

SQL - Funkcje okna (Window functions)

Lab 2

Imiona i nazwiska:

Paweł Surdyka

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 15/16/17
- SQLite
- Narzędzia do komunikacji z bazą danych
 - SSMS - Microsoft SQL Management Studio
 - DDataGrip lub DBeaver
- Przykładowa baza Northwind/Northwind3
 - 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/>
- W razie potrzeby - opis ikonek używanych w graficznej prezentacji planu zapytania w SSMS jest 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 Northwind3
 - na lokalnym serwerze MS SQL
 - na lokalnym serwerze PostgreSQL
 - z lokalną bazą SQLite

Można też skorzystać z innych narzędzi klienckich (wg własnego uznania)

Oryginalna baza Northwind jest bardzo mała. Warto zaobserwować działanie na nieco większym zbiorze danych.

Baza Northwind3 zawiera dodatkową tabelę product_history

- 2,2 mln wierszy

Bazę Northwind3 można pobrać z moodle (zakładka - Backupy baz danych)

Można też wygenerować tabelę product_history przy pomocy skryptu

Skrypt dla SQL Sriver

Stwórz tabelę o następującej strukturze:

```
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

Wyniki:

```
select count(*) from product_history
```

MS SQL

	(No column name)
1	2310000

PostgreSQL

	count
1	2310000

SQLite

	"count(*)"
1	2310000

Zadanie 1

Baza: Northwind, tabela product_history

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

W przypadku długiego czasu wykonania ogranicz zbiór wynikowy do kilkuset/kilku tysięcy wierszy

pomocna może być konstrukcja **with**

```
with t as (  
  
....  
)  
select * from t  
where id between ....
```

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)

Wyniki:

```
-- podzapytanie  
SELECT *  
FROM (  
    SELECT  
        ph.id,  
        ph.productid,  
        ph.productname,  
        ph.categoryid,  
        ph.unitprice,  
        (  
            SELECT AVG(ph2.unitprice)  
            FROM product_history ph2  
            WHERE ph2.categoryid = ph.categoryid  
        ) AS avg_price_category  
    FROM product_history ph  
) AS subquery  
WHERE unitprice > avg_price_category;  
  
-- join  
SELECT  
    ph.id,  
    ph.productid,  
    ph.productname,  
    ph.categoryid,  
    ph.unitprice,
```

```

    avg_price_category.avg AS avg_price_category
FROM product_history ph
JOIN (
    SELECT
        categoryid,
        AVG(unitprice) AS avg
    FROM product_history
    GROUP BY categoryid
) AS avg_price_category
    ON ph.categoryid = avg_price_category.categoryid
WHERE ph.unitprice > avg_price_category.avg;

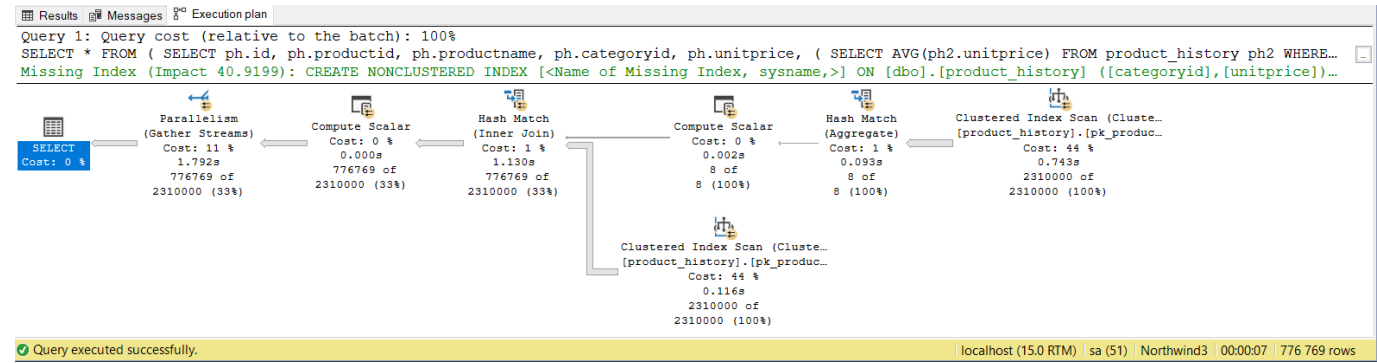
-- funkcja okna
SELECT *
FROM (
    SELECT
        ph.id,
        ph.productid,
        ph.productname,
        ph.categoryid,
        ph.unitprice,
        AVG(unitprice) OVER (PARTITION BY categoryid) AS avg_price_category
    FROM product_history ph
) AS subquery
WHERE unitprice > avg_price_category;

```

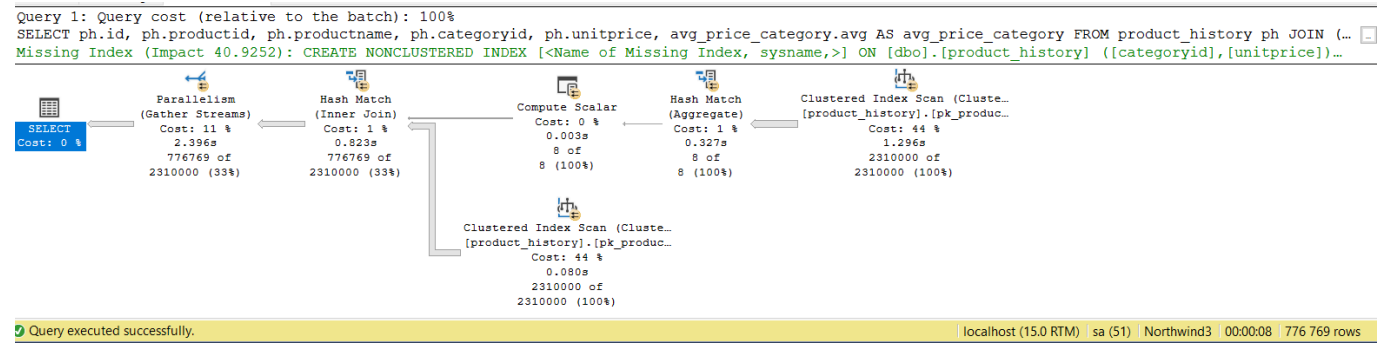
	id	productid	productname	categoryid	unitprice	avg_price_category
1	147724	38	Côte de Blaye	1	49.52	29.059460
2	147771	8	Northwoods Cranberry Sauce	2	27.40	21.573672
3	147772	9	Mishi Kobe Niku	6	52.20	37.102698
4	147773	10	Ikura	8	23.49	20.379300
5	147775	12	Queso Manchego La Pastora	4	26.53	24.417846
6	147781	18	Carnarvon Tigers	8	37.19	20.379300
7	147783	20	Sir Rodney's Marmalade	3	45.24	22.626289
8	147789	26	Gumbär Gummibärchen	3	23.59	22.626289
9	147790	27	Schoggi Schokolade	3	29.10	22.626289
10	147791	28	Rössle Sauerkraut	7	29.84	26.244567
11	147792	29	Thüringer Rostbratwurst	6	63.86	37.102698
12	147793	30	Nord-Ost Matjeshering	8	21.26	20.379300
13	147800	37	Gravad lax	8	21.31	20.379300
14	147801	38	Côte de Blaye	1	124.64	29.059460
15	147806	43	Ipoh Coffee	1	30.01	29.059460
16	147814	51	Manjimup Dried Apples	7	33.06	26.244567

MS SQL

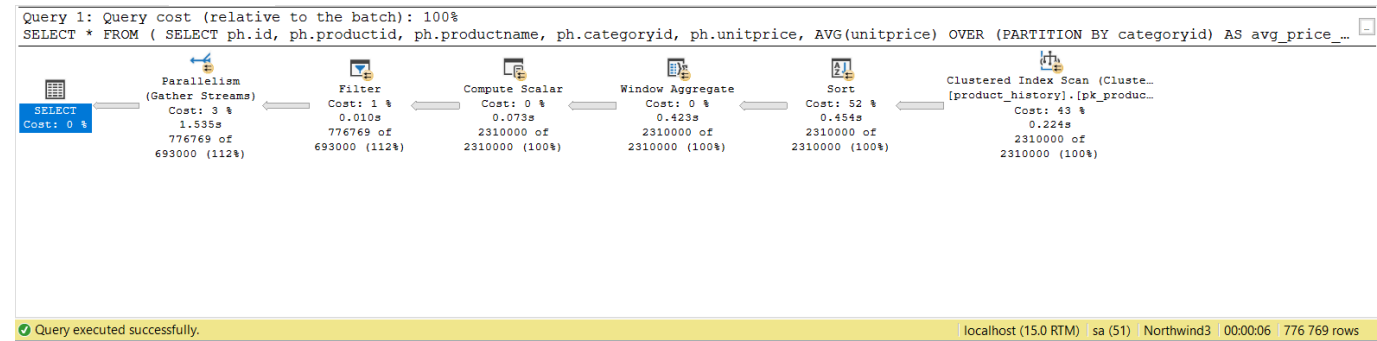
podzapytanie



Join



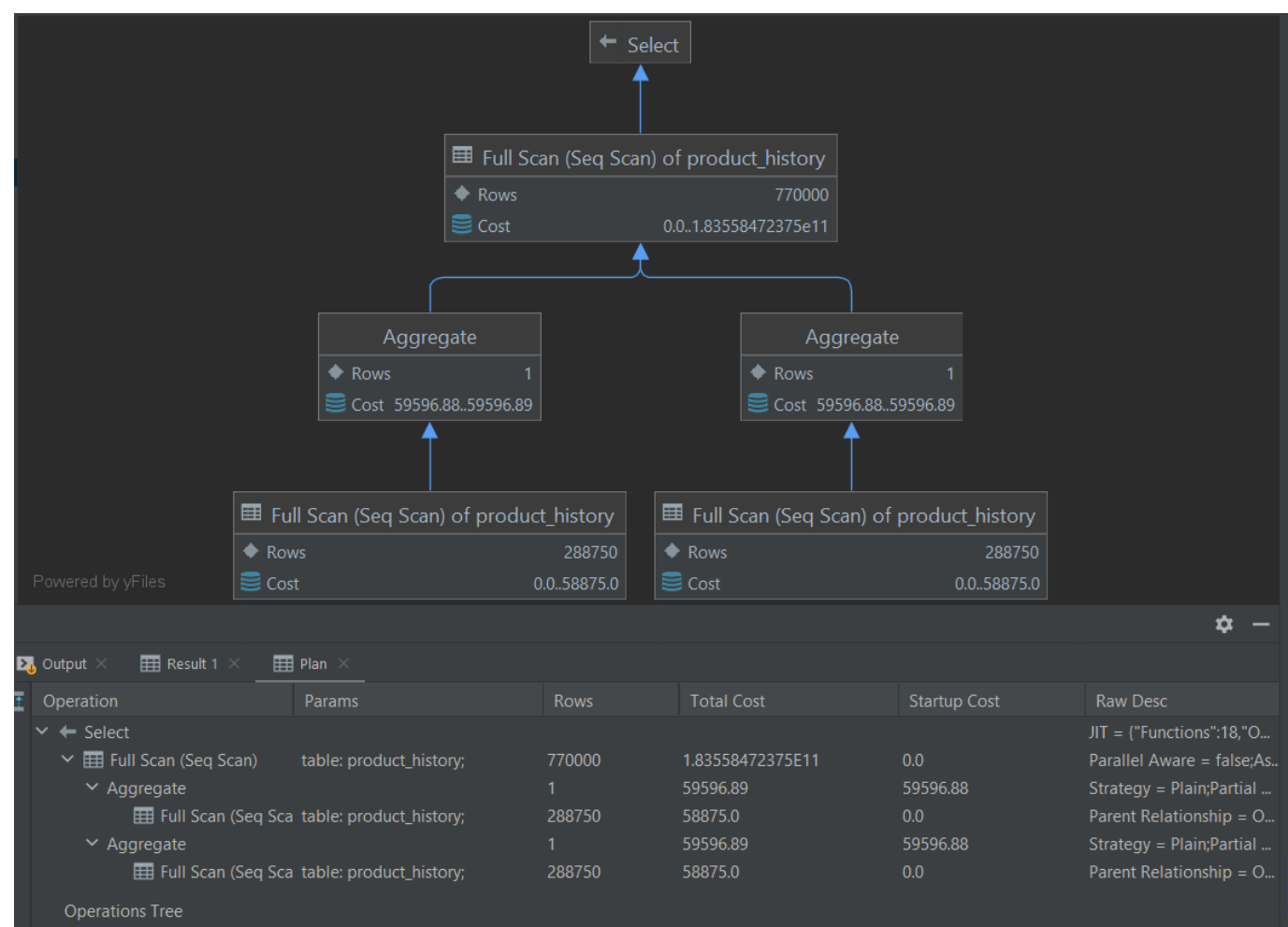
funkcja okna



PostgreSQL

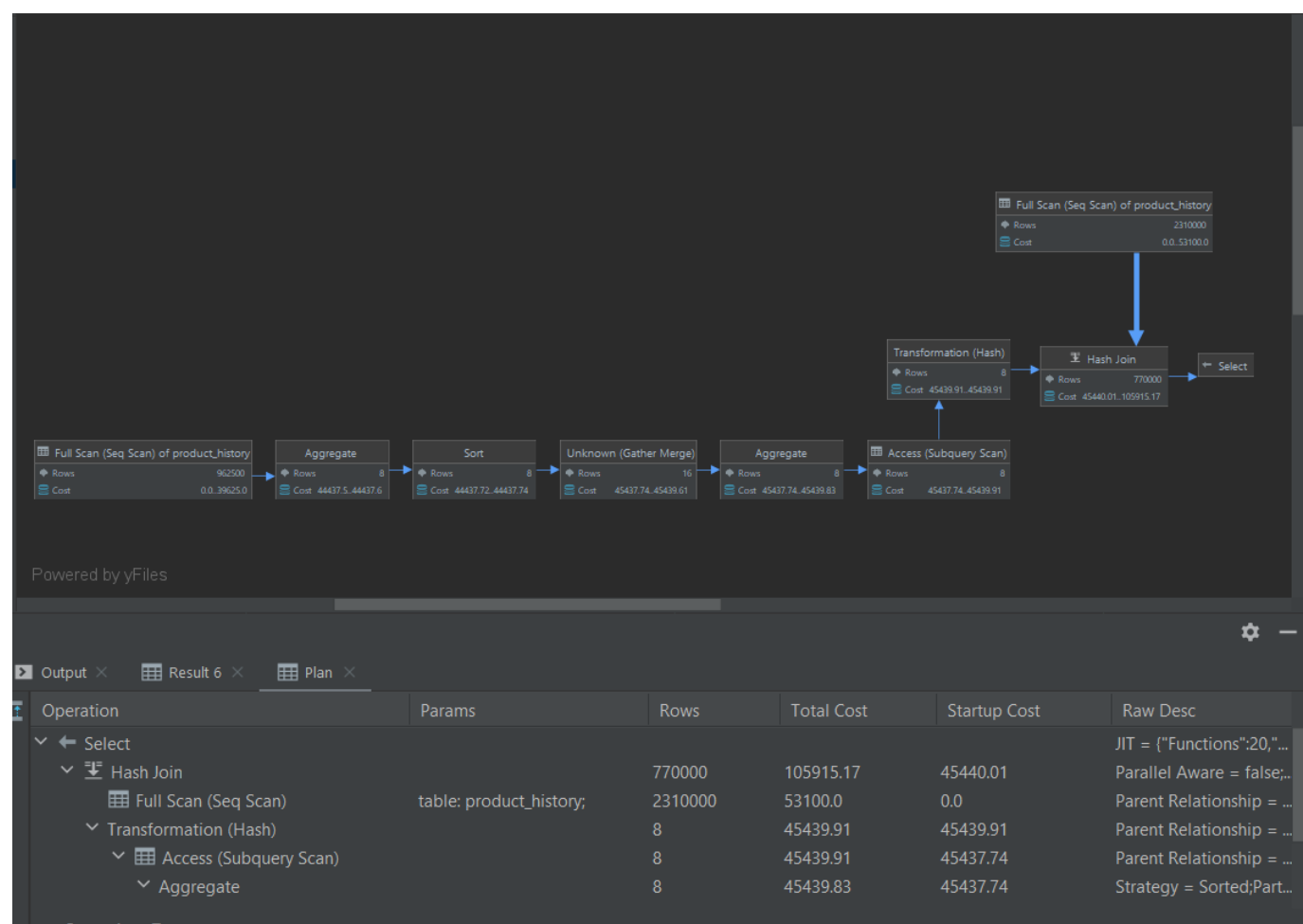
podzapytanie

execution: 4 m 9 s 395 ms, fetching: 685 ms



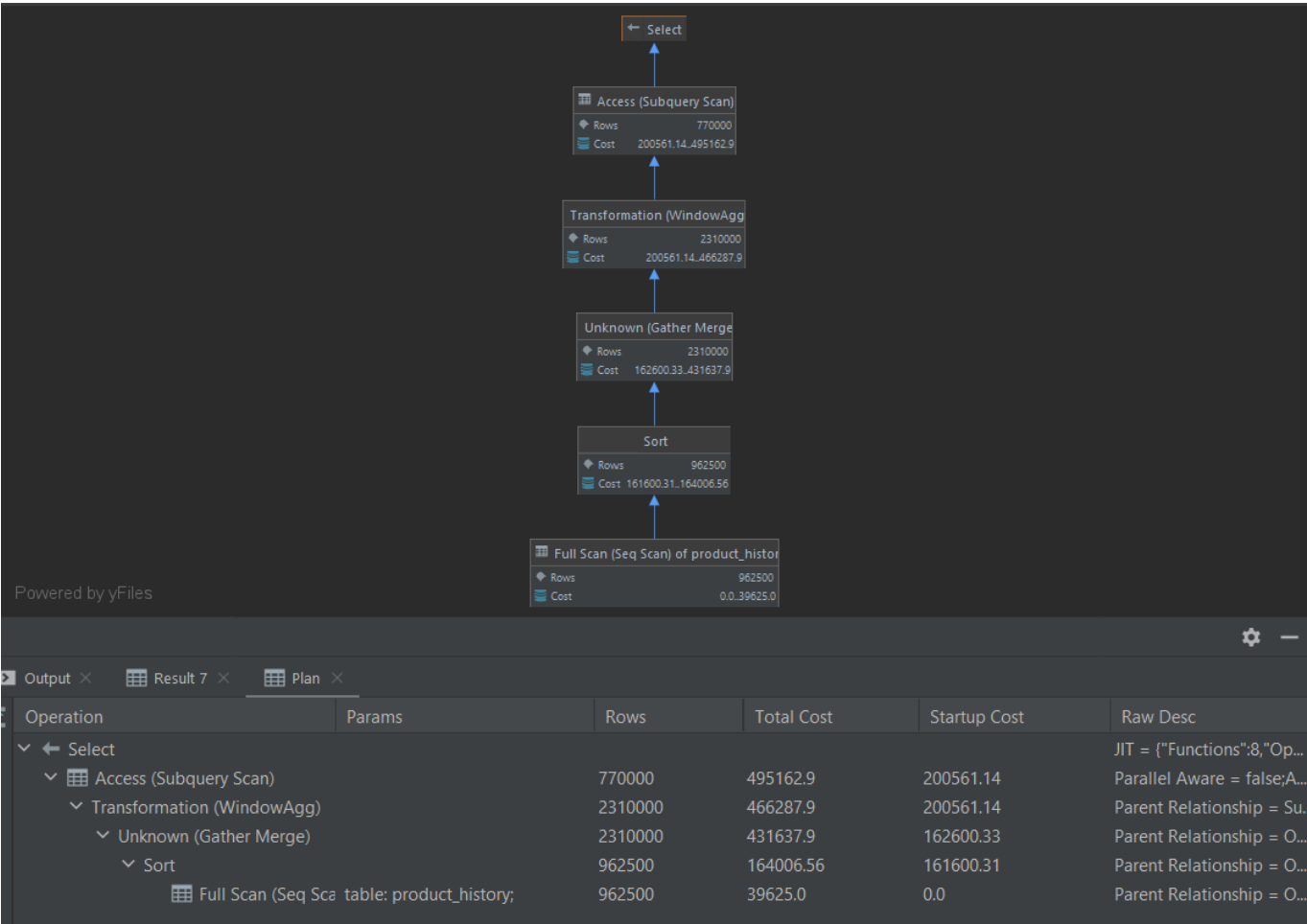
Join

execution: 504 ms, fetching: 61 ms



funkcja okna

execution: 5 s 576 ms, fetching: 107 ms



SQLite

podzapytanie

execution: 2 s 667 ms, fetching: 7 m 8 s 177 ms

Wersja z podzapytaniem skorelowanym (subquery) została poprawnie wykonana w SQLite, jednak EXPLAIN QUERY PLAN nie był w stanie wygenerować pełnego planu wykonania.

```
[2025-04-06 20:25:23] Database returned plan in unsupported format: unexpected SUBQUERY != ph2
```

plan w wersji "raw":

	id	parent	notused	detail
1	2	0	0	SCAN ph
2	6	0	0	CORRELATED SCALAR SUBQUERY 1
3	12	6	0	SEARCH ph2 USING INDEX ix_product_history_categoryid (categor...
4	33	0	0	CORRELATED SCALAR SUBQUERY 1
5	39	33	0	SEARCH ph2 USING INDEX ix_product_history_categoryid (categor...

Join

execution: 730 ms, fetching: 210 ms

Jak wyżej, zapytanie zostało wykonane poprawnie ale nie dało się wygenerować planu wykonania.

```
[2025-04-06 20:34:09] Database returned plan in unsupported format: unexpected SUBQUERY != avg_price_category
```

plan w wersji "raw":

	id	parent	notused	detail
1	2	0	0	CO-ROUTINE avg_price_category
2	9	2	0	SCAN product_history USING INDEX ix_product_history_categoryid
3	42	0	0	SCAN avg_price_category
4	45	0	0	SEARCH ph USING INDEX ix_product_history_categoryid (category...

funkcja okna

execution: 919 ms, fetching: 66 ms

```
[2025-04-06 20:52:48] Database returned plan in unsupported format: unexpected SUBQUERY != product_history
```

plan w wersji "raw":

	id	parent	notused	detail
1	2	0	0	CO-ROUTINE (subquery-1)
2	5	2	0	CO-ROUTINE (subquery-3)
3	9	5	0	SCAN product_history USING INDEX ix_product_history_categoryid
4	28	2	0	SCAN (subquery-3)
5	89	0	0	SCAN (subquery-1)

SZBD	Podzapytanie	JOIN	Funkcja okna
SQL Server	7s	8s	6s
PostgreSQL	4 m 9 s	504 ms	5 s 576 ms
SQLite	7 m 11 s	930 ms	985 ms

Wnioski:

- Niestety w środowisku DataGrip wykonywanie execution plan powodowało pewne kłopoty dla SQLite.
- Czasy wykonania dla SQL Server były podobne dla wszystkich metod i wynosiły około 7 sekund.
- Czasy wykonania dla PostgreSQL i SQLite były podobne dla wersji Join i funkcji okna ale dla podzapytań obydwa SZBD potrzebowały kilku minut.

Zadanie 2

Baza: Northwind, tabela product_history

Lekka modyfikacja poprzedniego zadania

Napisz polecenie, które zwraca: id pozycji, id produktu, datę, nazwę produktu, id_kategorii, 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 produktu w roku którego dotyczy dana pozycja (suma dla pola value)

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

Podobnie jak poprzednio, w przypadku długiego czasu wykonania ogranicz zbiór wynikowy do kilkuset/kilku tysięcy wierszy

Porównaj czasy oraz plany wykonania zapytań.

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

Wyniki:

```
-- podzapytanie

SELECT
    phist.id,
    phist.productid,
    phist.date,
    phist.productname,
    phist.categoryid,
    phist.unitprice,

    -- Średnia cena produktów w kategorii
    (
        SELECT AVG(phist2.unitprice)
        FROM product_history phist2
        WHERE phist2.categoryid = phist.categoryid
        AND phist2.id < 1000
    ) AS avg_category_price,

    -- łączna wartość sprzedaży produktów w danej kategorii
    (
        SELECT SUM(phist3.unitprice)
        FROM product_history phist3
        WHERE phist3.categoryid = phist.categoryid
        AND phist3.id < 1000
    ) AS total_category_sales,

    -- Średnia cena danego produktu w roku, którego dotyczy dana pozycja
    (
        SELECT AVG(phist4.unitprice)
        FROM product_history phist4
        WHERE phist4.productid = phist.productid
        AND YEAR(phist4.date) = YEAR(phist.date)
        AND phist4.id < 1000
    ) AS avg_product_price_year,

    -- łączna wartość sprzedaży produktu w roku, którego dotyczy dana pozycja
    (
        SELECT SUM(phist5.unitprice)
        FROM product_history phist5
        WHERE phist5.productid = phist.productid
```

```
        AND YEAR(phist5.date) = YEAR(phist.date)
        AND phist5.id < 1000
    ) AS total_product_sales_year

FROM product_history as phist WHERE id < 1000

-- JOIN

SELECT
    ph.id,
    ph.productid,
    ph.date,
    ph.productname,
    ph.categoryid,
    ph.unitprice,

    AVG(ph2.unitprice) AS avg_category_price,
    SUM(ph2.unitprice) AS total_category_sales,
    AVG(ph3.unitprice) AS avg_product_price_year,
    SUM(ph3.unitprice) AS total_product_sales_year

FROM product_history ph
JOIN product_history ph2 ON ph2.categoryid = ph.categoryid
    AND ph2.id < 1000
JOIN product_history ph3 ON ph3.productid = ph.productid
    AND ph3.id < 1000
    AND YEAR(ph3.date) = YEAR(ph.date)
WHERE ph.id < 1000
GROUP BY
    ph.id,
    ph.productid,
    ph.date,
    ph.productname,
    ph.categoryid,
    ph.unitprice;

-- funkcja okna

SELECT
    ph.id,
    ph.productid,
    ph.date,
    ph.productname,
    ph.categoryid,
    ph.unitprice,

    AVG(ph.unitprice) OVER (
        PARTITION BY ph.categoryid
    ) AS srednia_cena_w_kategorii,
    SUM(ph.unitprice) OVER (
        PARTITION BY ph.categoryid
    ) AS laczna_wartosc_kategorii,
    AVG(ph.unitprice) OVER (
```

```

PARTITION BY ph.productid, YEAR(ph.date)
) AS srednia_cena_produkту_w_roku,
SUM(ph.unitprice) OVER (
PARTITION BY ph.productid, YEAR(ph.date)
) AS laczna_wartosc_produkту_w_roku

FROM product_history ph
WHERE ph.id < 1000

```

	id	productid	date	productname	categoryid	unitprice	srednia_cena_w_kategorii	laczna_wartosc_kategorii	srednia_cena_produkту_w_roku	laczna_wartosc_produkту_w_roku
1	1	1	1940-01-02	Chai	1	15.85	31.551032	4890.41	20.172307	262.24
2	78	1	1940-01-03	Chai	1	25.67	31.551032	4890.41	20.172307	262.24
3	155	1	1940-01-04	Chai	1	27.42	31.551032	4890.41	20.172307	262.24
4	232	1	1940-01-05	Chai	1	14.03	31.551032	4890.41	20.172307	262.24
5	309	1	1940-01-06	Chai	1	26.58	31.551032	4890.41	20.172307	262.24
6	386	1	1940-01-07	Chai	1	20.25	31.551032	4890.41	20.172307	262.24
7	463	1	1940-01-08	Chai	1	21.23	31.551032	4890.41	20.172307	262.24
8	617	1	1940-01-10	Chai	1	23.31	31.551032	4890.41	20.172307	262.24
9	540	1	1940-01-09	Chai	1	18.74	31.551032	4890.41	20.172307	262.24
10	694	1	1940-01-11	Chai	1	21.02	31.551032	4890.41	20.172307	262.24
11	771	1	1940-01-12	Chai	1	17.87	31.551032	4890.41	20.172307	262.24
12	848	1	1940-01-13	Chai	1	12.55	31.551032	4890.41	20.172307	262.24
13	925	1	1940-01-14	Chai	1	17.72	31.551032	4890.41	20.172307	262.24
14	926	2	1940-01-14	Chang	1	18.15	31.551032	4890.41	20.736923	269.58
15	849	2	1940-01-13	Chang	1	12.70	31.551032	4890.41	20.736923	269.58
16	772	2	1940-01-12	Chang	1	18.30	31.551032	4890.41	20.736923	269.58

MS SQL

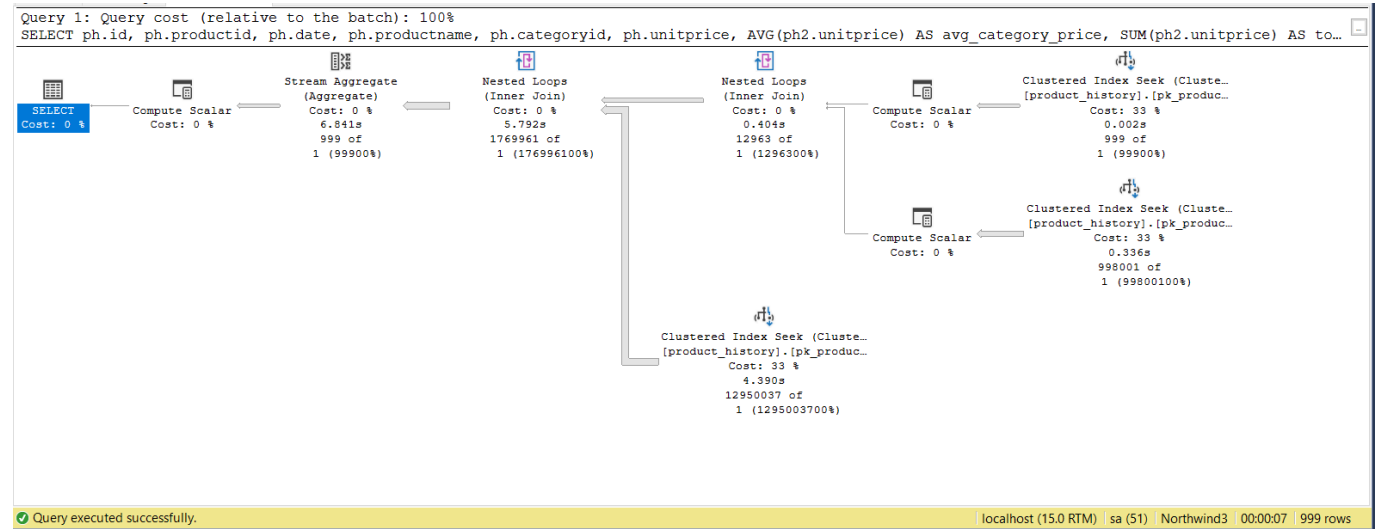
podzapytanie

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

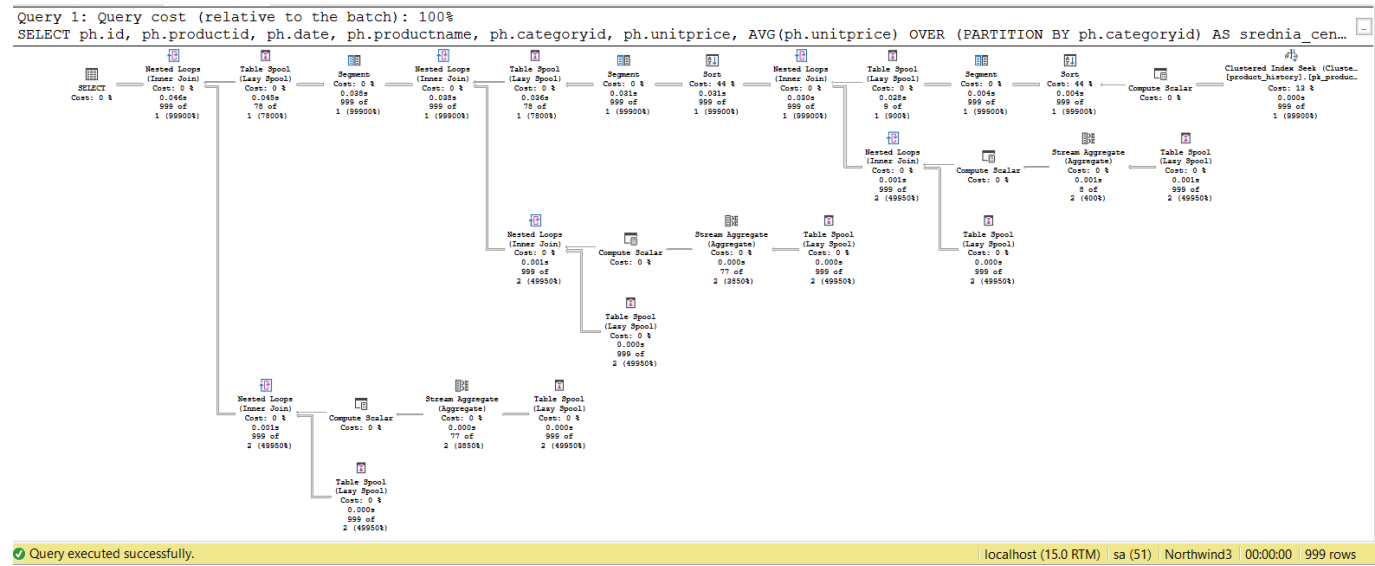
SELECT phist.id, phist.productid, phist.date, phist.productname, phist.categoryid, phist.unitprice, -- Średnia cena produktów w kategorii (SELECT A...

Query executed successfully. | localhost (15.0 RTM) | sa (51) | Northwind3 | 00:00:15 | 9 999 rows

Join



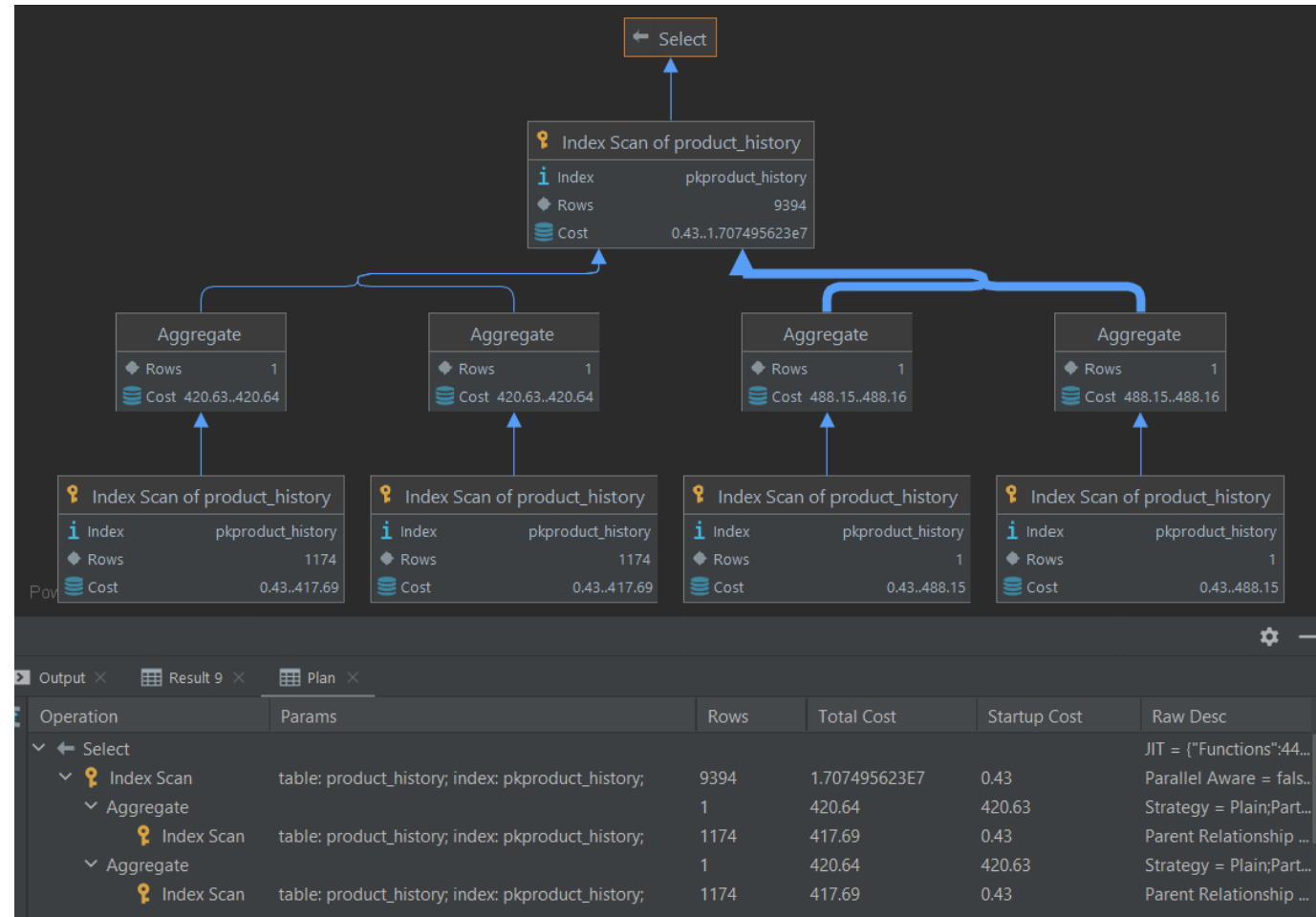
funkcja okna



PostgreSQL

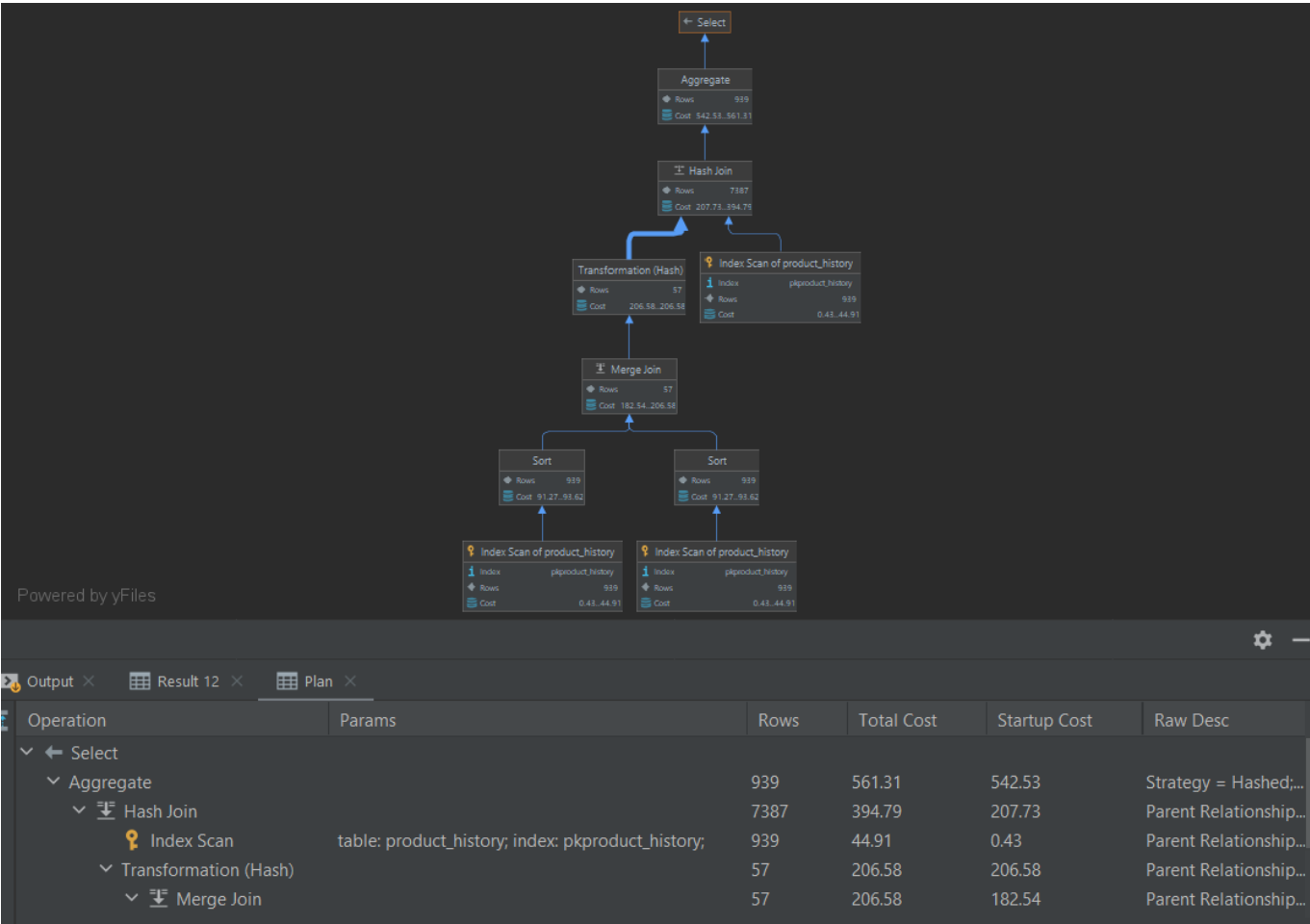
podzapytanie

execution: 3 s 757 ms, fetching: 327 ms



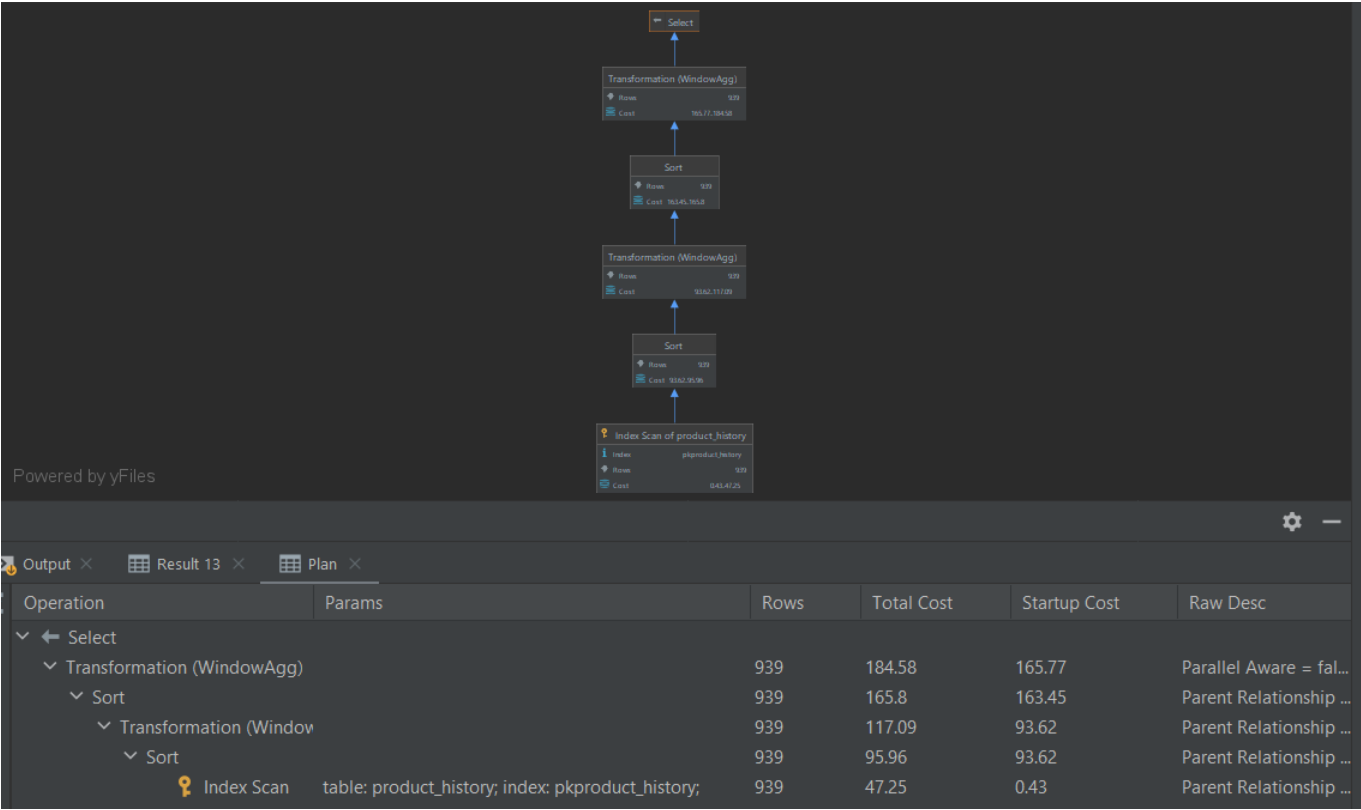
Join

execution: 1 s 9 ms, fetching: 99 ms



funkcja okna

execution: 71 ms, fetching: 235 ms



SQLite

podzapytanie

execution: 166 ms, fetching: 835 ms

id	parent	notused	detail
2	0	0	SEARCH phist USING INTEGER PRIMARY KEY (...)
14	0	0	CORRELATED SCALAR SUBQUERY 1
20	14	0	SEARCH phist2 USING INDEX ix_product_his...
37	0	0	CORRELATED SCALAR SUBQUERY 2
43	37	0	SEARCH phist3 USING INDEX ix_product_his...
60	0	0	CORRELATED SCALAR SUBQUERY 3
66	60	0	SEARCH phist4 USING INDEX ix_product_his...
88	0	0	CORRELATED SCALAR SUBQUERY 4
94	88	0	SEARCH phist5 USING INDEX ix_product_his...

Join

execution: 16 ms, fetching: 459 ms

id	parent	notused	detail
10	0	0	SEARCH ph USING INTEGER PRIMARY KEY (row...
15	0	0	SEARCH ph3 USING INDEX ix_product_histor...
28	0	0	SEARCH ph2 USING INDEX ix_product_histor...

funkcja okna

execution: 200 ms, fetching: 80 ms

id	parent	notused	detail
4	0	0	CO-ROUTINE (subquery-2)
8	4	0	CO-ROUTINE (subquery-3)
12	8	0	SCAN ph USING INDEX ix_product_history_productid
38	8	0	USE TEMP B-TREE FOR RIGHT PART OF ORDER BY
66	4	0	SCAN (subquery-3)
142	4	0	USE TEMP B-TREE FOR ORDER BY
166	0	0	SCAN (subquery-2)

SZBD	Podzapytanie	JOIN	Funkcja okna
SQL Server	15s	7s	0s (w przybliżeniu)
PostgreSQL	4 s 100 ms	1s 108 ms	306 ms
SQLite	1s 1 ms	475 ms	280 ms

Wnioski:

- Dla Postgre trzeba było zamienić `YEAR(phist5.date)` na `EXTRACT(YEAR FROM ph.date)`

- Dla SQLite do ekstrakcji daty zostało użyte `strftime('%Y', phist5.date)`
- Zapytania dla MS SQL dla podzapytań i joina trwała kilka/kilkanaście sekund a funkcja okna w mniej niż 1 sekundzie.
- Postgre i SQLite były szybsze od MS SQL, przy czym SQLite był najszybszy

Zadanie 3

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

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

```
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;
```

Wyniki:

MS SQL

	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	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	Rhinbräu Klosterbier	7.75	1	11	11	7

Query executed successfully.localhost (15.0 RTM) | sa (51) | Northwind3 | 00:00:00 | 77 rows

PostgreSQL

	productid	productname	unitprice	categoryid	rowno	rankprice	denserankprice
	38	Côte de Blaye	263.5	1	1	1	1
	43	Ipoh Coffee	46	1	2	2	2
	2	Chang	19	1	3	3	3
	1	Chai	18	1	4	4	4
	76	Lakkalikööri	18	1	5	4	4
	35	Steeleye Stout	18	1	6	4	4
	39	Chartreuse verte	18	1	7	4	4
	70	Outback Lager	15	1	8	8	5
	34	Sasquatch Ale	14	1	9	9	6

SQLite

ProductID	ProductName	UnitPrice	CategoryID	rowno	rankprice	denserankprice
38	Côte de Blaye	263.5	1	1	1	1
43	Ipoh Coffee	46	1	2	2	2
2	Chang	19	1	3	3	3
1	Chai	18	1	4	4	4
35	Steeleye Stout	18	1	5	4	4
39	Chartreuse verte	18	1	6	4	4
76	Lakkalikööri	18	1	7	4	4
70	Outback Lager	15	1	8	8	5
34	Sasquatch Ale	14	1	9	9	6

- **row_number** - Nadaje unikalny numer każdemu wierszowi według kolejności sortowania danych za pomocą **order by unitprice desc**. Oznacza to, że postaje ranking rekordów względem wartości **unitprice**, przy czym w przypadku remisu zostanie nadany kolejny unikatowy identyfikator.
- **rank** - Tworzony jest ranking tak jak w przypadku **row_number** z dwoma wyjątkami:
 - w przypadku remisów nadaje ten sam numer wierszom z tą samą wartością **unitprice**
 - w przypadku remisów dla kolejnej unikalnej wartości **unitprice** nastąpi "przeskok" w numeracji
- **dense_rank** - Numerowanie wygląda tak jak w **rank()** tj. w przypadku remisów nadaje ten sam numer, ale bez przeskakiwania numerów po remisach.

Dla każdego SZBD kolejność w przypadku remisów jest inna. Jednakże w przypadku tego samego SZBD przy wielokrotnym użyciu tych samych zapytań (zarówno z jak i bez funkcji okna) wynik jest taki sam. Oznacza to, że funkcje te są deterministyczne ale zdefiniowane w różny sposób dla różnych SZBD.

Zastosowanie:

- **row_number** - Dla jednoznacznego identyfikowania wierszy według podanej kolejności.
- **rank** - Klasyczny ranking z miejscami jak w zawodach sportowych.
- **dense_rank** - Gdy liczy się liczba unikalnych wartości, nie miejsce.

Zadanie

Spróbuj uzyskać ten sam wynik bez użycia funkcji okna

Wyniki:

```
SELECT
  p1.productid,
  p1.productname,
  p1.unitprice,
  p1.categoryid,
  (
    SELECT COUNT(*) + 1 FROM products p2
    WHERE p2.categoryid = p1.categoryid
    AND p2.unitprice > p1.unitprice
  ) AS rowno,
  (
    SELECT COUNT(*) + 1 FROM products p2
    WHERE p2.categoryid = p1.categoryid
    AND p2.unitprice > p1.unitprice
  ) AS rankprice,
```

```
(
    SELECT COUNT(DISTINCT p2.unitprice) + 1 FROM products p2
    WHERE p2.categoryid = p1.categoryid
    AND p2.unitprice > p1.unitprice
) AS denserankprice
FROM products p1
ORDER BY p1.categoryid, rowno;
```

	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	4	4	4
5	39	Chartreuse verte	18.00	1	4	4	4
6	35	Steeleye Stout	18.00	1	4	4	4
7	76	Lakkalikööri	18.00	1	4	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	9	9	6
11	75	Rhinbrau Klosterbier	7.75	1	11	11	7

Query executed successfully.localhost (15.0 RTM)sa (51)Northwind300:00:0077 rows

Zadanie 4

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

Wyniki:

```
WITH RankedPrice AS (
    SELECT
        YEAR(ph.Date) AS Year,
        ph.ProductID,
        ph.ProductName,
        ph.UnitPrice,
        ph.Date,
        DENSE_RANK() OVER (
            PARTITION BY YEAR(date), ph.Productid
            ORDER BY ph.Unitprice DESC
        ) AS PriceRank
    FROM product_history as ph
)
SELECT *
```

```
FROM RankedPrice
WHERE PriceRank <= 4
ORDER BY Year, ProductID, PriceRank;
```

MS SQL

	Year	ProductID	ProductName	UnitPrice	PriceRank
1	1940	1	Chai	27.98	1
2	1940	1	Chai	27.95	2
3	1940	1	Chai	27.92	3
4	1940	1	Chai	27.87	4
5	1940	2	Chang	28.98	1
6	1940	2	Chang	28.95	2
7	1940	2	Chang	28.92	3
8	1940	2	Chang	28.86	4
9	1940	3	Aniseed Syrup	19.99	1
10	1940	3	Aniseed Syrup	19.97	2
11	1940	3	Aniseed Syrup	19.96	3

PostgreSQL

Trzeba zamienić YEAR(date) na EXTRACT(YEAR FROM ph.Date)

	year	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

SQLite

Trzeba zamienić YEAR(date) na strftime('%Y', ph.Date)

	Year	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

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)

Wyniki:

```
WITH RankedPrice AS (  
    SELECT  
        YEAR(ph.Date) AS Year,  
        ph.ProductID,  
        ph.ProductName,  
        ph.UnitPrice,  
        ph.Date,  
        (  
            SELECT count(DISTINCT ph2.Unitprice) + 1  
            FROM product_history ph2  
            WHERE ph.Productid = ph2.Productid  
            AND YEAR(ph.Date) = YEAR(ph2.Date)  
            AND ph2.Unitprice > ph.Unitprice  
        ) AS PriceRank  
    FROM product_history as ph  
    -- wersja szybsza z ograniczeniem  
    WHERE ph.ProductID < 3  
)  
SELECT *  
FROM RankedPrice  
WHERE PriceRank <= 4  
ORDER BY Year, ProductID, PriceRank;
```

Ze względu, że zapytanie liczyło się prawie godzinę liczymy to wszystko tylko dla dwóch pierwszych produktów.

00:56:52

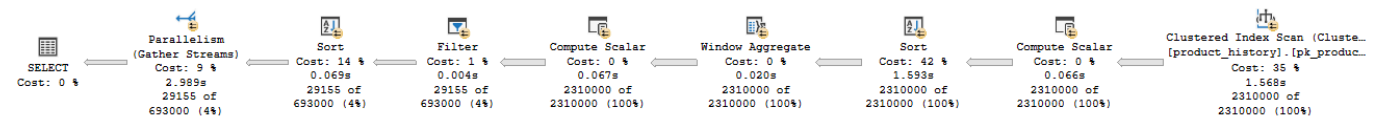
Porównanie czasów :

SZBD	Z funkcją okna	Z funkcją okna (ograniczona)	Bez funkcji okna (ograniczona)
SQL Server	00:00:01	00:00:00	00:01:54
PostgreSQL	12 s 617 ms	587 ms	38 m 40 s i dalej się liczy
SQLite	6 s 684 ms	2 s 525 ms	50 m 51 s i dalej się liczy

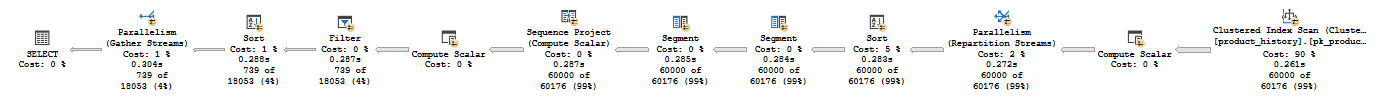
Porównanie planów

MS SQL

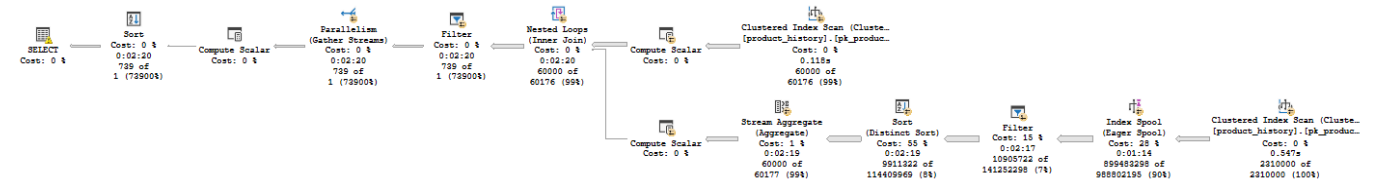
Z funkcją okna



Z funkcją okna (ograniczona)



Bez funkcji okna (ograniczona)



PostgreSQL

Z funkcją okna

← Select

Unknown (Incremental Sort)
Rows: 59598
Cost: 44907.63..55042.26

Transformation (WindowAgg)
Rows: 59598
Cost: 44906.08..53188.2

Unknown (Gather Merge)
Rows: 59598
Cost: 44906.08..51847.25

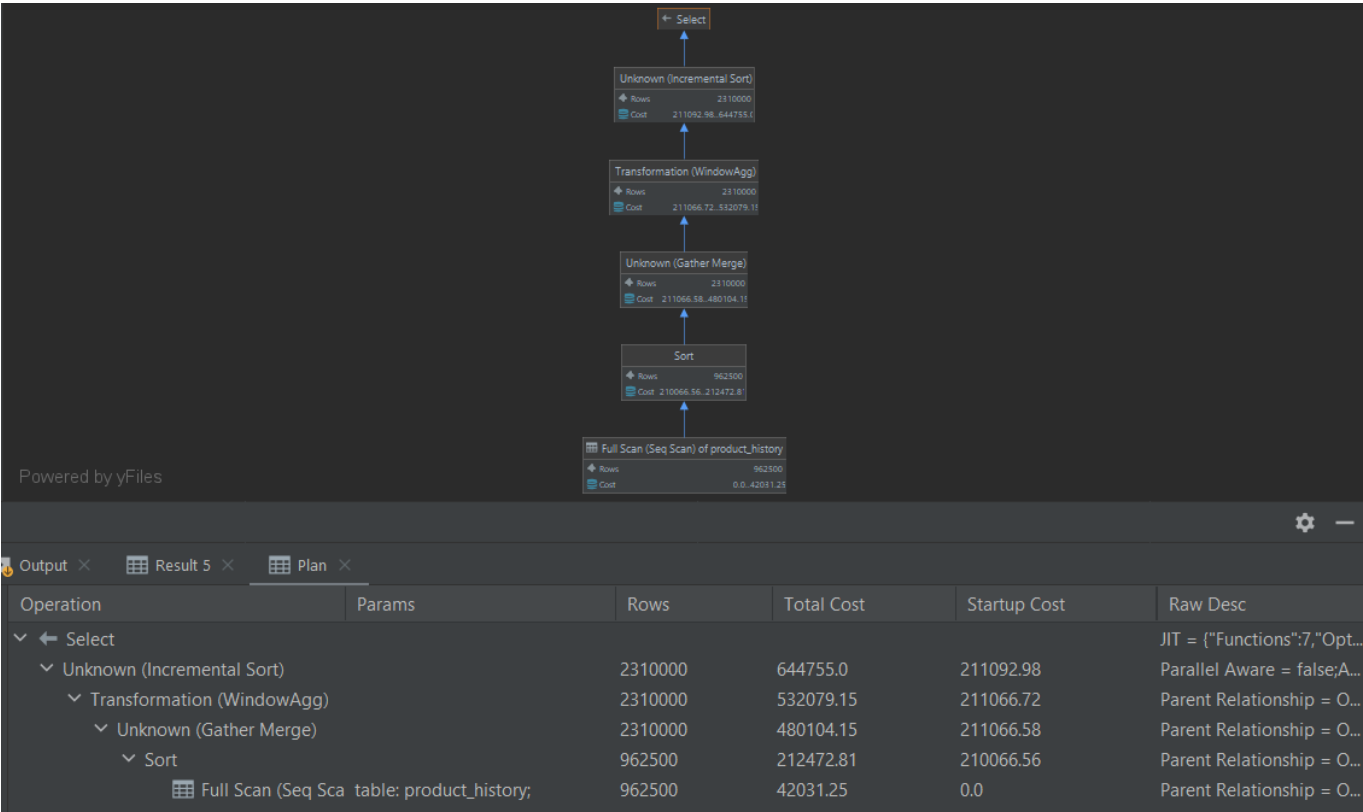
Sort
Rows: 24832
Cost: 43906.06..43968.14

Full Scan (Seq Scan) of product_history
Rows: 24832
Cost: 0.0..42093.33

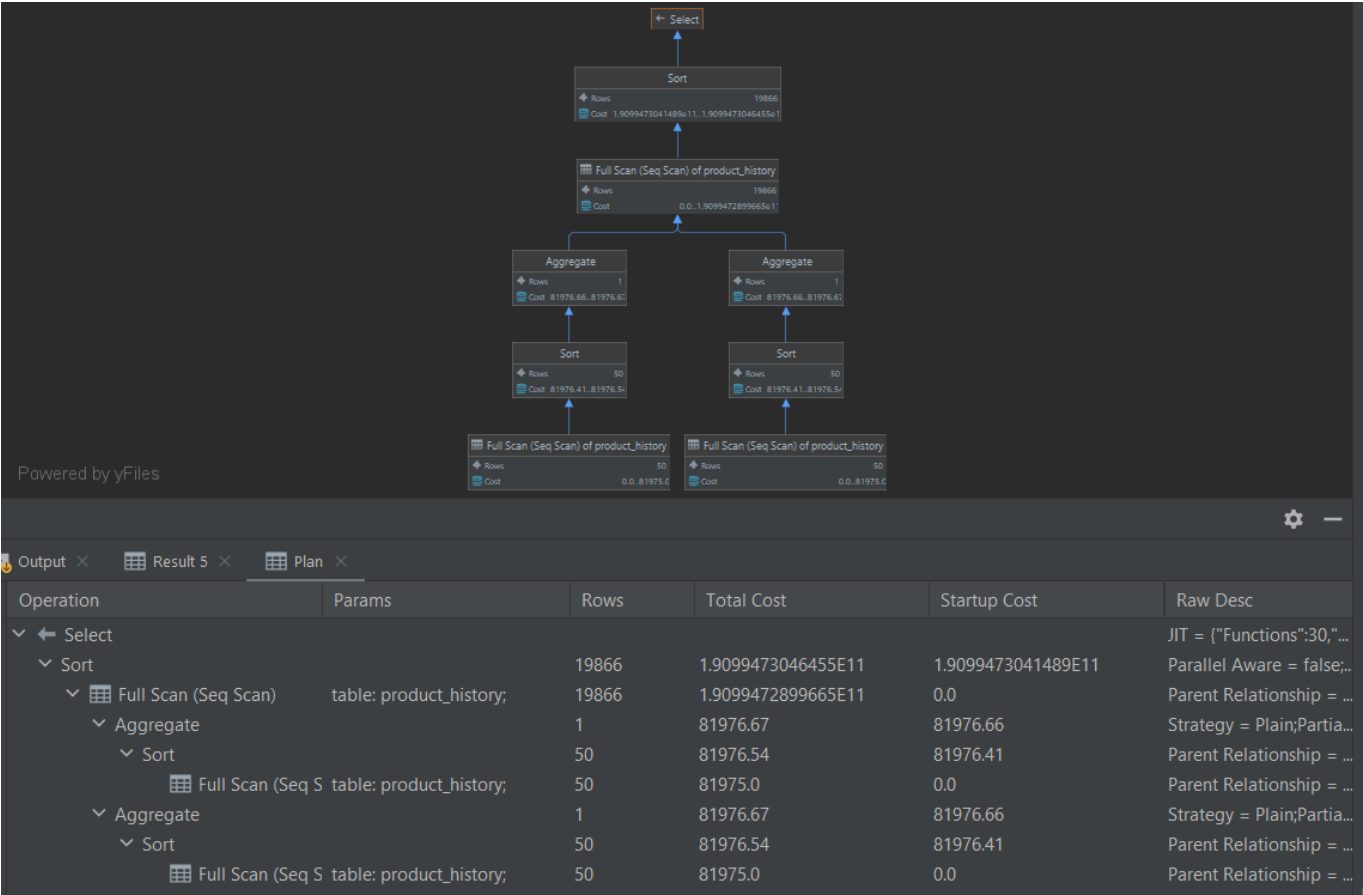
Powered by yFiles

Operation	Params	Rows	Total Cost	Startup Cost	Raw Desc
← Select					
Unknown (Incremental Sort)		59598	55042.26	44907.63	Parallel Aware = false;As...
Transformation (WindowAgg)		59598	53188.2	44906.08	Parent Relationship = Ou...
Unknown (Gather Merge)		59598	51847.25	44906.08	Parent Relationship = Ou...
Sort		24832	43968.14	43906.06	Parent Relationship = Ou...
Full Scan (Seq Sca	table: product_history;	24832	42093.33	0.0	Parent Relationship = Ou...

Z funkcją okna (ograniczona)



Bez funkcji okna (ograniczona)



SQLite

Z funkcją okna

	id	parent	notused	detail
1	2	0	0	CO-ROUTINE RankedPrice
2	5	2	0	CO-ROUTINE (subquery-3)
3	9	5	0	SEARCH ph USING INDEX ix_product_history_productid (productid...
4	26	5	0	USE TEMP B-TREE FOR ORDER BY
5	46	2	0	SCAN (subquery-3)
6	125	0	0	SCAN RankedPrice
7	139	0	0	USE TEMP B-TREE FOR ORDER BY

Z funkcją okna (ograniczona)

	id	parent	notused	detail
1	2	0	0	CO-ROUTINE RankedPrice
2	5	2	0	CO-ROUTINE (subquery-3)
3	8	5	0	SCAN ph
4	21	5	0	USE TEMP B-TREE FOR ORDER BY
5	41	2	0	SCAN (subquery-3)
6	120	0	0	SCAN RankedPrice
7	134	0	0	USE TEMP B-TREE FOR ORDER BY

Bez funkcji okna (ograniczona)

	id	parent	notused	detail
1	4	0	0	SEARCH ph USING INDEX ix_product_history_productid (productid...
2	11	0	0	CORRELATED SCALAR SUBQUERY 1
3	16	11	0	USE TEMP B-TREE FOR count(DISTINCT)
4	19	11	0	SEARCH ph2 USING INDEX ix_product_history_productid (producti...
5	53	0	0	CORRELATED SCALAR SUBQUERY 1
6	58	53	0	USE TEMP B-TREE FOR count(DISTINCT)
7	61	53	0	SEARCH ph2 USING INDEX ix_product_history_productid (producti...
8	91	0	0	USE TEMP B-TREE FOR ORDER BY

Wnioski:

- Korzystanie z funkcji okna znacząco skraca czas wykonania zapytań w porównaniu z alternatywnym podejściem z użyciem podzapytania. W każdym SZBD zauważalne jest przyspieszenie, jednak różnice między użyciem funkcji okna a ich brakiem są najbardziej widoczne w przypadku MS SQL. Tylko dla niego obliczenia zakończyły się w sensownym czasie, gdzie dla pozostałych musiały być zakończone po około 30 minutach.
- Plany wykonania dla MS SQL oraz Postgres są podobne do siebie i stosunkowo proste (w porównaniu do poprzednich zadań).

Zadanie 5

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;

```

Wyniki:

	productid	productname	categoryid	date	unitprice	previousprodprice	nextprodprice
1	1	Chai	1	2022-01-01	25.31	NULL	20.69
2	1	Chai	1	2022-01-02	20.69	25.31	18.16
3	1	Chai	1	2022-01-03	18.16	20.69	23.93
4	1	Chai	1	2022-01-04	23.93	18.16	13.69
5	1	Chai	1	2022-01-05	13.69	23.93	25.34
6	1	Chai	1	2022-01-06	25.34	13.69	17.89
7	1	Chai	1	2022-01-07	17.89	25.34	17.12
8	1	Chai	1	2022-01-08	17.12	17.89	26.73

	productid	productname	categoryid	date	unitprice	previousprodprice	nextprodprice
1	1	Chai	1	2022-01-01	25.31	21.10	20.69
2	1	Chai	1	2022-01-02	20.69	25.31	18.16
3	1	Chai	1	2022-01-03	18.16	20.69	23.93
4	1	Chai	1	2022-01-04	23.93	18.16	13.69
5	1	Chai	1	2022-01-05	13.69	23.93	25.34
6	1	Chai	1	2022-01-06	25.34	13.69	17.89
7	1	Chai	1	2022-01-07	17.89	25.34	17.12
8	1	Chai	1	2022-01-08	17.12	17.89	26.73

Wiersze są sortowane po wartości `date` a następnie

- `lag(unitprice)` - Pobiera wartość `unitprice` z poprzedniego wiersza
- `lead(unitprice)` - Pobiera wartość `unitprice` z następnego wiersza

Jednak wynik dla obydwu zapytań nie jest identyczny, ponieważ

- w pierwszym przypadku dane są najpierw filtrowane a dopiero potem stosowane są `lag` i `lead` przez co mają dostęp tylko do wierszy spełniających `where productid = 1 and year(date) = 2022`

- dla drugiego przypadku najpierw pobierane są wszystkie dane w CTE (`WITH t AS ...`), a dopiero potem są filtrowane. Dlatego `lag` dla daty `2022-01-01` może sięgnąć do dat wcześniejszych niż 2022 i dlatego jego wartość w pierwszym wierszu nie wynosi `NULL`.

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)

Wyniki:

```
SELECT
    ph.productid,
    ph.productname,
    ph.categoryid,
    ph.date,
    ph.unitprice,
    (
        SELECT TOP 1 ph2.unitprice
        FROM product_history ph2
        WHERE ph2.productid = ph.productid
            AND ph2.date < ph.date
            AND YEAR(ph2.date) = 2022
        ORDER BY ph2.date DESC
    ) AS previousprodprice,
    (
        SELECT TOP 1 ph3.unitprice
        FROM product_history ph3
        WHERE ph3.productid = ph.productid
            AND ph3.date > ph.date
            AND YEAR(ph3.date) = 2022
        ORDER BY ph3.date ASC
    ) AS nextprodprice
FROM product_history ph
WHERE ph.productid = 1 AND YEAR(ph.date) = 2022
ORDER BY ph.date;
```

```
with t as (
    SELECT
        ph.productid,
        ph.productname,
        ph.categoryid,
        ph.date,
        ph.unitprice,
        (
            SELECT TOP 1 ph2.unitprice
            FROM product_history ph2
            WHERE ph2.productid = ph.productid
                AND ph2.date < ph.date
```

```

        ORDER BY ph2.date DESC
    ) AS previousprodprice,
    (
        SELECT TOP 1 ph3.unitprice
        FROM product_history ph3
        WHERE ph3.productid = ph.productid
        AND ph3.date > ph.date
        ORDER BY ph3.date ASC
    ) AS nextprodprice
FROM product_history ph
)
select * from t
WHERE productid = 1 AND YEAR(date) = 2022
ORDER BY date;

```

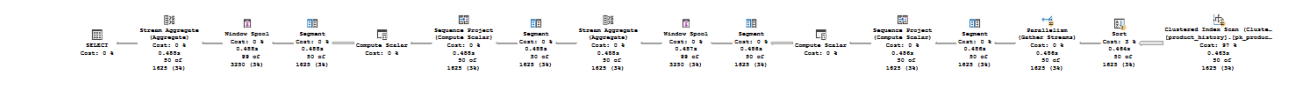
	productid	productname	categoryid	date	unitprice	previousprodprice	nextprodprice
1	1	Chai	1	2022-01-01	25.31	NULL	20.69
2	1	Chai	1	2022-01-02	20.69	25.31	18.16
3	1	Chai	1	2022-01-03	18.16	20.69	23.93
4	1	Chai	1	2022-01-04	23.93	18.16	13.69
5	1	Chai	1	2022-01-05	13.69	23.93	25.34
6	1	Chai	1	2022-01-06	25.34	13.69	17.89
7	1	Chai	1	2022-01-07	17.89	25.34	17.12
8	1	Chai	1	2022-01-08	17.12	17.89	26.73

	productid	productname	categoryid	date	unitprice	previousprodprice	nextprodprice
1	1	Chai	1	2022-01-01	25.31	21.10	20.69
2	1	Chai	1	2022-01-02	20.69	25.31	18.16
3	1	Chai	1	2022-01-03	18.16	20.69	23.93
4	1	Chai	1	2022-01-04	23.93	18.16	13.69
5	1	Chai	1	2022-01-05	13.69	23.93	25.34
6	1	Chai	1	2022-01-06	25.34	13.69	17.89
7	1	Chai	1	2022-01-07	17.89	25.34	17.12

SZBD	Z funkcją okna	Bez funkcji okna
SQL Server	00:00:01	00:00:01
PostgreSQL	4 s 799 ms	5 s 137 ms
SQLite	500 ms	22 s 811 ms

MS SQL

Z funkcją okna

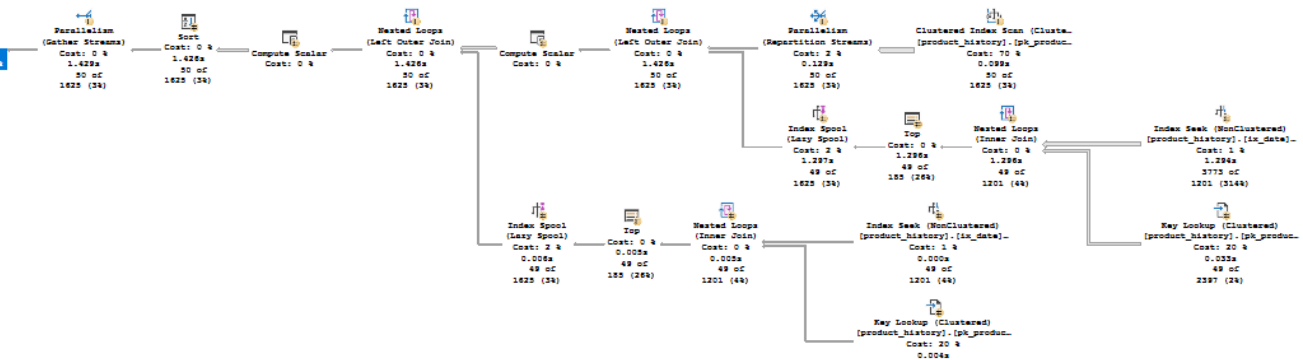


Query 2: Query cost (relative to the batch): 51%

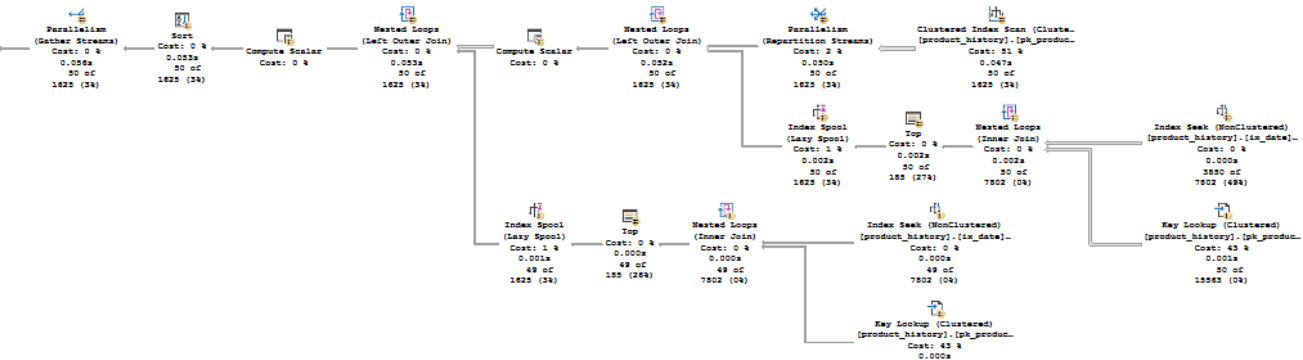
with t as (select productid, productname, categoryid, date, unitprice, lag(unitprice) over (partition by productid order by date) as previousprod...

Missing Index (Impact 93.0997): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[product_history] ([productid]) INCLUDE ([p...

Bez funkcji okna



Query Plan for Query 2 without window function. The plan shows a sequence of operations: Parallelism (Gather Streams), Sort, Compute Scalar, Nested Loops (Left Outer Join), Compute Scalar, Nested Loops (Left Outer Join), Parallelism (Repartition Streams), Clustered Index Scan (Clustered), Index Spool (Lazy Spool), Index Seek (NonClustered), Nested Loops (Inner Join), Index Seek (NonClustered), Key Lookup (Clustered), and Index Seek (NonClustered). The cost is 51%.



Query Plan for Query 2 with window function. The plan shows a sequence of operations: Parallelism (Gather Streams), Sort, Compute Scalar, Nested Loops (Left Outer Join), Compute Scalar, Nested Loops (Left Outer Join), Parallelism (Repartition Streams), Clustered Index Scan (Clustered), Index Spool (Lazy Spool), Index Seek (NonClustered), Nested Loops (Inner Join), Index Seek (NonClustered), Key Lookup (Clustered), and Index Seek (NonClustered). The cost is 51%.

PostgreSQL

Z funkcją okna

← Select

Transformation (WindowAgg)

Rows 165

Cost 47846.02, 47867.99

Unknown (Gather Merge)

Rows 165

Cost 47845.88, 47865.1

Sort

Rows 69

Cost 46845.86, 46846.03

Full Scan (Seq Scan) of product_histc

Rows 69

Cost 0.0, 46843.75

Powered by yFiles

Output × Result 6 × Result 6-2 × Plan ×

Operation	Params	Rows	Total Cost	Startup Cost	Raw Desc
← Select					
Transformation (WindowAgg)		165	47867.99	47846.02	Parallel Aware = false;As...
Unknown (Gather Merge)		165	47865.1	47845.88	Parent Relationship = Ou...
Sort		69	46846.03	46845.86	Parent Relationship = Ou...
Full Scan (Seq Scan table: product_history;		69	46843.75	0.0	Parent Relationship = Ou...

← Select

Access (Subquery Scan)

Rows 165

Cost 43975.29, 48884.36

Transformation (WindowAgg)

Rows 32956

Cost 43975.29, 48390.02

Unknown (Gather Merge)

Rows 32956

Cost 43975.02, 47813.29

Sort

Rows 13732

Cost 42975.0, 43009.33

Full Scan (Seq Scan) of product_history

Rows 13732

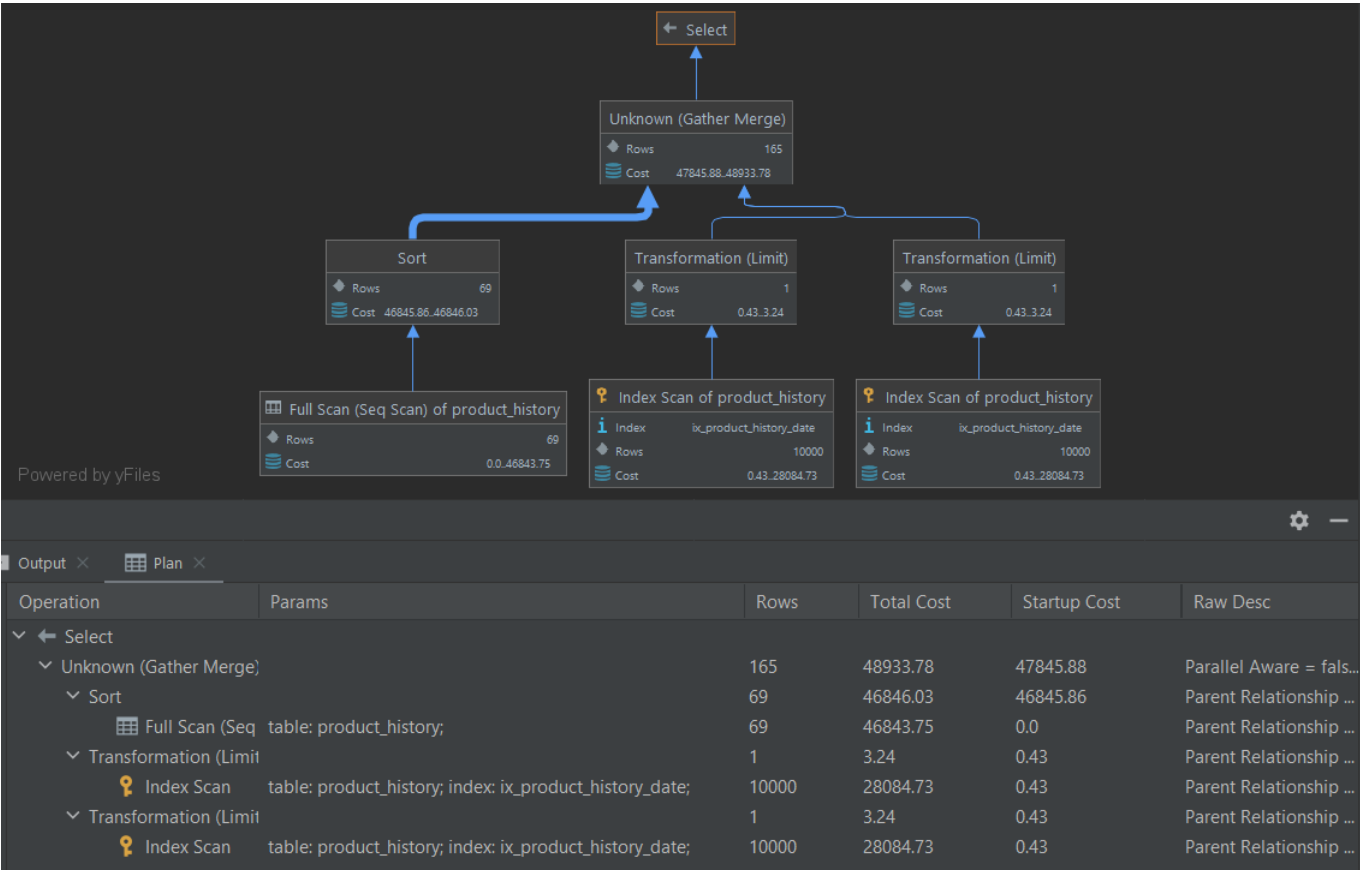
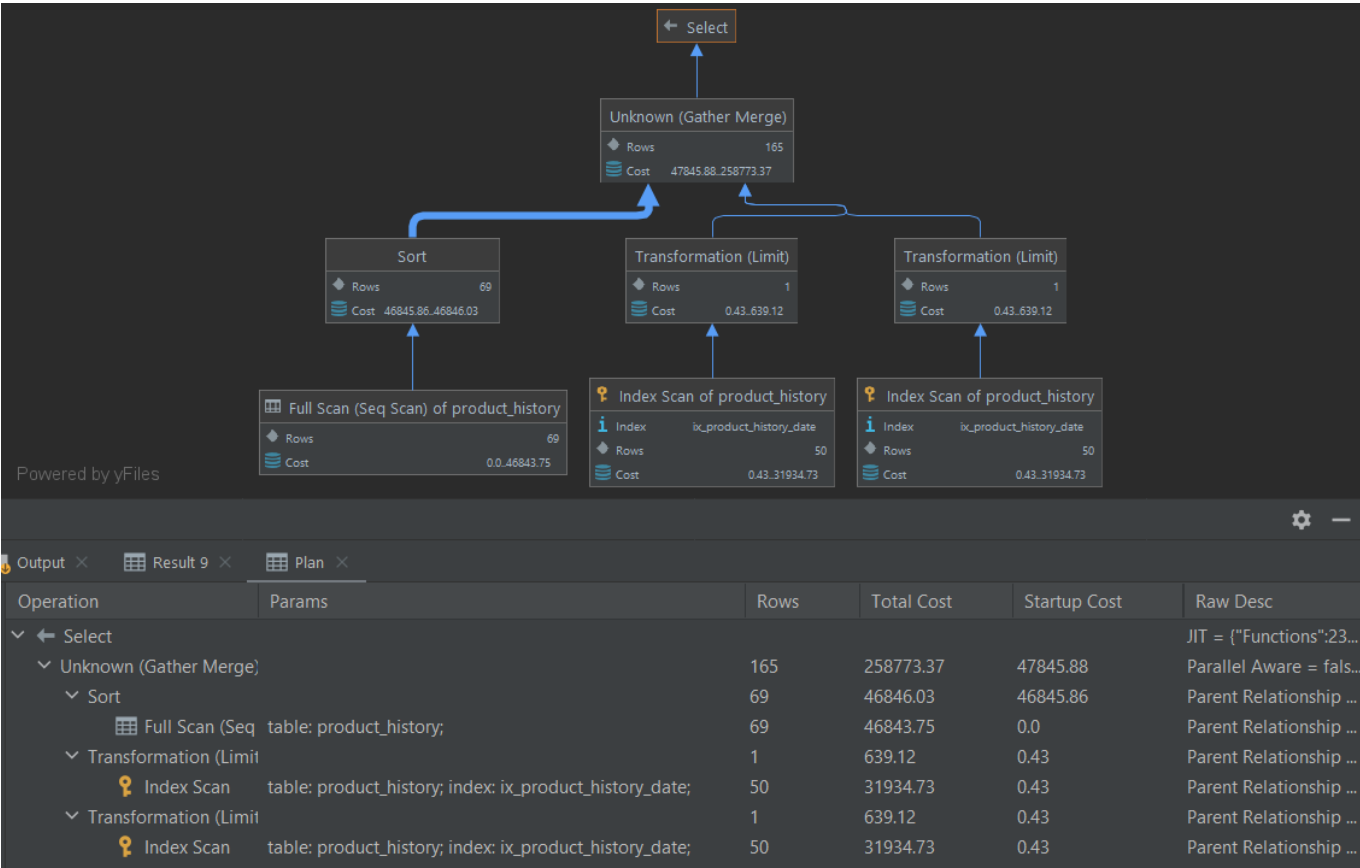
Cost 0.0, 42031.25

Powered by yFiles

Output × Plan ×

Operation	Params	Rows	Total Cost	Startup Cost	Raw Desc
← Select					
Access (Subquery Scan)		165	48884.36	43975.29	Parallel Aware = false;As...
Transformation (WindowAgg)		32956	48390.02	43975.29	Parent Relationship = Su...
Unknown (Gather Merge)		32956	47813.29	43975.02	Parent Relationship = Ou...
Sort		13732	43009.33	42975.0	Parent Relationship = Ou...
Full Scan (Seq Sca table: product_history;		13732	42031.25	0.0	Parent Relationship = Ou...

Bez funkcji okna



SQLite

Z funkcją okna

	id	parent	notused	detail
1	3	0	0	CO-ROUTINE (subquery-2)
2	6	3	0	CO-ROUTINE (subquery-3)
3	10	6	0	SEARCH product_history USING INDEX ix_pr...
4	36	6	0	USE TEMP B-TREE FOR RIGHT PART OF ORDER ...
5	62	3	0	SCAN (subquery-3)
6	125	0	0	SCAN (subquery-2)
7	187	0	0	USE TEMP B-TREE FOR ORDER BY

	id	parent	notused	detail
1	2	0	0	CO-ROUTINE t
2	5	2	0	CO-ROUTINE (subquery-3)
3	8	5	0	CO-ROUTINE (subquery-4)
4	12	8	0	SEARCH product_history USING INDEX ix_pr...
5	35	8	0	USE TEMP B-TREE FOR RIGHT PART OF ORDER ...
6	61	5	0	SCAN (subquery-4)
7	123	2	0	SCAN (subquery-3)
8	186	0	0	SCAN t
9	204	0	0	USE TEMP B-TREE FOR ORDER BY

Bez funkcji okna

	id	parent	notused	detail
1	4	0	0	SEARCH ph USING INDEX ix_product_history...
2	17	0	0	CORRELATED SCALAR SUBQUERY 1
3	27	17	0	SEARCH ph2 USING INDEX ix_product_histor...
4	51	17	0	USE TEMP B-TREE FOR ORDER BY
5	58	0	0	CORRELATED SCALAR SUBQUERY 2
6	68	58	0	SEARCH ph3 USING INDEX ix_product_histor...
7	92	58	0	USE TEMP B-TREE FOR ORDER BY
8	102	0	0	USE TEMP B-TREE FOR ORDER BY

	id	parent	notused	detail
1	4	0	0	SEARCH ph USING INDEX ix_product_history...
2	17	0	0	CORRELATED SCALAR SUBQUERY 1
3	27	17	0	SEARCH ph2 USING INDEX ix_product_histor...
4	48	17	0	USE TEMP B-TREE FOR ORDER BY
5	56	0	0	CORRELATED SCALAR SUBQUERY 2
6	66	56	0	SEARCH ph3 USING INDEX ix_product_histor...
7	87	56	0	USE TEMP B-TREE FOR ORDER BY
8	98	0	0	USE TEMP B-TREE FOR ORDER BY

Wnioski:

- Dla MS SQL i Postgre nie było większej różnicy w czasie wykonanie dla wersji z funkcją okna i bez niej.
- Dla SQLite wersja bez funkcji okna potrzebowała znacznie więcej czasu.
- Plany wykonania dla wszystkich SZBD był prostsze dla wersji używającej funkcji okna.

Zadanie 6

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.

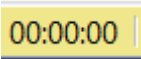
Wyniki:

```
WITH Data AS (  
    SELECT  
        (  
            SELECT CompanyName  
            FROM Customers C  
            WHERE C.CustomerID = O.CustomerID  
        ) AS CompanyName,  
        O.OrderID,  
        O.OrderDate,  
        O.Freight + (  
            SELECT SUM(OD.UnitPrice * OD.Quantity * (1 - OD.Discount))  
            FROM [Order Details] OD  
            WHERE OD.OrderID = O.OrderID  
        ) AS TotalCost,  
        LAG(O.OrderID) OVER (  
            PARTITION BY O.CustomerID ORDER BY O.OrderDate  
        ) AS PrevOrderID,  
        LAG(O.OrderDate) OVER (  
            PARTITION BY O.CustomerID ORDER BY O.OrderDate  
        ) AS PrevOrderDate  
    FROM Orders O  
)  
  
SELECT  
    D.CompanyName,  
    D.OrderID,  
    D.OrderDate,  
    D.TotalCost,
```

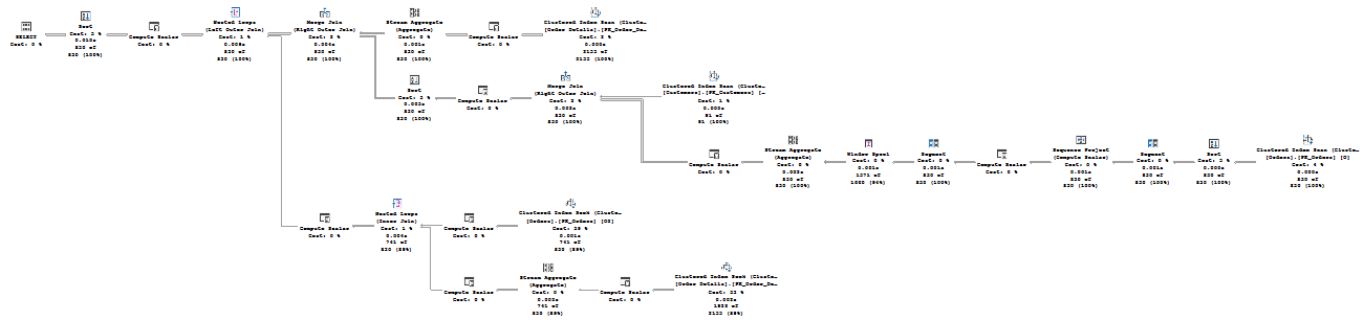
```
D.PrevOrderID,
D.PrevOrderDate,
(
    SELECT O2.Freight + (
        SELECT SUM(OD2.UnitPrice * OD2.Quantity * (1 - OD2.Discount))
        FROM [Order Details] OD2
        WHERE OD2.OrderID = D.PrevOrderID
    )
    FROM Orders O2
    WHERE O2.OrderID = D.PrevOrderID
) AS PrevCost
FROM Data D
ORDER BY D.OrderDate;
```

	CompanyName	OrderID	OrderDate	TotalCost	PrevOrderID	PrevOrderDate	PrevCost
1	Vins et alcools Chevalier	10248	1996-07-04 00:00:00.000	472.38	NULL	NULL	NULL
2	Toms Spezialitäten	10249	1996-07-05 00:00:00.000	1875,00999389648	NULL	NULL	NULL
3	Hanari Carnes	10250	1996-07-08 00:00:00.000	1618,43003662109	NULL	NULL	NULL
4	Victuailles en stock	10251	1996-07-08 00:00:00.000	695,400005187988	NULL	NULL	NULL
5	Suprêmes délices	10252	1996-07-09 00:00:00.000	3649,19990234375	NULL	NULL	NULL
6	Hanari Carnes	10253	1996-07-10 00:00:00.000	1502,96998779297	10250	1996-07-08 00:00:00.000	1618,43003662109
7	Chop-suey Chinese	10254	1996-07-11 00:00:00.000	579,60003326416	NULL	NULL	NULL
8	Richter Supermarkt	10255	1996-07-12 00:00:00.000	2638,83	NULL	NULL	NULL
9	Wellington Importadora	10256	1996-07-15 00:00:00.000	531,770003051758	NULL	NULL	NULL
10	HILARION-Abastos	10257	1996-07-16 00:00:00.000	1201,81000152588	NULL	NULL	NULL
11	Ernst Handel	10258	1996-07-17 00:00:00.000	1755,39000488281	NULL	NULL	NULL
12	Centro comercial Moctezuma	10259	1996-07-18 00:00:00.000	104,049999237061	NULL	NULL	NULL
13	Otilies Käseladen	10260	1996-07-19 00:00:00.000	1559,73999389648	NULL	NULL	NULL
14	Que Delicia	10261	1996-07-19 00:00:00.000	451,05	NULL	NULL	NULL
15	Rattlesnake Canyon Grocery	10262	1996-07-22 00:00:00.000	632,289996185303	NULL	NULL	NULL
16	Ernst Handel	10263	1996-07-23 00:00:00.000	2019,86000305176	10258	1996-07-17 00:00:00.000	1755,39000488281

Czas wykonania



Plan wykonania



Zadanie 7

Funkcje `first_value()`, `last_value()`

Baza: Northwind, tabele customers, orders, order details

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;
```

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

wyjaśnienie działanie funkcji

- `first_value()` - zwraca pierwszy produkt z okna (czyli najdroższy w kategorii)
- `last_value()` - zwraca ostatni produkt z okna do momentu aktualnego wiersza

Domyślnie SQL stosuje ramę okna (window frame) typu:

```
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

To oznacza patrz od początku okna do bieżącego wiersza.

Wyniki:

Należy zmienić zakres ramy okna tak, by obejmował cały przedział w kategorii czyli od pierwszego do ostatniego wiersza.

```

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
    ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
  ) AS last
from products
order by categoryid, unitprice desc;

```

	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 Lumberjack Lager	14,00	1	Côte de Blaye	Guaraná Fantástica
10	34	Sasquatch Ale	14,00	1	Côte de Blaye	Guaraná Fantástica
11	75	Rhönbräu Klosterbier	7,75	1	Côte de Blaye	Guaraná Fantástica
12	24	Guaraná Fantástica	4,50	1	Côte de Blaye	Guaraná Fantástica
13	63	Vegie-spread	43,90	2	Vegie-spread	Aniseed Syrup
14	8	Northwoods Cranberry Sauce	40,00	2	Vegie-spread	Aniseed Syrup
15	61	Sirop d'érable	28,50	2	Vegie-spread	Aniseed Syrup
16	6	Grandma's Boysenberry Spread	25,00	2	Vegie-spread	Aniseed Syrup
17	4	Chef Anton's Cajun Seasoning	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)

Wyniki:

```

SELECT
  p.productid,
  p.productname,
  p.unitprice,
  p.categoryid,
  (

```

```
SELECT TOP 1 p1.productname
FROM products p1
WHERE p1.categoryid = p.categoryid
ORDER BY p1.unitprice DESC
) AS first,
(
SELECT TOP 1 p2.productname
FROM products p2
WHERE p2.categoryid = p.categoryid
ORDER BY p2.unitprice ASC
) AS last
FROM products p
ORDER BY p.categoryid, p.unitprice DESC;
```

	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 Lumberjack Lager	14,00	1	Côte de Blaye	Guaraná Fantástica
10	34	Sasquatch Ale	14,00	1	Côte de Blaye	Guaraná Fantástica
11	75	Rhönbräu Klosterbier	7,75	1	Côte de Blaye	Guaraná Fantástica
12	24	Guaraná Fantástica	4,50	1	Côte de Blaye	Guaraná Fantástica
13	63	Vegie-spread	43,90	2	Vegie-spread	Aniseed Syrup
14	8	Northwoods Cranberry Sauce	40,00	2	Vegie-spread	Aniseed Syrup
15	61	Sirop d'érable	28,50	2	Vegie-spread	Aniseed Syrup
16	6	Grandma's Boysenberry Spread	25,00	2	Vegie-spread	Aniseed Syrup
17	4	Chef Anton's Cajun Seasoning	22,00	2	Vegie-spread	Aniseed Syrup

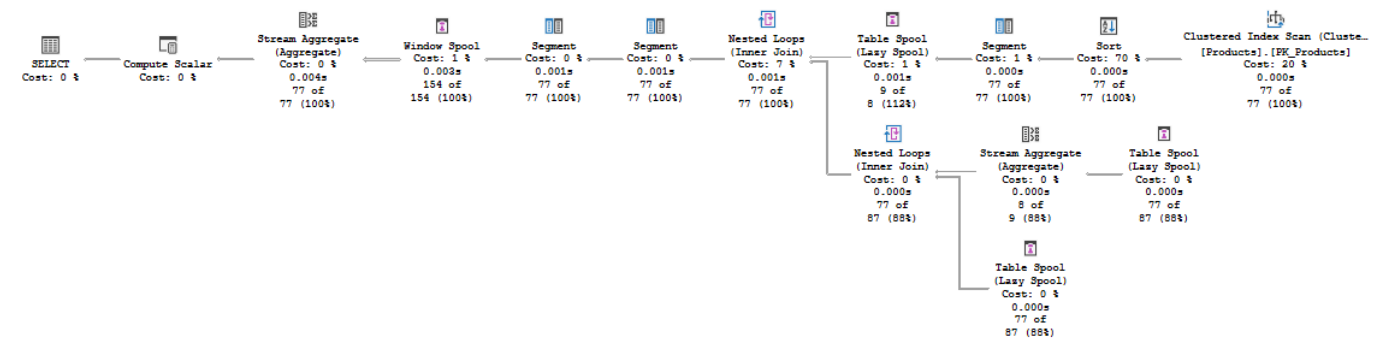
Porównanie czasów

SZBD	Z funkcją okna	Bez funkcji okna
SQL Server	00:00:00	00:00:00
PostgreSQL	881 ms	71 ms
SQLite	205 ms	82 ms

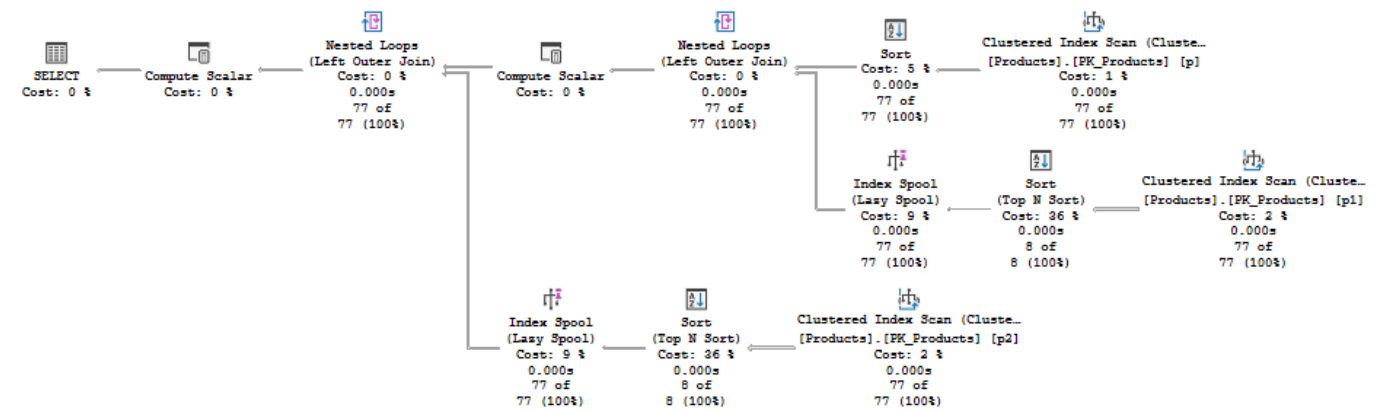
Porównanie planów wykonania

MS SQL

Z funkcją okna



Bez funkcji okna



PostgreSQL

Z funkcją okna

← Select

Transformation (WindowAgg)
Rows: 77
Cost: 4.54-7.07

Transformation (WindowAgg)
Rows: 77
Cost: 4.2-5.72

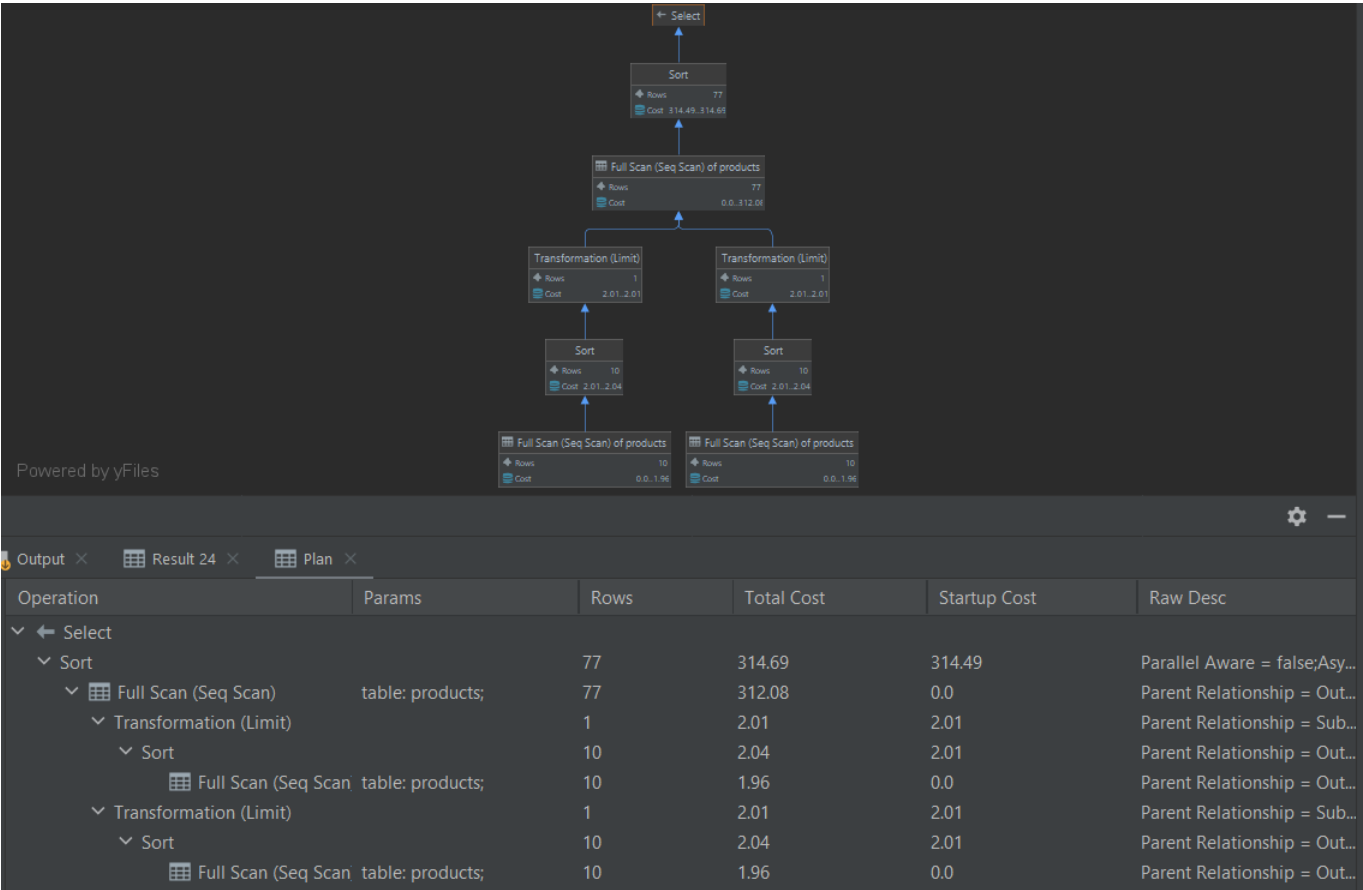
Sort
Rows: 77
Cost: 4.18-4.38

Full Scan (Seq Scan) of produc
Rows: 77
Cost: 0.0-1.77

Powered by yFiles

Operation	Params	Rows	Total Cost	Startup Cost	Raw Desc
← Select					
Transformation (WindowAgg)		77	7.07	4.54	Parallel Aware = false;Asy...
Transformation (WindowAgg)		77	5.72	4.2	Parent Relationship = Out...
Sort		77	4.38	4.18	Parent Relationship = Out...
Full Scan (Seq Scan) table: products;		77	1.77	0.0	Parent Relationship = Out...

Bez funkcji okna



SQLite

Z funkcją okna

	id	parent	notused	detail
1	3	0	0	CO-ROUTINE (subquery-2)
2	6	3	0	CO-ROUTINE (subquery-3)
3	10	6	0	SCAN products USING INDEX ix_products_categoryid
4	30	6	0	USE TEMP B-TREE FOR RIGHT PART OF ORDER BY
5	54	3	0	SCAN (subquery-3)
6	112	0	0	SCAN (subquery-2)

Bez funkcji okna

	id	parent	notused	detail
1	4	0	0	SCAN p USING INDEX ix_products_categoryid
2	11	0	0	CORRELATED SCALAR SUBQUERY 1
3	21	11	0	SEARCH p1 USING INDEX ix_products_categoryid (CategoryID=?)
4	39	11	0	USE TEMP B-TREE FOR ORDER BY
5	46	0	0	CORRELATED SCALAR SUBQUERY 2
6	56	46	0	SEARCH p2 USING INDEX ix_products_categoryid (CategoryID=?)
7	74	46	0	USE TEMP B-TREE FOR ORDER BY
8	91	0	0	USE TEMP B-TREE FOR RIGHT PART OF ORDER BY

Wnioski

- Wszystkie SZBD działają szybko i w każdym przypadku wersja bez funkcji okna jest szybsza.
- Plany wykonania są prostsze dla wersji z funkcją okna.

Zadanie 8

Baza: Northwind, tabele orders, order details

Napisz polecenie które wyświetla inf. o zamówieniach

Zbiór wynikowy 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

Wyniki:

```
WITH OrderValues AS (
    SELECT
        o.OrderID,
        o.CustomerID,
        o.OrderDate,
        DATEPART(YEAR, o.OrderDate) AS Year,
        DATEPART(MONTH, o.OrderDate) AS Month,
        (
            SELECT SUM(od.UnitPrice * od.Quantity * (1 - od.Discount))
            FROM [Order Details] od
            WHERE od.OrderID = o.OrderID
        ) + o.Freight AS OrderTotal
    FROM Orders o
)

SELECT
    ov.CustomerID,
    ov.OrderID,
    ov.OrderDate,
    ov.OrderTotal,

    -- Najtańsze zamówienie klienta w danym miesiącu
    (
        SELECT TOP 1 OrderID
        FROM OrderValues ov2
        WHERE ov2.CustomerID = ov.CustomerID
```

```

        AND ov2.Year = ov.Year
        AND ov2.Month = ov.Month
    ORDER BY ov2.OrderTotal ASC
) AS MinOrderID,

(
    SELECT TOP 1 OrderDate
    FROM OrderValues ov2
    WHERE ov2.CustomerID = ov.CustomerID
        AND ov2.Year = ov.Year
        AND ov2.Month = ov.Month
    ORDER BY ov2.OrderTotal ASC
) AS MinOrderDate,

(
    SELECT TOP 1 OrderTotal
    FROM OrderValues ov2
    WHERE ov2.CustomerID = ov.CustomerID
        AND ov2.Year = ov.Year
        AND ov2.Month = ov.Month
    ORDER BY ov2.OrderTotal ASC
) AS MinOrderValue,

-- Najdroższe zamówienie klienta w danym miesiącu
(
    SELECT TOP 1 OrderID
    FROM OrderValues ov3
    WHERE ov3.CustomerID = ov.CustomerID
        AND ov3.Year = ov.Year
        AND ov3.Month = ov.Month
    ORDER BY ov3.OrderTotal DESC
) AS MaxOrderID,

(
    SELECT TOP 1 OrderDate
    FROM OrderValues ov3
    WHERE ov3.CustomerID = ov.CustomerID
        AND ov3.Year = ov.Year
        AND ov3.Month = ov.Month
    ORDER BY ov3.OrderTotal DESC
) AS MaxOrderDate,

(
    SELECT TOP 1 OrderTotal
    FROM OrderValues ov3
    WHERE ov3.CustomerID = ov.CustomerID
        AND ov3.Year = ov.Year
        AND ov3.Month = ov.Month
    ORDER BY ov3.OrderTotal DESC
) AS MaxOrderValue

FROM OrderValues ov
ORDER BY ov.CustomerID, ov.OrderDate;

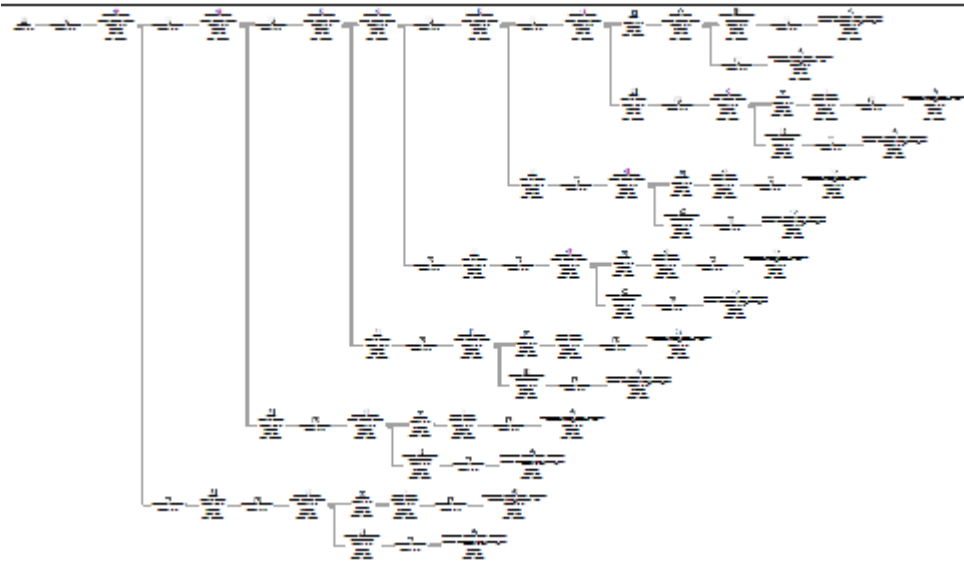
```

	CustomerID	OrderID	OrderDate	OrderTotal	MinOrderID	MinOrderDate	MinOrderValue	MaxOrderID	MaxOrderDate	MaxOrderValue
1	ALFKI	10643	1997-08-25 00:00:00.000	843.96	10643	1997-08-25 00:00:00.000	843.96	10643	1997-08-25 00:00:00.000	843.96
2	ALFKI	10692	1997-10-03 00:00:00.000	939.02	10702	1997-10-13 00:00:00.000	353.94	10692	1997-10-03 00:00:00.000	939.02
3	ALFKI	10702	1997-10-13 00:00:00.000	353.94	10702	1997-10-13 00:00:00.000	353.94	10692	1997-10-03 00:00:00.000	939.02
4	ALFKI	10835	1998-01-15 00:00:00.000	915.330001144409	10835	1998-01-15 00:00:00.000	915.330001144409	10835	1998-01-15 00:00:00.000	915.330001144409
5	ALFKI	10952	1998-03-16 00:00:00.000	511.619996948242	10952	1998-03-16 00:00:00.000	511.619996948242	10952	1998-03-16 00:00:00.000	511.619996948242
6	ALFKI	11011	1998-04-09 00:00:00.000	934.71	11011	1998-04-09 00:00:00.000	934.71	11011	1998-04-09 00:00:00.000	934.71
7	ANATR	10308	1996-09-18 00:00:00.000	90.4099992370605	10308	1996-09-18 00:00:00.000	90.4099992370605	10308	1996-09-18 00:00:00.000	90.4099992370605
8	ANATR	10625	1997-08-08 00:00:00.000	523.65	10625	1997-08-08 00:00:00.000	523.65	10625	1997-08-08 00:00:00.000	523.65
9	ANATR	10759	1997-11-28 00:00:00.000	331.99	10759	1997-11-28 00:00:00.000	331.99	10759	1997-11-28 00:00:00.000	331.99
10	ANATR	10926	1998-03-04 00:00:00.000	554.320001525879	10926	1998-03-04 00:00:00.000	554.320001525879	10926	1998-03-04 00:00:00.000	554.320001525879
11	ANTON	10365	1996-11-27 00:00:00.000	425.200012207031	10365	1996-11-27 00:00:00.000	425.200012207031	10365	1996-11-27 00:00:00.000	425.200012207031
12	ANTON	10507	1997-04-15 00:00:00.000	796.5125	10507	1997-04-15 00:00:00.000	796.5125	10507	1997-04-15 00:00:00.000	796.5125
13	ANTON	10535	1997-05-13 00:00:00.000	1956.48999084473	10535	1997-05-13 00:00:00.000	1956.48999084473	10535	1997-05-13 00:00:00.000	1956.48999084473
14	ANTON	10573	1997-06-19 00:00:00.000	2166.84	10573	1997-06-19 00:00:00.000	2166.84	10573	1997-06-19 00:00:00.000	2166.84
15	ANTON	10677	1997-09-22 00:00:00.000	817.395051269531	10682	1997-09-25 00:00:00.000	411.63	10677	1997-09-22 00:00:00.000	817.395051269531
16	ANTON	10682	1997-09-25 00:00:00.000	411.63	10682	1997-09-25 00:00:00.000	411.63	10677	1997-09-22 00:00:00.000	817.395051269531
17	ANTON	10856	1998-01-28 00:00:00.000	718.43	10856	1998-01-28 00:00:00.000	718.43	10856	1998-01-28 00:00:00.000	718.43

Czas:

00:00:01

Plan:



Zadanie 9

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

W przypadku długiego czasu wykonania ogranicz zbiór wynikowy do kilkuset/kilku tysięcy wierszy

```
SELECT
    ph.id,
    ph.productid,
    ph.date,

    sum(ph.value) over(
        partition by productid, YEAR(ph.date), MONTH(ph.date), DAY(ph.date)
    ) dayValue,

    SUM(ph.value) OVER (
        PARTITION BY ph.productid,
                     YEAR(ph.date),
                     MONTH(ph.date)
        ORDER BY ph.date, ph.id
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS cumulative_monthly_sales

FROM product_history ph
ORDER BY ph.productid, ph.date;
```

	id	productid	date	dayValue	cumulative_monthly_sales
1	1	1	1940-01-02	158.50	158.50
2	78	1	1940-01-03	256.70	415.20
3	155	1	1940-01-04	274.20	689.40
4	232	1	1940-01-05	140.30	829.70
5	309	1	1940-01-06	265.80	1095.50
6	386	1	1940-01-07	202.50	1298.00
7	463	1	1940-01-08	212.30	1510.30
8	540	1	1940-01-09	187.40	1697.70
9	617	1	1940-01-10	233.10	1930.80
10	694	1	1940-01-11	210.20	2141.00
11	771	1	1940-01-12	178.70	2319.70
12	848	1	1940-01-13	125.50	2445.20
13	925	1	1940-01-14	177.20	2622.40
14	1002	1	1940-01-15	198.00	2820.40
15	1079	1	1940-01-16	270.80	3091.20
16	1156	1	1940-01-17	188.20	3279.40
17	1233	1	1940-01-18	218.80	3498.20
18	1310	1	1940-01-19	106.80	3605.00
19	1387	1	1940-01-20	114.90	3719.90
20	1464	1	1940-01-21	204.10	3924.00
21	1541	1	1940-01-22	182.70	4106.70
22	1618	1	1940-01-23	144.60	4251.30
23	1695	1	1940-01-24	180.60	4431.90
24	1772	1	1940-01-25	169.70	4601.60
25	1849	1	1940-01-26	202.70	4804.30
26	1926	1	1940-01-27	232.90	5037.20
27	2003	1	1940-01-28	133.10	5170.30
28	2080	1	1940-01-29	272.70	5443.00
29	2157	1	1940-01-30	185.20	5628.20
30	2234	1	1940-01-31	141.50	5769.70
31	2311	1	1940-02-01	118.60	118.60
32	2388	1	1940-02-02	162.30	280.90
33	2465	1	1940-02-03	228.00	508.90

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)

Wyniki: -

```
SELECT
  ph1.id,
  ph1.productid,
  ph1.date,
  sum(ph1.value) over(
    partition by ph1.productid,
    YEAR(ph1.date),
    MONTH(ph1.date),
```

```
DAY(ph1.date)
) dayValue,

(
  SELECT SUM(ph2.value)
  FROM product_history ph2
  WHERE ph2.productid = ph1.productid
    AND FORMAT(ph2.date, 'yyyy-MM') = FORMAT(ph1.date, 'yyyy-MM')
    AND (
      ph2.date < ph1.date OR (ph2.date = ph1.date AND ph2.id <= ph1.id)
    )
) AS cumulative_monthly_sales

FROM product_history ph1
ORDER BY ph1.productid, ph1.date;
```

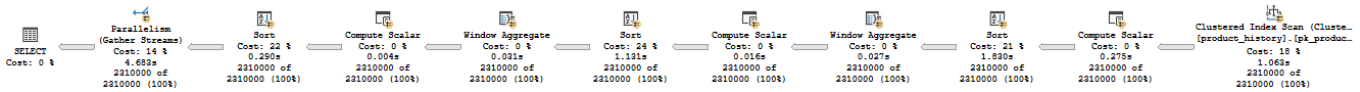
Porównanie czasów:

SZBD	Z funkcją okna	Bez funkcji okna
SQL Server	00:00:18	00:30:07 (zapytanie zostało przerwane)
PostgreSQL	15 s 390 ms	18 s 852 ms
SQLite	19 s 869 ms	19 m 8 s (zapytanie zostało przerwane)

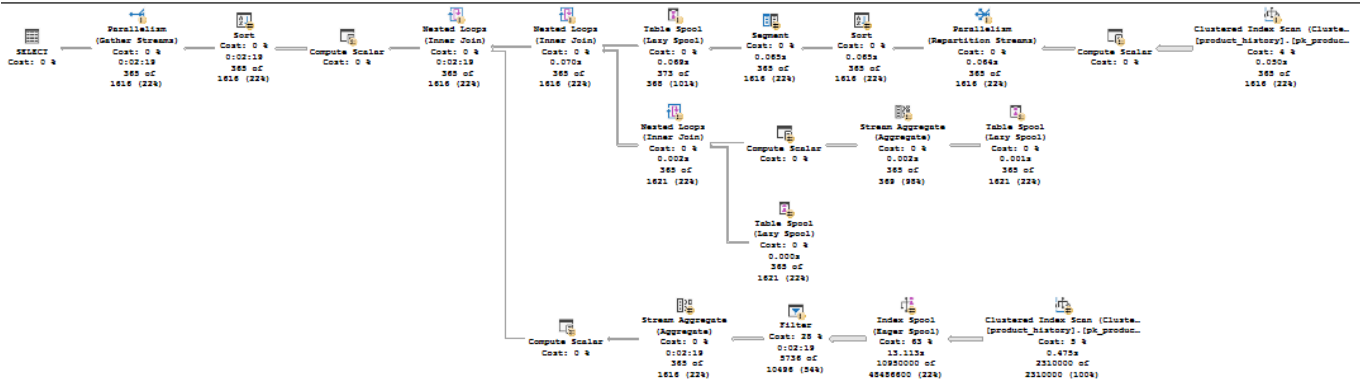
Porównanie planów:

MS SQL

Z funkcją okna

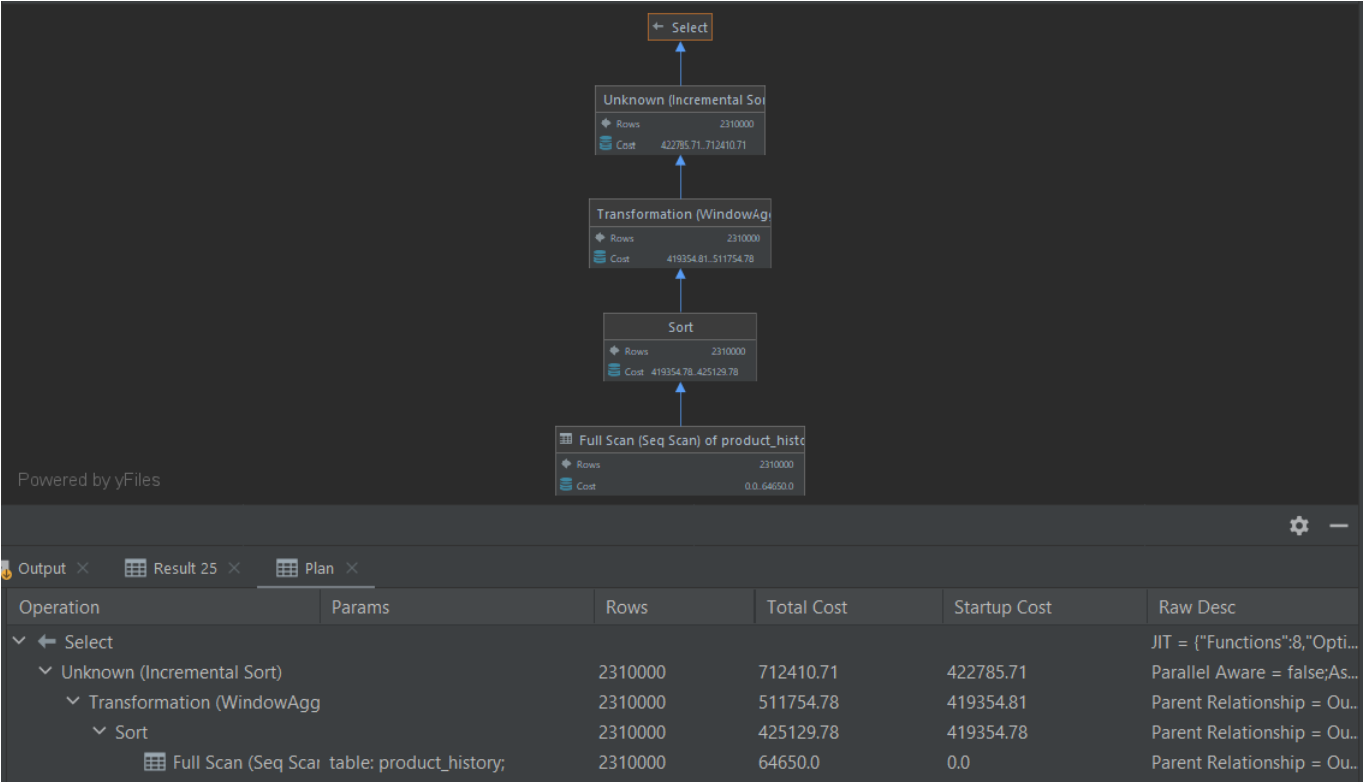


Bez funkcji okna

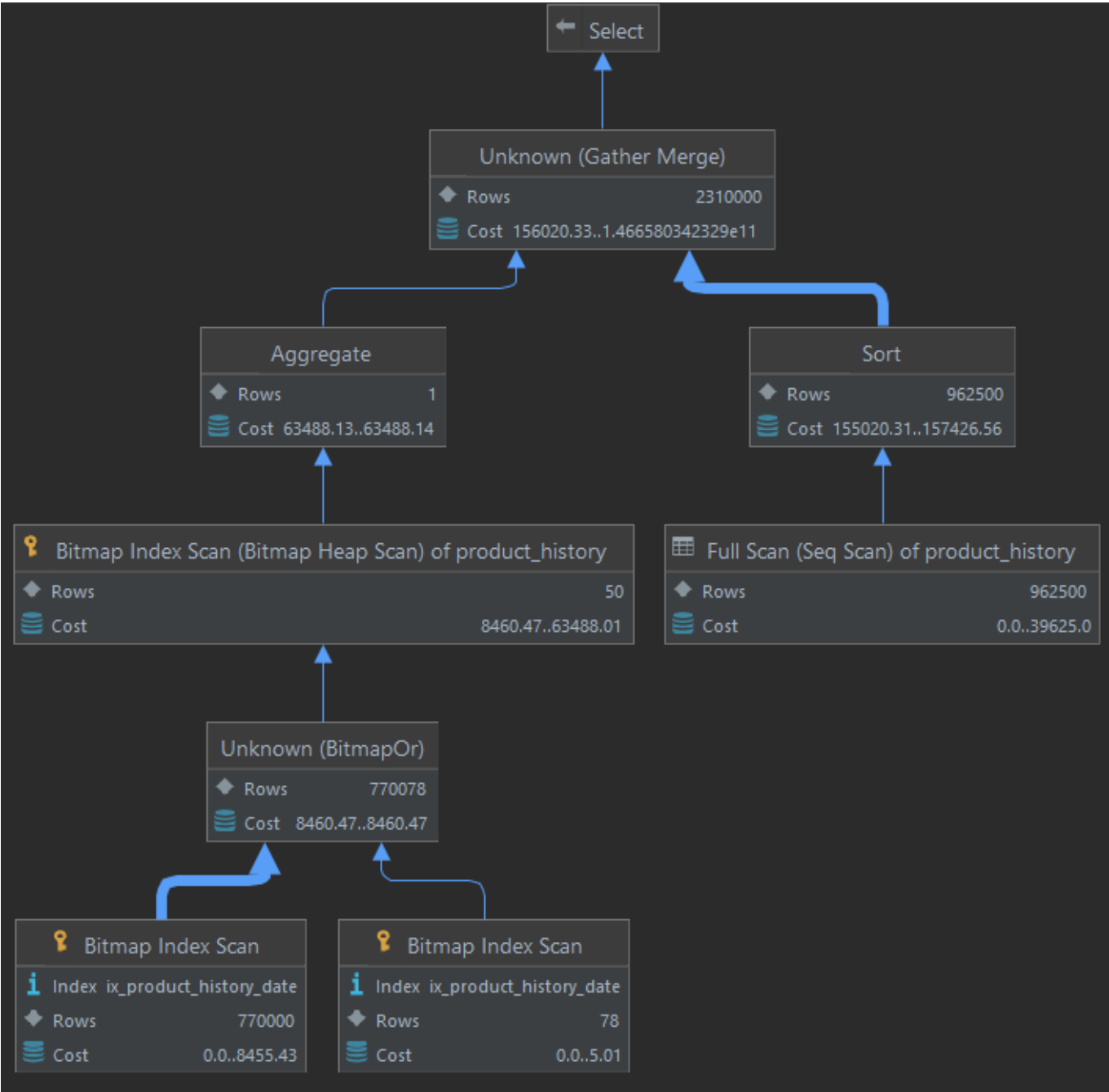


PostgreSQL

Z funkcją okna



Bez funkcji okna



Operation	Params	Rows	Total Cost	Startup Cost	Raw Desc
← Select					JIT = {"Functions":13,"...
Unknown (Gather Merge)		2310000	1.466580342329E11	156020.33	Parallel Aware = false...
Sort		962500	157426.56	155020.31	Parent Relationship = ...
Full Scan (Seq Scan)	table: product_history;	962500	39625.0	0.0	Parent Relationship = ...
Aggregate		1	63488.14	63488.13	Strategy = Plain;Parti...
Bitmap Index Scan (Bi	table: product_history;	50	63488.01	8460.47	Parent Relationship = ...
Unknown (BitmapOr)		770078	8460.47	8460.47	Parent Relationship = ...
Bitmap Index S	index: ix_product_history_date;	770000	8455.43	0.0	Parent Relationship = ...
Bitmap Index S	index: ix_product_history_date;	78	5.01	0.0	Parent Relationship = ...

SQLite

Z funkcją okna

	id	parent	notused	detail
1	3	0	0	CO-ROUTINE (subquery-2)
2	7	3	0	SCAN ph USING INDEX ix_product_history_productid
3	33	3	0	USE TEMP B-TREE FOR RIGHT PART OF ORDER BY
4	61	0	0	SCAN (subquery-2)
5	118	0	0	USE TEMP B-TREE FOR ORDER BY

Bez funkcji okna

	id	parent	notused	detail
1	4	0	0	SCAN ph1 USING INDEX ix_product_history_productid
2	11	0	0	CORRELATED SCALAR SUBQUERY 1
3	17	11	0	SEARCH ph2 USING INDEX ix_product_history_productid (producti...
4	57	0	0	USE TEMP B-TREE FOR RIGHT PART OF ORDER BY

Wnioski:

- PostgreSQL korzysta z **Bitmap index** dla kolumny **date** w tabeli **product_history**. To znaczy zamiast przeglądać każdy wiersz tworzy bitmapę pozycji spełniających warunek `DATE_TRUNC('month', ph2.date) = DATE_TRUNC('month', ph1.date)`, a potem czyta tylko te wybrane bloki z dysku.
- MS SQL i SQLite przeszukują wiersz po wierszu przez co zapytania dla nich zajmują znacznie większą ilość czasu.

Zadanie 10

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.

Wyniki:

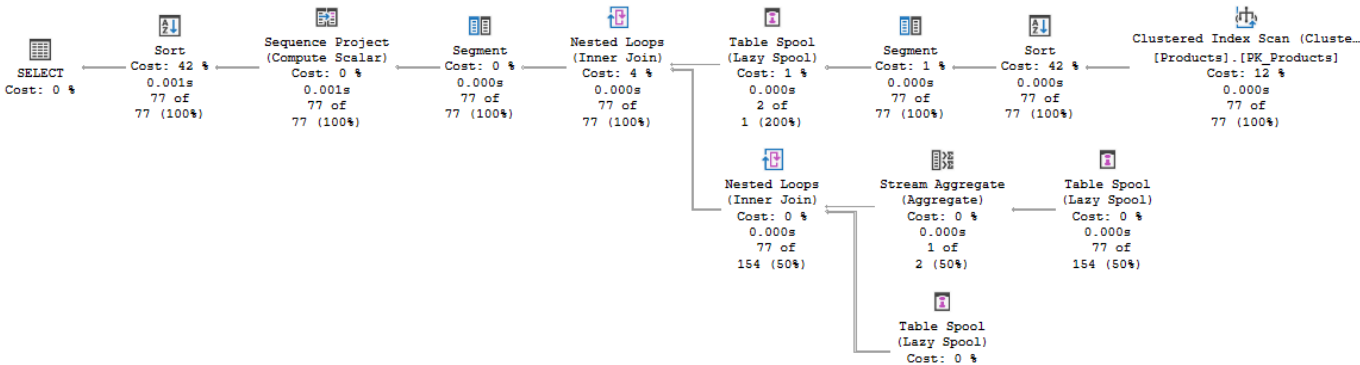
```
-- Podziel produkty na kwartyle cenowe.
SELECT
    ProductID,
    ProductName,
    UnitPrice,
    NTILE(4) OVER (ORDER BY UnitPrice) AS PriceQuartile
FROM Products
ORDER BY PriceQuartile, UnitPrice;
```

	ProductID	ProductName	UnitPrice	PriceQuartile
1	33	Geitost	2,50	1
2	24	Guaraná Fantástica	4,50	1
3	13	Konbu	6,00	1
4	52	Filo Mix	7,00	1
5	54	Tourtière	7,45	1
6	75	Rhönbräu Klosterbier	7,75	1
7	23	Tunnbröd	9,00	1
8	19	Teatime Chocolate Biscuits	9,20	1
9	47	Zaanse koeken	9,50	1
10	45	Rogede sild	9,50	1
11	41	Jack's New England Clam Chowder	9,65	1
12	21	Sir Rodney's Scones	10,00	1
13	3	Aniseed Syrup	10,00	1
14	74	Longlife Tofu	10,00	1
15	46	Spegesild	12,00	1
16	68	Scottish Longbreads	12,50	1
17	31	Gorgonzola Telino	12,50	1
18	48	Chocolade	12,75	1
19	77	Original Frankfurter grüne Soße	13,00	1
20	58	Escargots de Bourgogne	13,25	1
21	67	Laughing Lumberjack Lager	14,00	2
22	42	Singaporean Hokkien Fried Mee	14,00	2
23	34	Sasquatch Ale	14,00	2
24	25	NuNuCa Nuß-Nougat-Creme	14,00	2
25	73	Röd Kaviar	15,00	2
26	70	Outback Lager	15,00	2
27	15	Genen Shouyu	15,50	2
28	50	Valkoinen suklaa	16,25	2
29	66	Louisiana Hot Spiced Okra	17,00	2
30	16	Pavlova	17,45	2
31	1	Chai	18,00	2
32	35	Steeleye Stout	18,00	2
33	39	Chartreuse verte	18,00	2

czas:

00:00:00

plan:



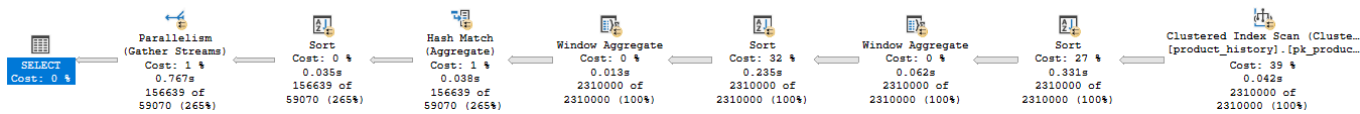
```
-- Podziel wartości zamówień na kwartyly cenowe
SELECT distinct
    value,
    NTILE(4) OVER (ORDER BY value) AS valueQuartile
FROM product_history
ORDER BY valueQuartile, value;
```

	value	valueQuartile
1	100.00	1
2	100.10	1
3	100.20	1
4	100.30	1
5	100.40	1
6	100.50	1
7	100.60	1
8	100.70	1
9	100.80	1
10	100.90	1
11	101.00	1
12	101.10	1
13	101.20	1
14	101.30	1
15	101.40	1
16	101.50	1

czas:

00:00:01 |

plan:



Punktacja

zadanie	pkt
1	2
2	2
3	2
4	2
5	2
6	2

7	2
8	2
9	2
10	2
razem	20