 00:00:00.000	531.7700030517578	1
6	Yang Wang	10254	1996-07-11 00:00:00.000	579.6000332641602	1
7	Paula Wilson	10262	1996-07-22 00:00:00.000	632.2899961853027	2
8	Karl Jablonski	10269	1996-07-31 00:00:00.000	646.7600122070312	2
9	Mary Saveley	10251	1996-07-08 00:00:00.000	695.4000051879883	2
10	Maria Larsson	10264	1996-07-24 00:00:00.000	699.295	2
11	Manuel Pereira	10268	1996-07-30 00:00:00.000	1167.4899969482422	2
12	Carlos Hernández	10257	1996-07-16 00:00:00.000	1201.810001525879	2
13	Frédérique Citeaux	10265	1996-07-25 00:00:00.000	1231.28	3
14	Mario Pontes	10253	1996-07-10 00:00:00.000	1502.9699877929688	3
15	Henriette Pfalzheim	10260	1996-07-19 00:00:00.000	1559.7399938964843	3
16	Mario Pontes	10250	1996-07-08 00:00:00.000	1618.4300366210937	3
17	Roland Mendel	10258	1996-07-17 00:00:00.000	1755.3900048828125	3
18	Karin Josephs	10249	1996-07-05 00:00:00.000	1875.0099938964843	4
19	Roland Mendel	10263	1996-07-23 00:00:00.000	2019.8600030517578	4
20	Michael Holz	10255	1996-07-12 00:00:00.000	2638.83	4
21	Pascale Cartrain	10252	1996-07-09 00:00:00.000	3649.19990234375	4
22	Peter Franken	10267	1996-07-29 00:00:00.000	3745.1800061035156	4
23	Art Braunschweiger	10271	1996-08-01 00:00:00.000	52.54	1
24	Maurizio Moroni	10288	1996-08-23 00:00:00.000	87.5499984741211	1
25	Alejandra Camino	10281	1996-08-14 00:00:00.000	89.43999904632568	1
26	Alejandra Camino	10282	1996-08-15 00:00:00.000	168.08999771118164	1
27	Giovanni Rovelli	10275	1996-08-07 00:00:00.000	318.77000396728516	1

```

with t as (
    select
        o.CustomerID,
        c.ContactName,
        o.OrderID,
        (sum(od.UnitPrice * od.Quantity * (1 - od.Discount)) + o.Freight) as OrderValue,
        row_number() over (partition by o.CustomerID order by o.OrderDate desc ) as OrderRank
    from
        orders o
    join
        [order details] od on o.OrderID = od.OrderID
    join Customers c on o.CustomerID = c.CustomerID
    group by
        o.CustomerID, c.ContactName, o.OrderID, o.Freight, o.OrderDate
)
select t.CustomerID, t.ContactName, t.OrderID, t.OrderValue
from t
where OrderRank = 1;

```

Wykorzystujemy funkcję okna ROW_NUMBER() do przypisania kolejności zamówień dla każdego klienta na podstawie daty zamówienia, sortując malejąco po dacie zamówienia, a następnie wybieramy tylko te zamówienia, które mają wartość 1 w kolumnie OrderRank, co oznacza ostatnie zamówienie dla każdego klienta.

	☐ CustomerID	÷ ☐ ContactName	÷ ☐ OrderID	÷ ☐ OrderValue
1	ALFKI	Maria Anders	11011	934.71
2	ANATR	Ana Trujillo	10926	554.3200015258789
3	ANTON	Antonio Moreno	10856	718.43
4	AROUT	Thomas Hardy	11016	525.3
5	BERGS	Christina Berglund	10924	1987.219951171875
6	BLAUS	Hanna Moos	11058	889.14
7	BLONP	Frédérique Citeaux	10826	737.09
8	BOLID	Martin Sommer	10970	240.16
9	BONAP	Laurence Lebihan	11076	831.03
10	BOTTM	Elizabeth Lincoln	11048	549.12
11	BSBEV	Victoria Ashworth	11023	1623.83
12	CACTU	Patricio Simpson	11054	305.33
13	CENTC	Francisco Chang	10259	104.04999923706055
14	CHOPS	Yang Wang	11041	1821.22
15	COMMI	Pedro Afonso	11042	435.74
16	CONSH	Elizabeth Brown	10848	969.74
17	DRACD	Sven Ottlieb	11067	94.8299984741211
18	DUMON	Janine Labrune	10890	892.8600061035156
19	EASTC	Ann Devon	11056	4018.96
20	ERNSH	Roland Mendel	11072	5476.64
21	FAMIA	Aria Cruz	10725	298.6300030517578
22	FOLIG	Martine Rancé	10789	3787.6
23	FOLKO	Maria Larsson	11050	869.41
24	FRANK	Peter Franken	11012	3068.2499267578123
25	FRANR	Carine Schmitt	10971	1854.88005859375
26	FRANS	Paolo Accorti	11060	276.98

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