SQL - Funkcje okna (Window functions)
Lab 2

Imiona i nazwiska: Damian Torbus, Adam Woźny

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 Managment Studio
    - DtataGrip 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 Srerver

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 Srerver

```
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(1) FROM Northwind3.product_history;
-- 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`

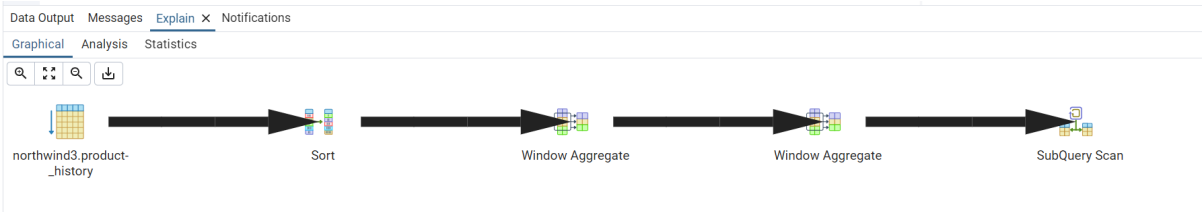
```
--okno
WITH PRODUCT_PRICES AS
    (SELECT ID
        , PRODUCTID
        , PRODUCTNAME
```

```
        , CATEGORYID
        , UNITPRICE
        , AVG(UNITPRICE) OVER (PARTITION BY CATEGORYID) AS AVG_PRICE
        , AVG(UNITPRICE) OVER () AS OVERALL_AVG
    FROM Northwind3.PRODUCT_HISTORY)
SELECT * FROM PRODUCT_PRICES WHERE UNITPRICE > OVERALL_AVG;

-- join
SELECT ID
    , p.CATEGORID
    , p.PRODUCTID
    , p.PRODUCTNAME
    , p.UNITPRICE
    , q.AVG_PRICE
FROM Northwind3.PRODUCT_HISTORY p
INNER JOIN (SELECT CATEGORYID, AVG(UNITPRICE) AS AVG_PRICE
            FROM Northwind3.PRODUCT_HISTORY
            GROUP BY CATEGORYID) q
    ON p.CATEGORYID = q.CATEGORYID
WHERE p.UNITPRICE > (SELECT AVG(UNITPRICE)
                     FROM Northwind3.PRODUCT_HISTORY);

-- pozapytanie
SELECT ID
    , CATEGORYID
    , p.PRODUCTID
    , p.PRODUCTNAME
    , p.UNITPRICE
    , (SELECT AVG(UNITPRICE)
       FROM Northwind3.PRODUCT_HISTORY q
       WHERE q.CATEGORYID = p.CATEGORYID) AS AVG_PRICE
FROM Northwind3.PRODUCT_HISTORY p
WHERE p.UNITPRICE > (SELECT AVG(UNITPRICE)
                     FROM Northwind3.PRODUCT_HISTORY);
```

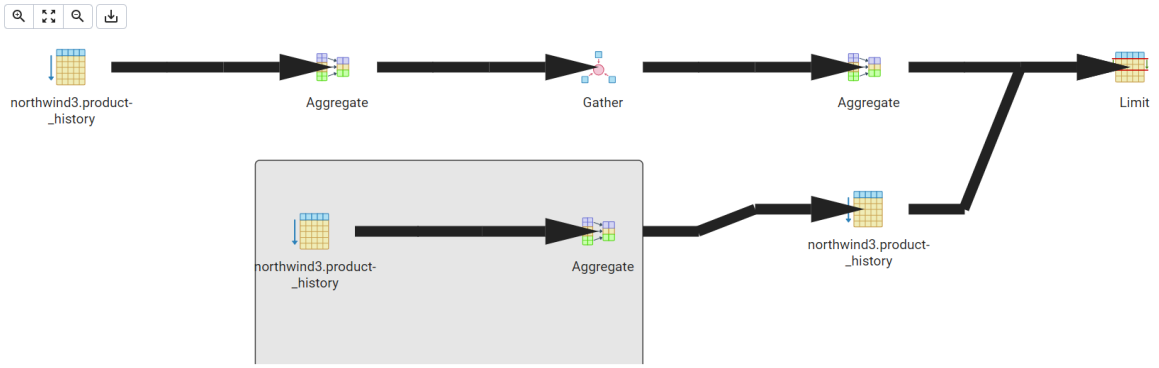
Napisz polecenie z wykorzystaniem podzapytania, join'a oraz funkcji okna. Porównaj zapytania. Porównaj czasy oraz plany wykonania zapytań.

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

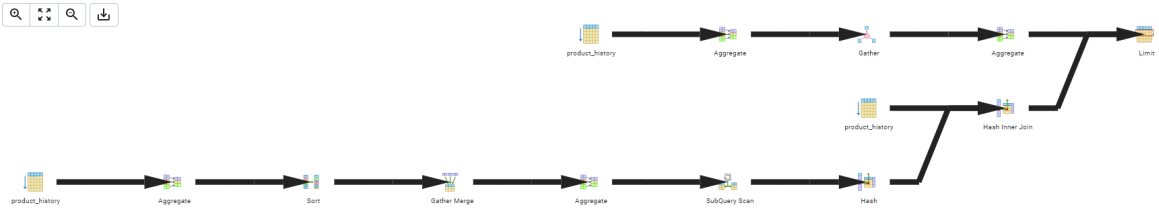
Wyniki PostgreSQL - wyniki limitowane do 500

1. z funckją okna
    - Czas wykonania 3s
    - 

2. z podzapytaniem
    - Czas wykonanie 71s

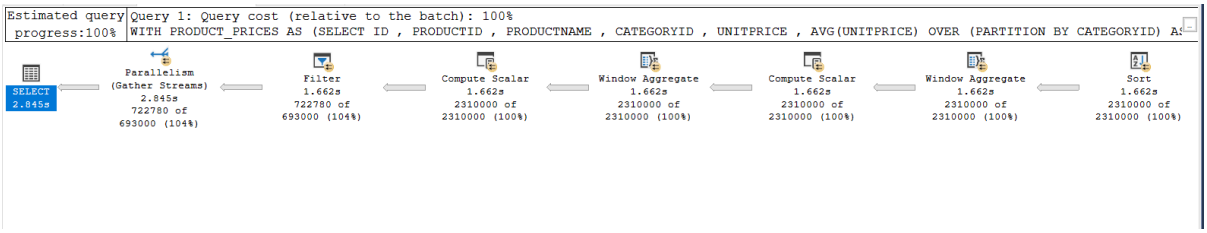


3. z joinem
   - Czas wykonanie 0.5s

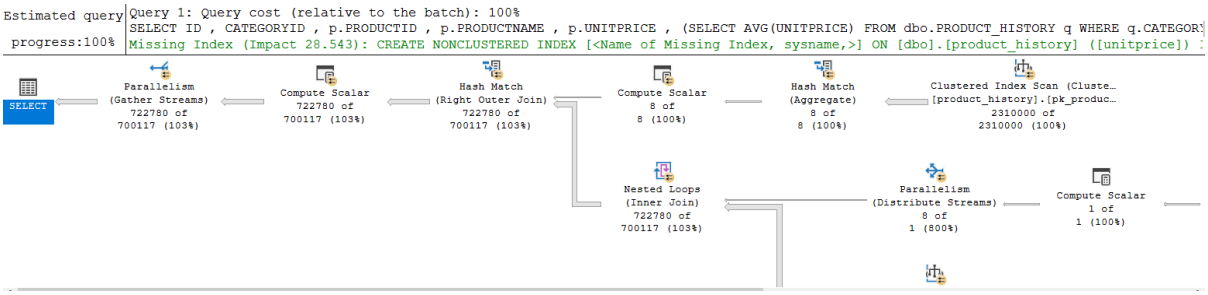


Wyniki MsSQL

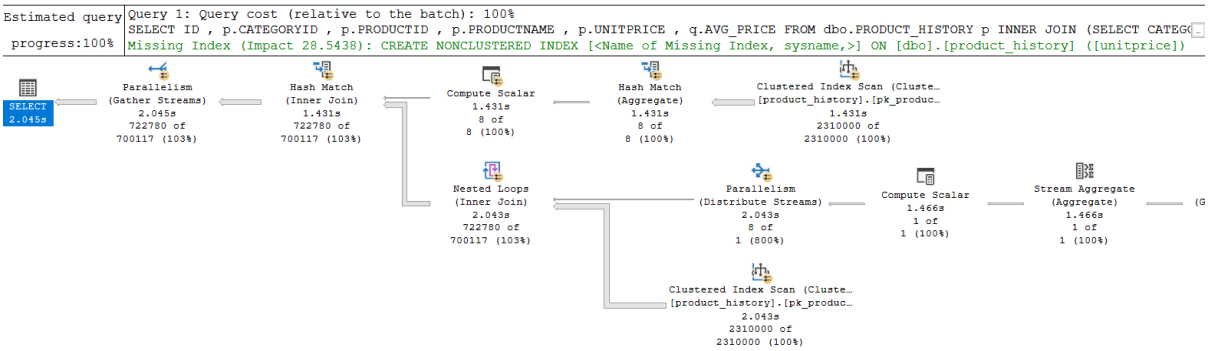
1. z funckją okna
   - Czas wykonania 2.8s



2. z podzapytaniem
   - Czas wykonanie 6s



3. z joinem
   - Czas wykonanie 2s



Wyniki SQlite

1. z funckją okna
   - Czas wykonania 16,9s

	ID	PRODUCTID	PRODUCTNAME	CATEGORYID	UNITPRICE	AVG_PRICE	OVERALL_AVG
1	38	38	Côte de Blaye	1	95.67	29.0594604444444	24.4862882683983
2	43	43	Ipoh Coffee	1	24.96	29.0594604444444	24.4862882683983
3	78	1	Chai	1	25.67	29.0594604444444	24.4862882683983
4	79	2	Chang	1	26.54	29.0594604444444	24.4862882683983
5	112	35	Steeleye Stout	1	25.67	29.0594604444444	24.4862882683983

```
Wykonano bez błędów.
Wynik: Zwrócono 722780 wierszy w czasie 16927ms
W wierszu 1:
WITH PRODUCT_PRICES AS (
  SELECT
    ID,
    PRODUCTID,
    PRODUCTNAME,
    CATEGORYID,
    UNITPRICE,
    AVG(UNITPRICE) OVER (PARTITION BY CATEGORYID) AS AVG_PRICE,
    AVG(UNITPRICE) OVER () AS OVERALL_AVG
  FROM PRODUCT_HISTORY
)
SELECT *
FROM PRODUCT_PRICES
WHERE UNITPRICE > OVERALL_AVG;
```

2. z podzapytaniem
   - Czas wykonanie ponad 15min (zbyt długi czas oczekiwania)

SQL 1*

```
SELECT
  p.ID,
  p.CATEGORYID,
  p.PRODUCTID,
  p.PRODUCTNAME,
  p.UNITPRICE,
  (
    SELECT AVG(UNITPRICE)
    FROM PRODUCT_HISTORY q
    WHERE q.CATEGORYID = p.CATEGORYID
  ) AS AVG_PRICE
FROM PRODUCT_HISTORY p
WHERE p.UNITPRICE > (
    SELECT AVG(UNITPRICE)
    FROM PRODUCT_HISTORY
);
```

id	categoryid	productid	productname	unitprice	AVG_PRICE

Wyniki ostatnio wykonanych poleceń

3. z joinem
   - Czas wykonanie 6,59s

	id	categoryid	productid	productname	unitprice	AVG_PRICE
1	9	6	9	Mishi Kobe Niku	41.54	37.1026981111111
2	18	8	18	Carnarvon Tigers	30.32	20.3793001111111
3	20	3	20	Sir Rodney's Marmalade	36.34	22.6262894102564
4	28	7	28	Rössle Sauerkraut	24.83	26.2445674
5	29	6	29	Thüringer Rostbratwurst	50.25	37.1026981111111
6	38	1	38	Côte de Blaye	95.67	29.0594604444444

```
Wykonano bez błędów.
Wynik: Zwrócono 722780 wierszy w czasie 6590ms
W wierszu 1:
SELECT
  p.ID,
  p.CATEGORYID,
  p.PRODUCTID,
  p.PRODUCTNAME,
  p.UNITPRICE,
  q.AVG_PRICE
FROM PRODUCT_HISTORY p
INNER JOIN (
    SELECT CATEGORYID, AVG(UNITPRICE) AS AVG_PRICE
    FROM PRODUCT_HISTORY
    GROUP BY CATEGORYID
) q ON p.CATEGORYID = q.CATEGORYID
WHERE p.UNITPRICE > (
    SELECT AVG(UNITPRICE)
    FROM PRODUCT_HISTORY
);
```

Wnioski:

Funkcje okna umożliwiają obliczenie średniej w ramach kategorii w sposób prosty i czytelny. Wydajnościowo najlepsze okazało się rozwiązanie z użyciem JOIN. Podzapytania były najwolniejsze i najmniej czytelne.

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.

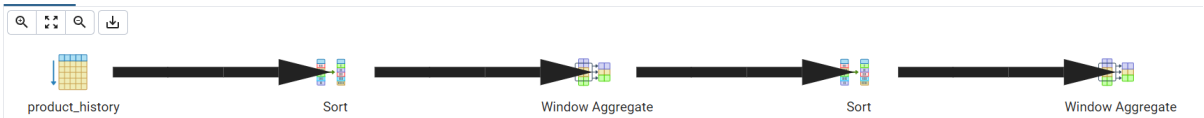
Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSql, SQLite)

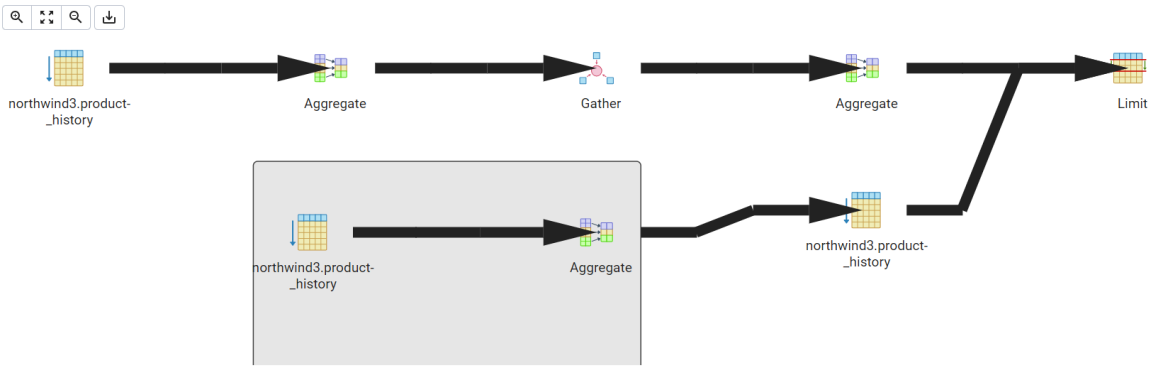
> Wyniki:

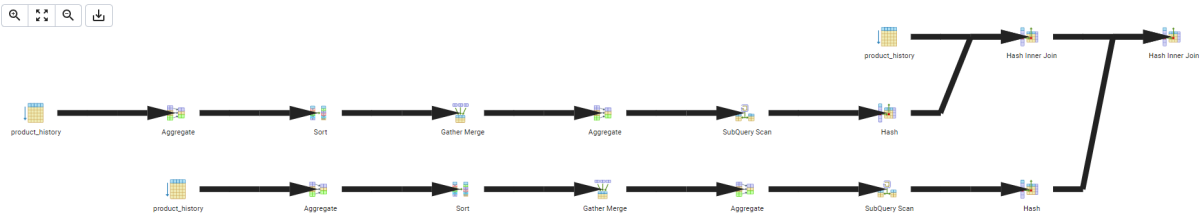
```
-- funkcje okna
SELECT ID
  , PRODUCTID
  , DATE
  , PRODUCTNAME
  , CATEGORYID
  , UNITPRICE
  , AVG(UNITPRICE) OVER category_win AS AVG_CATEGORY
  , SUM(VALUE) OVER category_win AS SUM_CATEGORY
  , AVG(UNITPRICE) OVER year_win AS AVG_YEAR
  , SUM(VALUE) OVER year_win AS SUM_YEAR
FROM Northwind3.PRODUCT_HISTORY
WINDOW category_win AS (PARTITION BY CATEGORYID),
       year_win AS (PARTITION BY DATE_PART('Year', DATE));

-- podzapytanie
SELECT ID
  , p.PRODUCTID
  , p.DATE
  , p.PRODUCTNAME
  , p.CATEGORYID
  , p.UNITPRICE
  , (SELECT AVG(UNITPRICE) WHERE CATEGORYID = p.CATEGORYID) AS AVG_CATEGORY
  , (SELECT SUM(VALUE) WHERE CATEGORYID = p.CATEGORYID) AS SUM_CATEGORY
  , (SELECT AVG(UNITPRICE) WHERE DATE_PART('Year', DATE) = DATE_PART('Year',
p.DATE)) AS AVG_DATE
  , (SELECT SUM(VALUE) WHERE DATE_PART('Year', DATE) = DATE_PART('Year', p.DATE))
AS SUM_DATE
FROM Northwind3.PRODUCT_HISTORY p;

-- join
SELECT ID
  , p.PRODUCTID
  , p.DATE
  , p.PRODUCTNAME
  , p.CATEGORYID
  , p.UNITPRICE
  , c.AVG_CATEGORY
  , c.SUM_CATEGORY
  , y.SUM_YEAR
  , y.AVG_YEAR
FROM Northwind3.PRODUCT_HISTORY p
INNER JOIN (SELECT  CATEGORYID
                  , AVG(UNITPRICE) AS AVG_CATEGORY
                  , SUM(VALUE) AS SUM_CATEGORY
                FROM Northwind3.PRODUCT_HISTORY
                GROUP BY CATEGORYID) c
ON c.CATEGORYID = p.CATEGORYID
```

```
INNER JOIN (SELECT  DATE_PART('Year', DATE) AS YEAR
                  , AVG(UNITPRICE) AS AVG_YEAR
                  , SUM(VALUE) AS SUM_YEAR
                FROM Northwind3.PRODUCT_HISTORY
                GROUP BY DATE_PART('Year', DATE)) y
ON DATE_PART('Year', p.DATE) = y.YEAR;
```

Wyniki PostgreSQL - wyniki limitowane do 500

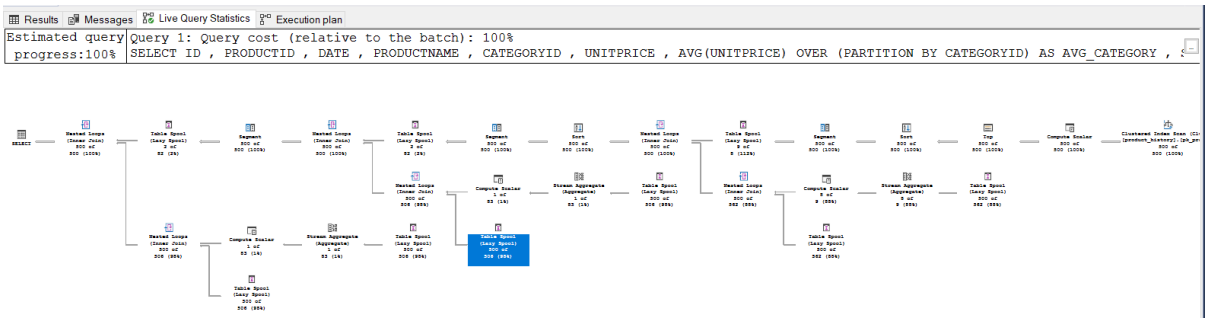
1. z funckją okna
   - Czas wykonania 5.5s
   - 

2. z podzapytaniem
   - Czas wykonanie 71s
   - 

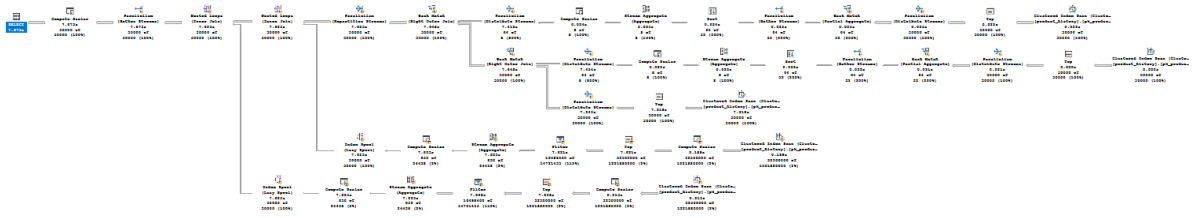
3. z joinem
   - Czas wykonanie 1s
   - 

Wyniki MsSQL - wyniki limitowane do 50tys

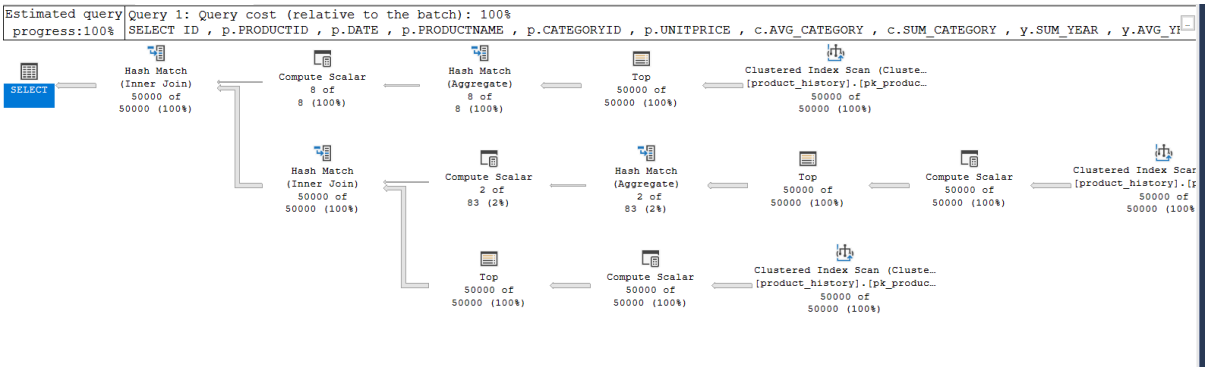
1. z funckją okna
   - Czas wykonania 1s



2. z podzapytaniem
    - Czas wykonanie 9.2s



3. z joinem
    - Czas wykonanie 345ms



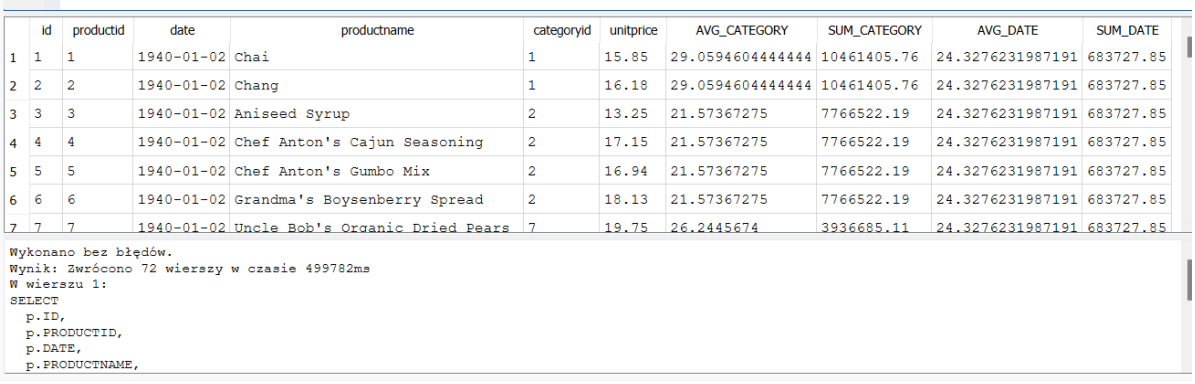
Wyniki SQlite

1. z funckją okna
    - Czas wykonania 33,5s

	id	productid	date	productname	categoryid	unitprice	AVG_CATEGORY	AVG_YEAR	SUM_CATEGORY	SUM_YEAR
1	1	1	1940-01-02	Chai	1	15.85	29.0594604444444	24.3276231987191	10461405.76	683727.85
2	2	2	1940-01-02	Chang	1	16.18	29.0594604444444	24.3276231987191	10461405.76	683727.85
3	24	24	1940-01-02	Guaraná Fantástica	1	11.46	29.0594604444444	24.3276231987191	10461405.76	683727.85
4	34	34	1940-01-02	Sasquatch Ale	1	14.55	29.0594604444444	24.3276231987191	10461405.76	683727.85
5	35	35	1940-01-02	Steeleye Stout	1	15.85	29.0594604444444	24.3276231987191	10461405.76	683727.85
6	38	38	1940-01-02	Côte de Blaye	1	95.67	29.0594604444444	24.3276231987191	10461405.76	683727.85
7	39	39	1940-01-02	Chartreuse verte	1	15.85	29.0594604444444	24.3276231987191	10461405.76	683727.85

```
Wykonano bez błędów.
Wynik: Zwrócono 2310000 wierszy w czasie 33502ms
W wierszu 1:
SELECT
  ID,
  PRODUCTID,
  DATE,
  PRODUCTNAME,
  CATEGORYID,
  UNITPRICE,
```

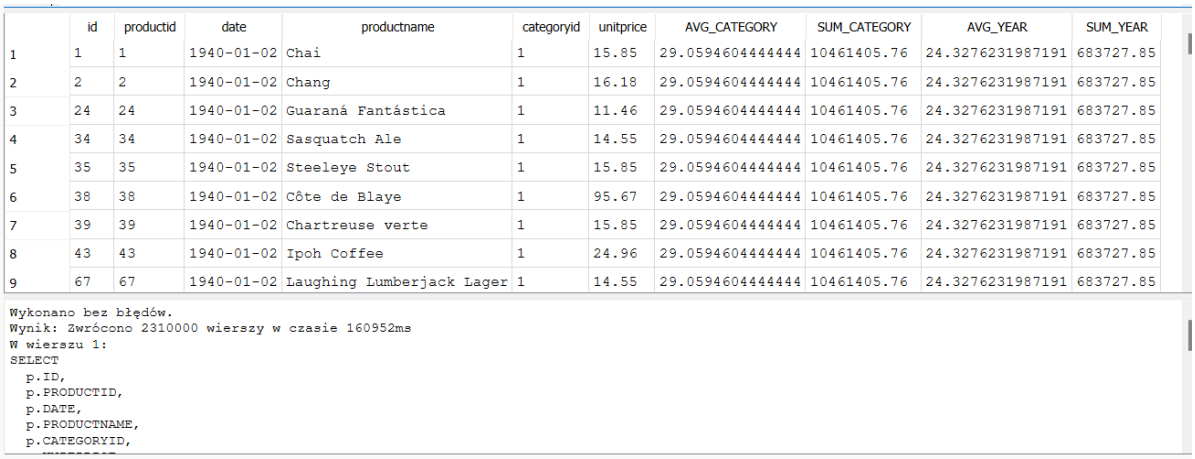
2. z podzapytaniem
    - Czas wykonanie 8min 19s



	id	productid	date	productname	categoryid	unitprice	AVG_CATEGORY	SUM_CATEGORY	AVG_DATE	SUM_DATE
1	1	1	1940-01-02	Chai	1	15.85	29.0594604444444	10461405.76	24.3276231987191	683727.85
2	2	2	1940-01-02	Chang	1	16.18	29.0594604444444	10461405.76	24.3276231987191	683727.85
3	3	3	1940-01-02	Aniseed Syrup	2	13.25	21.57367275	7766522.19	24.3276231987191	683727.85
4	4	4	1940-01-02	Chef Anton's Cajun Seasoning	2	17.15	21.57367275	7766522.19	24.3276231987191	683727.85
5	5	5	1940-01-02	Chef Anton's Gumbo Mix	2	16.94	21.57367275	7766522.19	24.3276231987191	683727.85
6	6	6	1940-01-02	Grandma's Boysenberry Spread	2	18.13	21.57367275	7766522.19	24.3276231987191	683727.85
7	7	7	1940-01-02	Uncle Bob's Organic Dried Pears	7	19.75	26.2445674	3936685.11	24.3276231987191	683727.85

```
Wykonano bez błędów.
Wynik: Zwrócono 72 wierszy w czasie 499782ms
W wierszu 1:
SELECT
  p.ID,
  p.PRODUCTID,
  p.DATE,
  p.PRODUCTNAME,
```

3. z joinem
   - Czas wykonanie 2min 41s



	id	productid	date	productname	categoryid	unitprice	AVG_CATEGORY	SUM_CATEGORY	AVG_YEAR	SUM_YEAR
1	1	1	1940-01-02	Chai	1	15.85	29.0594604444444	10461405.76	24.3276231987191	683727.85
2	2	2	1940-01-02	Chang	1	16.18	29.0594604444444	10461405.76	24.3276231987191	683727.85
3	24	24	1940-01-02	Guaraná Fantástica	1	11.46	29.0594604444444	10461405.76	24.3276231987191	683727.85
4	34	34	1940-01-02	Sasquatch Ale	1	14.55	29.0594604444444	10461405.76	24.3276231987191	683727.85
5	35	35	1940-01-02	Steeleye Stout	1	15.85	29.0594604444444	10461405.76	24.3276231987191	683727.85
6	38	38	1940-01-02	Côte de Blaye	1	95.67	29.0594604444444	10461405.76	24.3276231987191	683727.85
7	39	39	1940-01-02	Chartreuse verte	1	15.85	29.0594604444444	10461405.76	24.3276231987191	683727.85
8	43	43	1940-01-02	Ipoh Coffee	1	24.96	29.0594604444444	10461405.76	24.3276231987191	683727.85
9	67	67	1940-01-02	Laughing Lumberjack Lager	1	14.55	29.0594604444444	10461405.76	24.3276231987191	683727.85

```
Wykonano bez błędów.
Wynik: Zwrócono 2310000 wierszy w czasie 160952ms
W wierszu 1:
SELECT
  p.ID,
  p.PRODUCTID,
  p.DATE,
  p.PRODUCTNAME,
  p.CATEGORYID,
```

Wnioski:

To zadanie pokazuje moc funkcji okna przy analizie złożonych agregatów (średnia, suma) po różnych wymiarach. Zastosowanie WINDOW znacznie skraca kod. Wydajnościowo: JOIN działał najszybciej, funkcje okna — szybko i wygodnie, a podzapytania znowu były najwolniejsze.

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:

```sql
-- funkcje okna
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;
```

1. z funckją okna SQLLite:
    - Czas wykonania 4ms

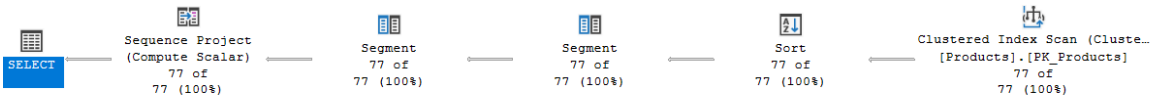
SQL 1*

```sql
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;
```

	ProductID	ProductName	UnitPrice	CategoryID	rowno	rankprice	denserankprice
1	38	Côte de Blaye	263.5	1	1	1	1
2	43	Ipoh Coffee	46	1	2	2	2
3	2	Chang	19	1	3	3	3
4	1	Chai	18	1	4	4	4
5	35	Steeleye Stout	18	1	5	4	4
6	39	Chartreuse verte	18	1	6	4	4
7	76	Lakkalikööri	18	1	7	4	4
8	70	Outback Lager	15	1	8	8	5
9	34	Sasquatch Ale	14	1	9	9	6
10	67	Laughing Lumberjack Lager	14	1	10	9	6
11	75	Rhönbräu Klosterbier	7.75	1	11	11	7
12	24	Guaraná Fantástica	4.5	1	12	12	8

```
Wykonano bez błędów.
Wynik: Zwrócono 77 wierszy w czasie 4ms
W wierszu 1:
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;
```

2. z funckją okna MsSQL:
    - Czas wykonania 55ms



3. z funckją okna PostgreSQL:
    - Czas wykonania 63ms

-

Zadanie

Spróbuj uzyskać ten sam wynik bez użycia funkcji okna

> Wyniki:

```
SELECT p1.PRODUCTID, p1.PRODUCTNAME, p1.UNITPRICE, p1.CATEGORYID,
       COUNT(*) 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
GROUP BY p1.PRODUCTID, p1.PRODUCTNAME, p1.UNITPRICE, p1.CATEGORYID
ORDER BY p1.CATEGORYID, p1.UNITPRICE DESC;
```

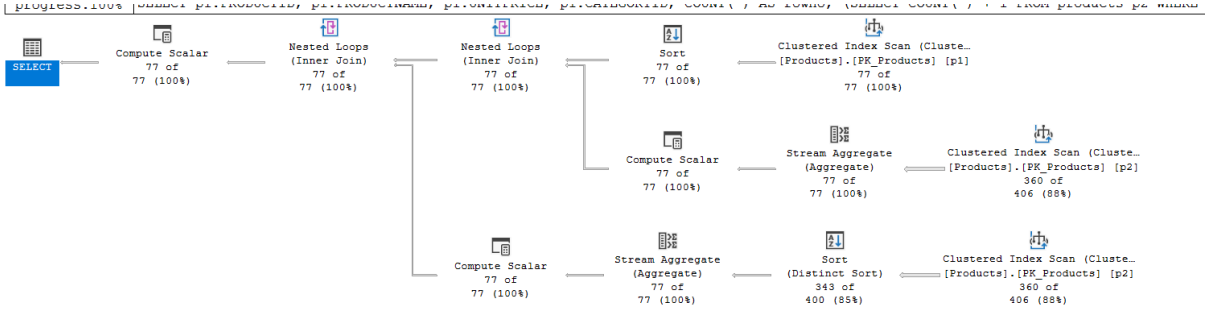
2. bez funckji okna SQLLite:
    - Czas wykonania 4ms

```sql
SELECT p1.PRODUCTID, p1.PRODUCTNAME, p1.UNITPRICE, p1.CATEGORYID,
       COUNT(*) 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
GROUP BY p1.PRODUCTID, p1.PRODUCTNAME, p1.UNITPRICE, p1.CATEGORYID
ORDER BY p1.CATEGORYID, p1.UNITPRICE DESC;
```

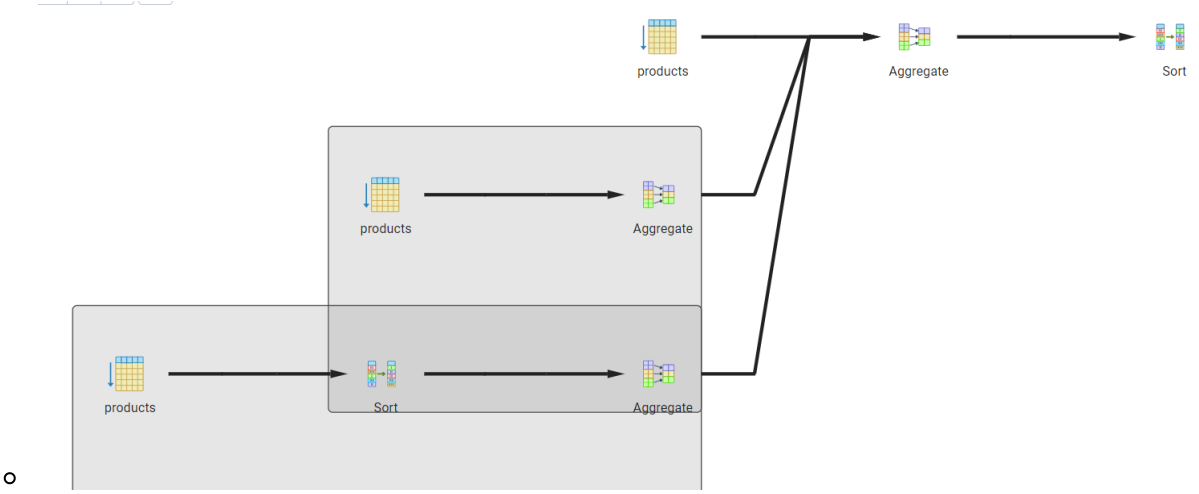
	ProductID	ProductName	UnitPrice	CategoryID	rowno	rankprice	denserankprice
1	38	Côte de Blaye	263.5	1	1	1	1
2	43	Ipoh Coffee	46	1	1	2	2
3	2	Chang	19	1	1	3	3
4	1	Chai	18	1	1	4	4
5	35	Steeleye Stout	18	1	1	4	4
6	39	Chartreuse verte	18	1	1	4	4
7	76	Lakkalikööri	18	1	1	4	4
8	70	Outback Lager	15	1	1	8	5
9	34	Sasquatch Ale	14	1	1	9	6
10	67	Laughing Lumberjack Lager	14	1	1	9	6
11	75	Rhönbräu Klosterbier	7.75	1	1	11	7
12	24	Guaraná Fantástica	4.5	1	1	12	8

```
Wykonano bez błędów.
Wynik: Zwrócono 77 wierszy w czasie 4ms
W wierszu 1:
SELECT p1.PRODUCTID, p1.PRODUCTNAME, p1.UNITPRICE, p1.CATEGORYID,
       COUNT(*) AS rowno,
```

3. bez funckji okna MsSQL:
   - Czas wykonania 74ms



4. bez funckji okna PostgreSQL:
   - Czas wykonania 4ms



Komentarz:

W zapytaniu porównano trzy funkcje rankujące:

1. ROW_NUMBER() – unikalna numeracja, nawet dla identycznych cen
2. RANK() – te same ceny dostają ten sam numer, ale są "przeskoki"
3. DENSE_RANK() – jak RANK(), ale bez przeskoków

Porównanie:

1. Funkcje okna: prostsze, szybsze, bardziej czytelne
2. Bez funkcji: działa, ale mniej wydajne i trudniejsze do modyfikacji

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 A AS
(SELECT DATE_PART('Year', DATE)
  , ID
```

```
  , PRODUCTNAME
  , UNITPRICE
  , RANK() OVER (PARTITION BY ID, DATE_PART('Year', DATE) ORDER BY UNITPRICE DESC)
AS R
  FROM Northwind3.PRODUCT_HISTORY
)
SELECT * FROM A
WHERE R < 5;

-- bez okna
SELECT ID
  , PRODUCTNAME
  , UNITPRICE
  , YEAR
  , RANKING
FROM (
  SELECT DATE_PART('Year', ph.DATE) AS YEAR
    , ph.ID,
    , ph.PRODUCTNAME,
    , ph.UNITPRICE,
    (
      SELECT COUNT(DISTINCT ph2.UNITPRICE)
      FROM Northwind3.PRODUCT_HISTORY ph2
        WHERE ph2.ID = ph.ID
        AND DATE_PART('Year', ph2.DATE) = DATE_PART('Year', ph.DATE)
        AND ph2.UNITPRICE > ph.UNITPRICE
    ) + 1 AS RANKING
  FROM Northwind3.PRODUCT_HISTORY ph
) AS ranked
WHERE RANKING <= 4
ORDER BY YEAR, ID, RANKING;
```

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)

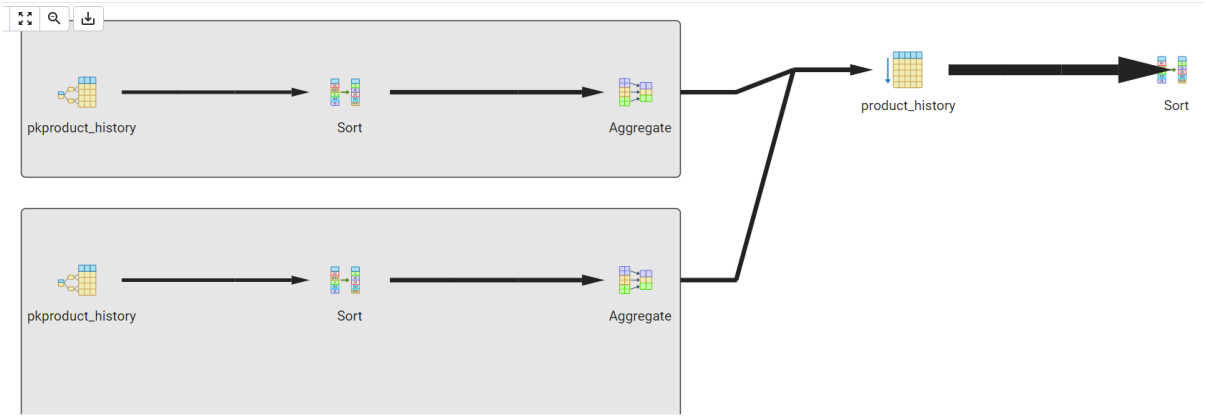
> Wyniki:

```
--  ...
```

Wyniki PostgreSQL

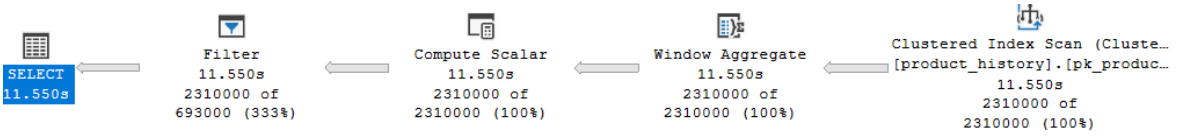
1. z funckją okna
    - Czas wykonania 2.1s

2. bez funkcji okna
   - Czas wykonanie 9.5s

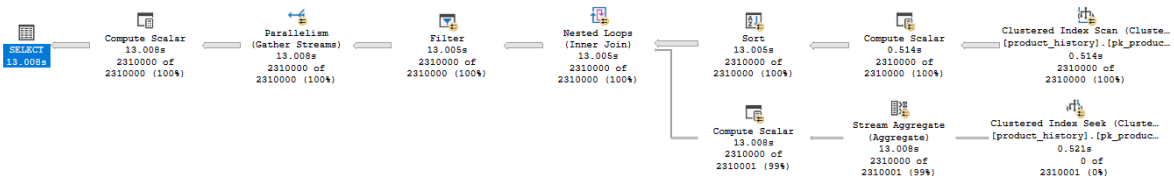


Wyniki MsSQL

1. z funckją okna
   - Czas wykonania 11.2s



2. bez funkcji okna
   - Czas wykonanie 13.8s



Wyniki SQLite

1. z funckją okna
   - Czas wykonania 14,86s

	YEAR	ID	PRODUCTNAME	UNITPRICE	R
1	1940	1	Chai	15.85	1
2	1940	2	Chang	16.18	1
3	1940	3	Aniseed Syrup	13.25	1
4	1940	4	Chef Anton's Cajun Seasoning	17.15	1
5	1940	5	Chef Anton's Gumbo Mix	16.94	1
6	1940	6	Grandma's Boysenberry Spread	18.13	1
7	1940	7	Uncle Bob's Organic Dried Pears	19.75	1

```
Wykonano bez błędów.
Wynik: Zwrócono 2310000 wierszy w czasie 14859ms
W wierszu 1:
WITH A AS (
  SELECT
    strftime('%Y', DATE) AS YEAR,
    ID,
    PRODUCTNAME,
    UNITPRICE,
    RANK() OVER (PARTITION BY ID, strftime('%Y', DATE) ORDER BY UNITP
  FROM PRODUCT_HISTORY
```

2. bez funkcji okna
   - Czas wykonanie ponad 15min (zbyt długi czas oczekiwania)

SQL 1*

```
SELECT
  ID,
  PRODUCTNAME,
  UNITPRICE,
  YEAR,
  RANKING
FROM (
  SELECT
    ph.ID,
    ph.PRODUCTNAME,
    ph.UNITPRICE,
```

ID	PRODUCTNAME	UNITPRICE	YEAR	RANKING

Wnioski:

Funkcja RANK() w połączeniu z PARTITION BY pozwala łatwo wybrać top-N rekordów w danej grupie (tu: ceny w danym roku). Bez funkcji okna zapytanie działa poprawnie, ale jest dużo bardziej złożone i mniej wydajne. W obu przypadkach wyniki są zgodne, ale czas wykonania wyraźnie lepszy przy zastosowaniu funkcji okna.

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:

```
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 strftime('%Y', date) = '2022'
ORDER BY date;
```

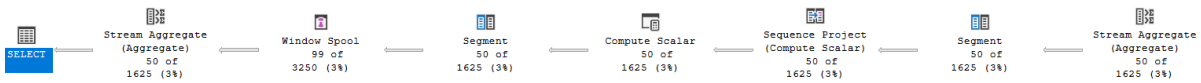
1. z funckją okna SQLLite:
    - Czas wykonania 397ms

```sql
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 strftime('%Y', 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
9	1	Chai	1	2022-01-09	26.73	17.12	19.5
10	1	Chai	1	2022-01-10	19.5	26.73	10.01

```
Wykonano bez błędów.
Wynik: Zwrócono 50 wierszy w czasie 397ms
W wierszu 1:
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 strftime('%Y', date) = '2022'
ORDER BY date;
```

2. z funckją okna MsSQL:
   - Czas wykonania 576ms



3. z funckją okna PostgreSQL:
   - Czas wykonania 608ms

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:

```sql
SELECT
  p.productid,
```

```
  p.productname,
  p.categoryid,
  p.date,
  p.unitprice,
  (
    SELECT ph2.unitprice
    FROM product_history ph2
    WHERE ph2.productid = p.productid
      AND ph2.date < p.date
      AND strftime('%Y', ph2.date) = '2022'
    ORDER BY ph2.date DESC
    LIMIT 1
  ) AS previousprodprice,
  (
    SELECT ph3.unitprice
    FROM product_history ph3
    WHERE ph3.productid = p.productid
      AND ph3.date > p.date
      AND strftime('%Y', ph3.date) = '2022'
    ORDER BY ph3.date ASC
    LIMIT 1
  ) AS nextprodprice

FROM product_history p
WHERE p.productid = 1 AND strftime('%Y', p.date) = '2022'
ORDER BY p.date;
```

---

1. bez funkji okna SQLLite:
    - Czas wykonania 43,3s

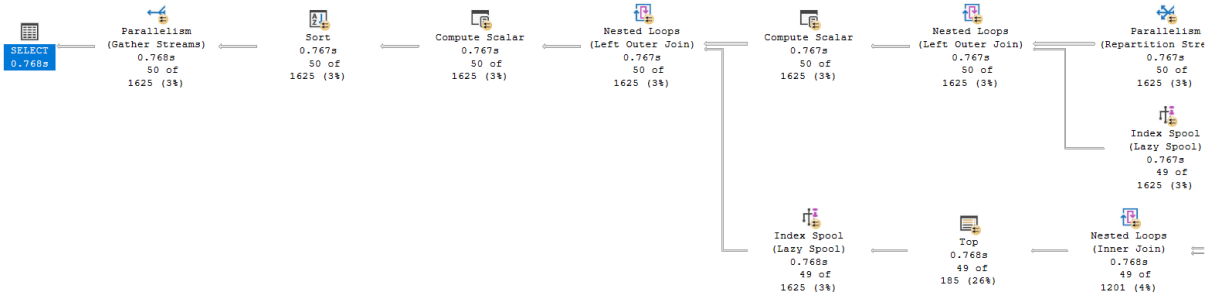
SQL 1*

```sql
SELECT
  p.productid,
  p.productname,
  p.categoryid,
  p.date,
  p.unitprice,
  (
    SELECT ph2.unitprice
    FROM product_history ph2
    WHERE ph2.productid = p.productid
      AND ph2.date < p.date
      AND strftime('%Y', ph2.date) = '2022'
    ORDER BY ph2.date DESC
    LIMIT 1
  ) AS previousprodprice,
  (
    SELECT ph3.unitprice
    FROM product_history ph3
    WHERE ph3.productid = p.productid
      AND ph3.date > p.date
```

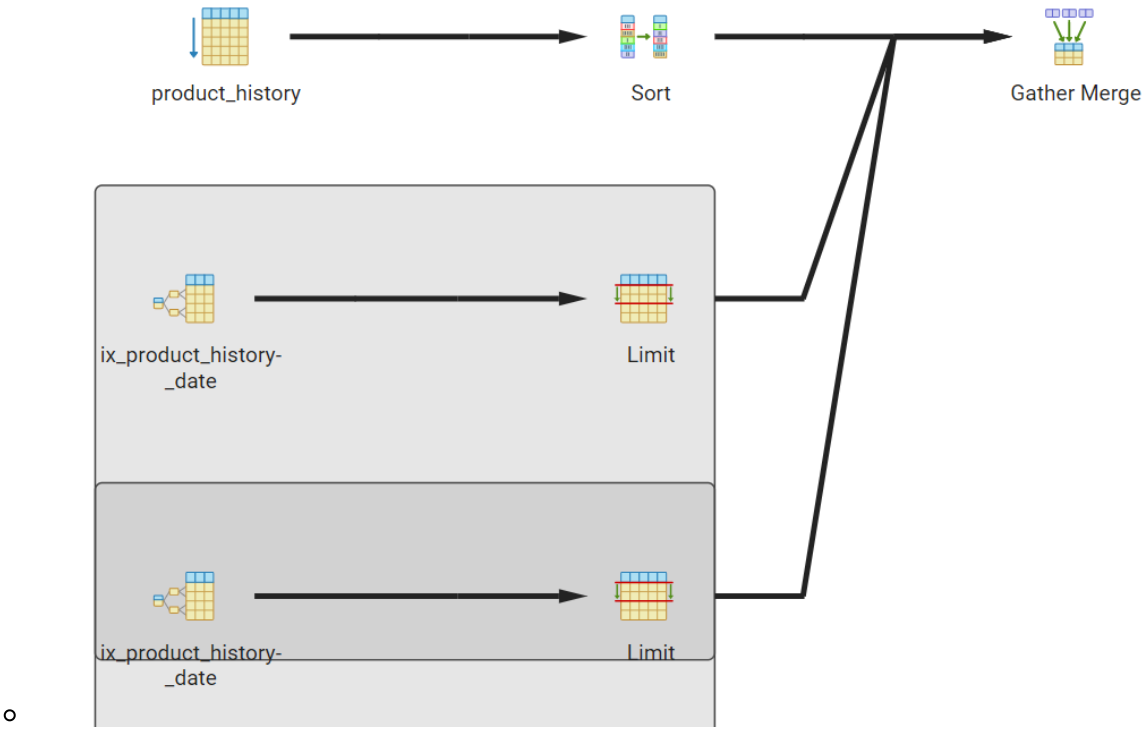
	productid	productname	categoryid	date	unitprice	previousprodprice	nextprodprice
1	1	Chai	1	2022-01-01	25.31	*NULL*	20.69
2	1	Chai	1	2022-01-02	20.69	25.31	18.16
3	1	Chai	1	2022-01-03	18.16	20.69	23.93
4	1	Chai	1	2022-01-04	23.93	18.16	13.69
5	1	Chai	1	2022-01-05	13.69	23.93	25.34
6	1	Chai	1	2022-01-06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

```
Wykonano bez błędów.
Wynik: Zwrócono 50 wierszy w czasie 43291ms
W wierszu 1:
SELECT
  p.productid,
```

2. bez funkji okna MsSQL:
   - Czas wykonania 0,7s



3. bez funkji okna PostgreSQL:
   - Czas wykonania 1,2s



Komentarz:

1. Funkcja `LAG()` zwraca wartość z poprzedniego wiersza.
2. Funkcja `LEAD()` z następnego — w określonym porządku, np. po dacie.

Bez funkji okna czas wykonania jest znacznie dłuższy.

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:

```
SELECT COMPANYNAME
  , DATE_PART('Year', ORDERDATE)
  , FREIGHT + (UNITPRICE * QUANTITY) * (1 - DISCOUNT) AS PRICE
  , LAG(o.ORDERID) OVER win
  , LAG(o.ORDERDATE) OVER win
```

```
  , LAG(FREIGHT + (UNITPRICE * QUANTITY) * (1 - DISCOUNT)) OVER win
FROM Northwind3.ORDERS o
INNER JOIN Northwind3.ORDERDETAILS od
ON o.ORDERID = od.ORDERID
INNER JOIN NOrthwind3.CUSTOMERS c
ON c.CUSTOMERID = o.CUSTOMERID
WINDOW win as (PARTITION BY o.CUSTOMERID ORDER BY DATE);
```

Komentarz:

- funkcja okna LAG zwraca dane dotyczące poprzedniego rekordu (albo NULL jesli on nie istnieje) wedlug danego sortowania. Ważne jest zeby pamietac o rozmiarze domyslnego okna
- wyżej wymieniona funkcja znacząco poprawia łatwość implementacji oraz efektywnośc samego zapytania

---

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;

-- bez funkcji okna
SELECT
    p.productid,
    p.productname,
    p.unitprice,
    p.categoryid,
    (SELECT p2.productname
     FROM Northwind3.products p2
     WHERE p2.categoryid = p.categoryid
     ORDER BY p2.unitprice DESC
     LIMIT 1) AS first,
    (SELECT p2.productname
     FROM Northwind3.products p2
```

```
      WHERE p2.categoryid = p.categoryid
      ORDER BY p2.unitprice ASC
      LIMIT 1) AS last
FROM Northwind3.products p
ORDER BY p.categoryid, p.unitprice DESC;
```

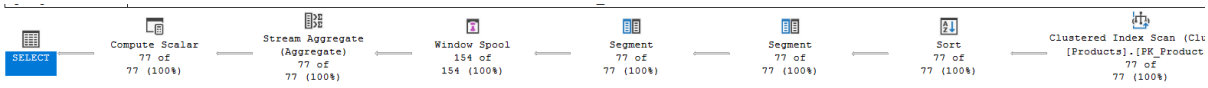
> Wyniki:

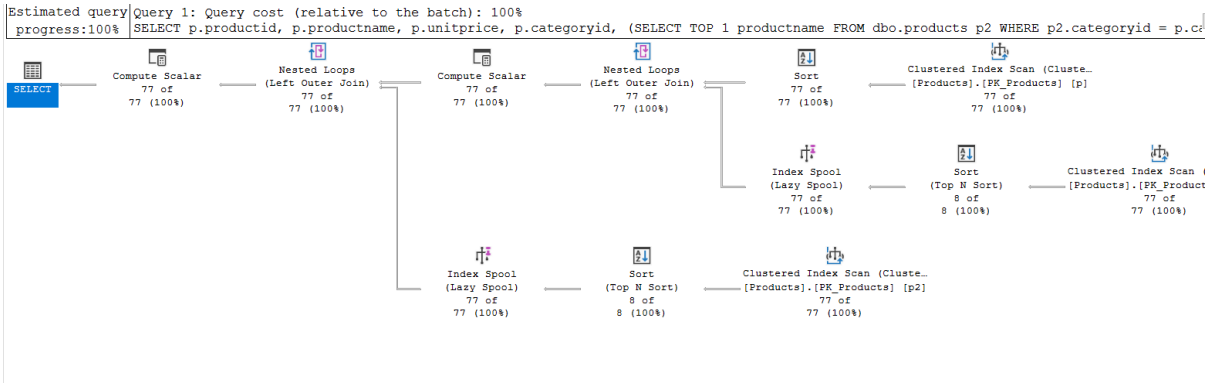
W zwróconym wyniku dla każdego rekordu zwracana jest najwieksza cena w danej kategorii, ale nie najmniejsza, ponieważ jak jest używana klauzula *ORDER BY* to domyślny zakres okna jest od pierwszego rekordu do tego w którym jest rekord którego dotyczy, tak więc jak posortujemy rosnąco to ta największa wartość znajdzie się w każdym oknie, ale najmniejsza już niekoniecznie (albo prawie na pewno nie jeśli nie mówimy o najtanszym produkcie z kategorii). Podsumowując dane zapytanie zwraca zawsze najdrozszy produkt w danej kategorii i zwykle siebie jako najtanszy (zwykle, bo moze być kilka o tej samej cenie)

Komentarz

Na podstawie pomiarów możemy stwierdzić, że w tym przypadku funkcje okna są porównywalnia efektywne z innymi rozwiązaniami. Jednakże zapytanie jest dużo łatwiejsze do napisania oraz czytania

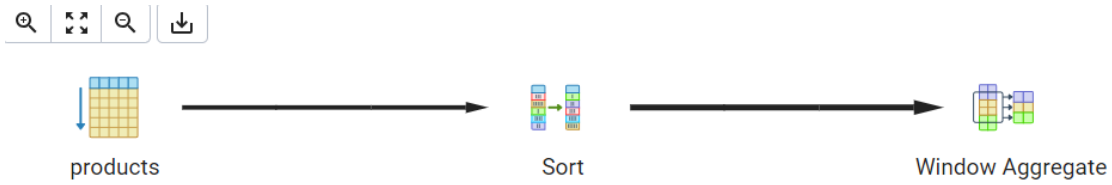
Wyniki MsSQL

1. z funckją okna
   - Czas wykonania 61ms
   - 

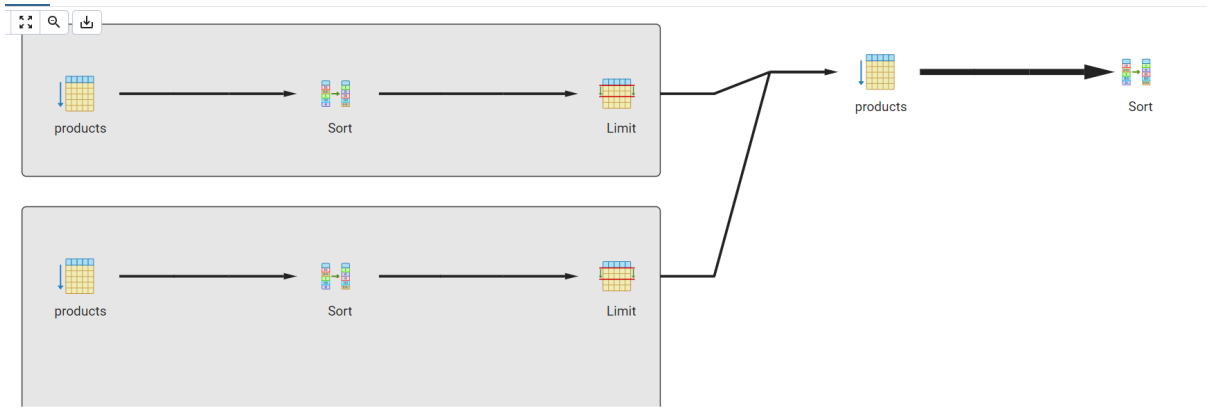
2. bez funkcji okna
   - Czas wykonanie 78ms
   - 

Wyniki PostgreSQL

1. z funckją okna
   - Czas wykonania 111ms



2. bez funkcji okna
   - Czas wykonanie 68ms



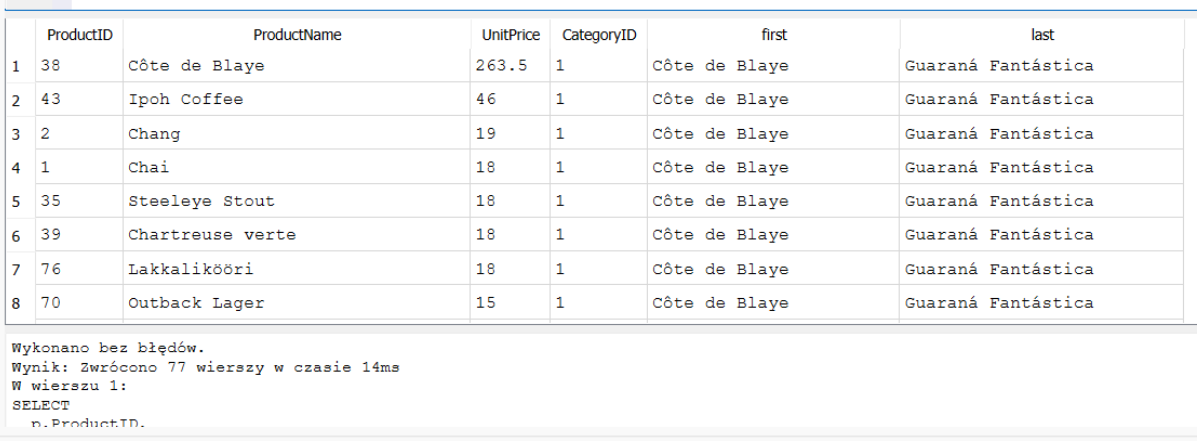
- Wyniki SQLite

1. z funckją okna
   - Czas wykonania 7ms

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

```
Wykonano bez błędów.
Wynik: Zwrócono 77 wierszy w czasie 7ms
W wierszu 1:
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;
```

2. bez funkcji okna
   - Czas wykonanie 14ms



	ProductID	ProductName	UnitPrice	CategoryID	first	last
1	38	Côte de Blaye	263.5	1	Côte de Blaye	Guaraná Fantástica
2	43	Ipoh Coffee	46	1	Côte de Blaye	Guaraná Fantástica
3	2	Chang	19	1	Côte de Blaye	Guaraná Fantástica
4	1	Chai	18	1	Côte de Blaye	Guaraná Fantástica
5	35	Steeleye Stout	18	1	Côte de Blaye	Guaraná Fantástica
6	39	Chartreuse verte	18	1	Côte de Blaye	Guaraná Fantástica
7	76	Lakkalikööri	18	1	Côte de Blaye	Guaraná Fantástica
8	70	Outback Lager	15	1	Côte de Blaye	Guaraná Fantástica

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:

Zadanie 8

Baza: Northwind, tabele orders, order details

Napisz polecenie które wyświetla inf. o zamówieniach

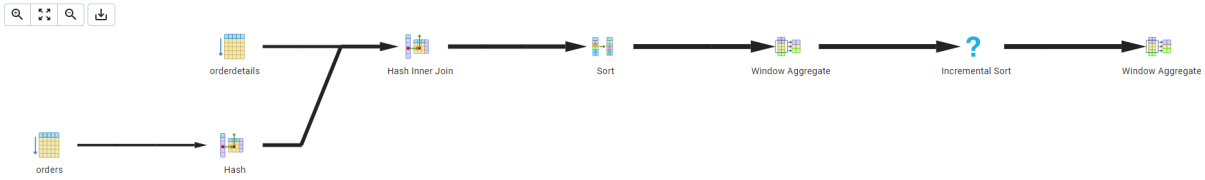
Zbiór wynikowy powinien zawierać:

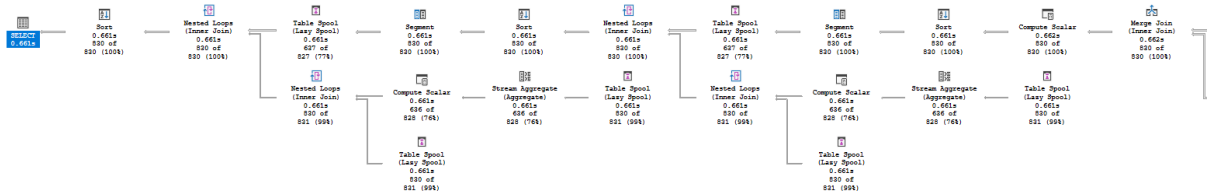
- Id klienta,
- nr zamówienia,
- datę zamówienia,
- wartość zamówienia (wraz z opłatą za przesyłkę),
- dane zamówienia klienta o najniższej wartości w danym miesiącu
  - nr zamówienia o najniższej wartości w danym miesiącu
  - datę tego zamówienia
  - wartość tego zamówienia
- dane zamówienia klienta o najwyższej wartości w danym miesiącu
  - nr zamówienia o najniższej wartości w danym miesiącu
  - datę tego zamówienia
  - wartość tego zamówienia

> Wyniki:

```
SELECT CUSTOMERID
      , o.ORDERID
      , ORDERDATE
      , FREIGHT + (UNITPRICE * QUANTITY * (1 - DISCOUNT)) AS TOTAL_COST
      , FIRST_VALUE(o.ORDERID) OVER WIN_MONTH_ASC AS MIN_ID
      , FIRST_VALUE(ORDERDATE) OVER WIN_MONTH_ASC AS MIN_DATE
      , FIRST_VALUE(FREIGHT + (UNITPRICE * QUANTITY * (1 - DISCOUNT))) OVER
WIN_MONTH_ASC AS MIN_VALUE
      , FIRST_VALUE(o.ORDERID) OVER WIN_MONTH_DESC AS MAX_ID
      , FIRST_VALUE(ORDERDATE) OVER WIN_MONTH_DESC AS MAX_DATE
      , FIRST_VALUE(FREIGHT + (UNITPRICE * QUANTITY * (1 - DISCOUNT))) OVER
WIN_MONTH_DESC AS MAX_VALUE
FROM Northwind3.ORDERDETAILS od
INNER JOIN Northwind3.ORDERS o
   ON o.ORDERID = od.ORDERID
WINDOW
WIN_MONTH_ASC AS (PARTITION BY CUSTOMERID
                            , DATE_PART('year', ORDERDATE)
                            , DATE_PART('month', ORDERDATE)
                  ORDER BY FREIGHT + (UNITPRICE * QUANTITY * (1 - DISCOUNT)) ASC)
, WIN_MONTH_DESC AS (PARTITION BY CUSTOMERID
                            , DATE_PART('year', ORDERDATE)
                            , DATE_PART('month', ORDERDATE)
                  ORDER BY FREIGHT + (UNITPRICE * QUANTITY * (1 - DISCOUNT))
DESC);
```

Wyniki

1. PostgreSQL:
   - 87ms
   - 

2. MsSQL
   - 110ms
   - 
3. SQLite:
   - 531ms

	CustomerID	OrderID	OrderDate	Total_Cost	Min_ID	Min_Date	Min_Value	Max_ID	Max_Date	Max_Value
1	ALFKI	10692	1997-03-10	939.02	10692	1997-03-10	939.02	10692	1997-03-10	939.02
2	ALFKI	10643	1997-08-25	542.46	10643	1997-08-25	47.46	10643	1997-08-25	542.46
3	ALFKI	10643	1997-08-25	312.96	10643	1997-08-25	47.46	10643	1997-08-25	542.46
4	ALFKI	10643	1997-08-25	47.46	10643	1997-08-25	47.46	10643	1997-08-25	542.46
5	ALFKI	10702	1997-10-13	83.94	10702	1997-10-13	83.94	10702	1997-10-13	293.94
6	ALFKI	10702	1997-10-13	293.94	10702	1997-10-13	83.94	10702	1997-10-13	293.94
7	ALFKI	10835	1998-01-15	894.53	10835	1998-01-15	90.33	10835	1998-01-15	894.53

```
Wykonano bez błędów.
Wynik: Zwrócono 2155 wierszy w czasie 531ms
W wierszu 1:
SELECT
  o.CustomerID,
  o.OrderID,
  o.OrderDate,
```

Komentarz

- przy odpowiednik ustawieniu okna wraz z sortowaniem funkcje first i last value pozwalają na znalezienie największego i najmniejszego elementu w danej kategorii

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 ID
   , PRODUCTID
   , DATE
   , SUM(VALUE) OVER (PARTITION BY PRODUCTID, DATE) AS PRODUCT_DATE_VALUE
   , SUM(VALUE) OVER (PARTITION BY PRODUCTID
                              , DATE_PART('year', DATE)
                              , DATE_PART('month', DATE)
                      ORDER BY DATE ASC)
FROM Northwind3.PRODUCT_HISTORY ORDER BY PRODUCTID, DATE;

-- bez okna
WITH DAILY AS (
    SELECT
          PRODUCTID
```

```
        , DATE
        , SUM(VALUE) AS PRODUCT_DATE_VALUE
    FROM Northwind3.PRODUCT_HISTORY
    GROUP BY PRODUCTID, DATE
)
SELECT
      ph.ID
    , ph.PRODUCTID
    , ph.DATE
    , dps.PRODUCT_DATE_VALUE,
    (
        SELECT SUM(ph_inner.VALUE)
        FROM Northwind3.PRODUCT_HISTORY ph_inner
        WHERE ph_inner.PRODUCTID = ph.PRODUCTID
          AND DATE_PART('year', ph_inner.DATE) = DATE_PART('year', ph.DATE)
          AND DATE_PART('month', ph_inner.DATE) = DATE_PART('month', ph.DATE)
          AND ph_inner.DATE <= ph.DATE
    ) AS monthly_cumulative_value
FROM Northwind3.PRODUCT_HISTORY ph
INNER JOIN DAILY dps
    ON ph.PRODUCTID = dps.PRODUCTID AND ph.DATE = dps.DATE
ORDER BY ph.PRODUCTID, ph.DATE;
```

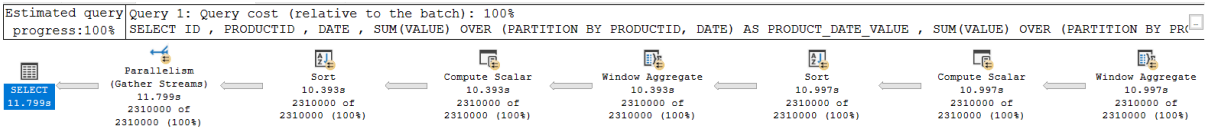
Spróbuj wykonać zadanie bez użycia funkcji okna. Spróbuj uzyskać ten sam wynik bez użycia funkcji okna, porównaj wyniki, czasy i plany zapytań. Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSql, SQLite)

> Wyniki:

```
--  ...
```

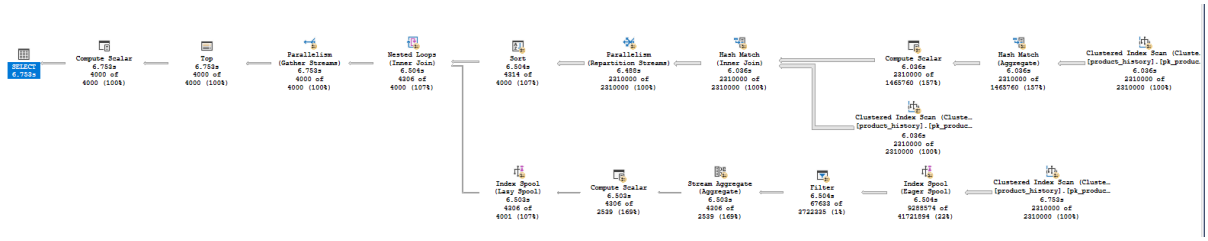
Wyniki MsSQL

1. z funckją okna
   - Czas wykonania 11,8s



   -

2. bez funkcji okna
   - Czas wykonanie 6,7 [DLA 2000]

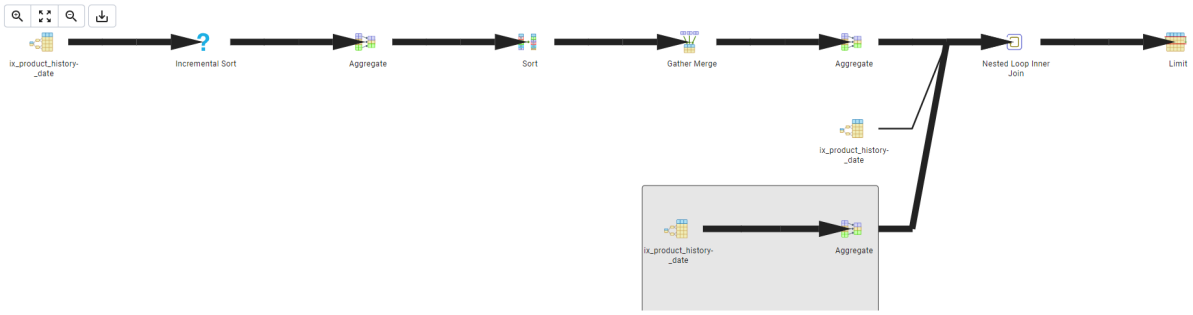


   -

Wyniki PostgreSQL

1. z funckją okna
   - Czas wykonania 5,8s

   -
2. bez funkcji okna
   - Czas wykonanie 20s [DLA 2000]



   -

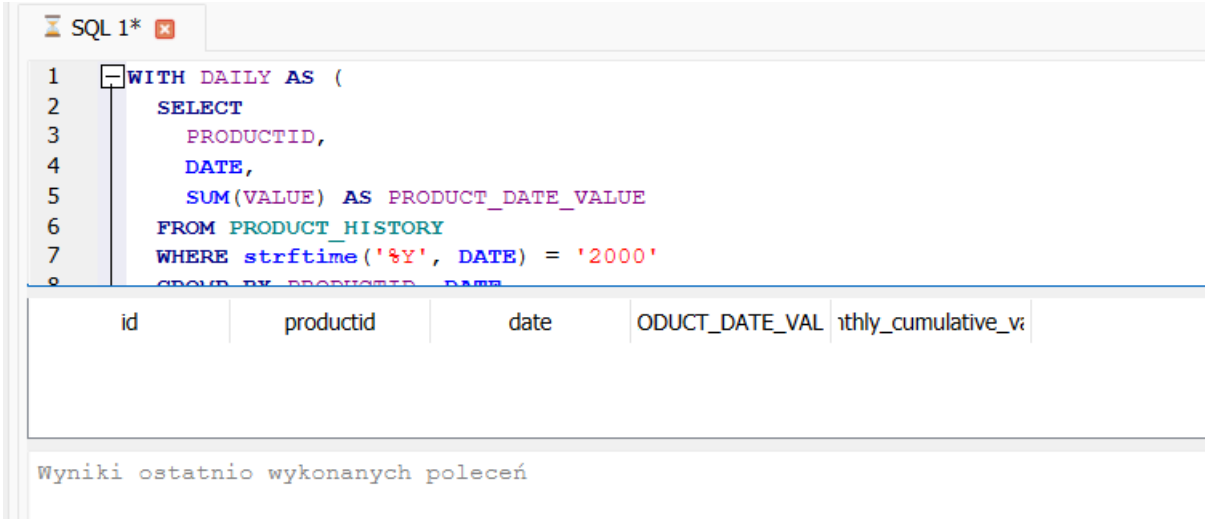
Wyniki SQLite

1. z funkcją okna
   - Czas wykonania 34,37s

	id	productid	date	PRODUCT_DATE_VALUE	PRODUCT_MONTH_VALUE
1	1	1	1940-01-02	158.5	158.5
2	78	1	1940-01-03	256.7	415.2
3	155	1	1940-01-04	274.2	689.4
4	232	1	1940-01-05	140.3	829.7
5	309	1	1940-01-06	265.8	1095.5
6	386	1	1940-01-07	202.5	1298.0

```
Wykonano bez błędów.
Wynik: Zwrócono 2310000 wierszy w czasie 34367ms
W wierszu 1:
SELECT
  ID,
  PRODUCTID,
  DATE,

  -- suma wartości w danym dniu dla produktu
```

   -
2. bez funkcji okna
   - Czas wykonanie ponad 15min (zbyt długi czas oczekiwania)



Komentarz

- w tym zastosowaniu funkcje okna zdeklasowały resztę rozwiązań pod względem czytelności i efektywności

---

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:

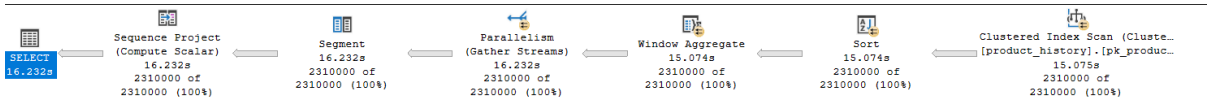
```
--  nieuzywana funkcja NTILE
-- podajemy dodatkowo kwantyl
WITH PRODUCT_PRICES AS
    (SELECT ID
        , PRODUCTID
        , PRODUCTNAME
        , CATEGORYID
        , UNITPRICE
        , NTILE(4) OVER (PARTITION BY CATEGORYID ORDER BY VALUE) AS AVG_PRICE
    FROM Northwind3.PRODUCT_HISTORY)
SELECT * FROM PRODUCT_PRICES;

-- to samo co wyzej tylko bez okna

SELECT
      ph.ID
    , ph.PRODUCTID
    , ph.PRODUCTNAME
    , ph.CATEGORYID
    , ph.UNITPRICE
    FLOOR((((SELECT COUNT(*)
                FROM dbo.PRODUCT_HISTORY ph2
```

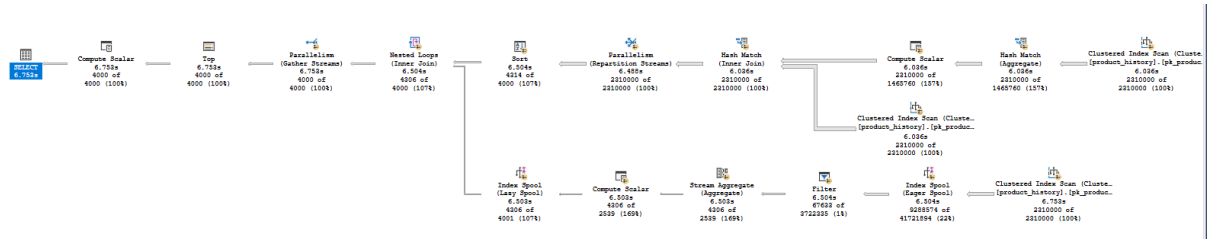
```
                WHERE ph2.CATEGORYID = ph.CATEGORYID
                    AND ph2.UNITPRICE < ph.UNITPRICE) * 4.0
                ) /
                (SELECT COUNT(*)
                 FROM dbo.PRODUCT_HISTORY ph3
                 WHERE ph3.CATEGORYID = ph.CATEGORYID)
            ) ) + 1 AS AVG_PRICE
FROM dbo.PRODUCT_HISTORY ph;


-- czy efektywnie jest wybierać więcej niż jedną kolumne podzapytaniem a potem
splitować ?
SELECT ID
  , p.PRODUCTID
  , p.DATE
  , p.PRODUCTNAME
  , p.CATEGORYID
  , p.UNITPRICE
  , (SELECT AVG(UNITPRICE) WHERE CATEGORYID = p.CATEGORYID) AS AVG_CATEGORY
  , (SELECT SUM(VALUE) WHERE CATEGORYID = p.CATEGORYID) AS SUM_CATEGORY
  , (SELECT AVG(UNITPRICE) WHERE DATE_PART('Year', DATE) = DATE_PART('Year',
p.DATE)) AS AVG_DATE
  , (SELECT SUM(VALUE) WHERE DATE_PART('Year', DATE) = DATE_PART('Year', p.DATE))
AS SUM_DATE
FROM Northwind3.PRODUCT_HISTORY p;
-- vs
WITH T AS
   (SELECT ID
   , p.PRODUCTID
   , p.DATE
   , p.PRODUCTNAME
   , p.CATEGORYID
   , p.UNITPRICE
   , (SELECT AVG(UNITPRICE)|| '|' || SUM(VALUE) WHERE CATEGORYID = p.CATEGORYID)
AS STATS_CATEGORY
   , (SELECT AVG(UNITPRICE)|| '|' || SUM(VALUE) WHERE DATE_PART('Year', DATE) =
DATE_PART('Year', p.DATE)) AS STATS_DATE
   FROM Northwind3.PRODUCT_HISTORY p)
SELECT ID
   , PRODUCTID
   , DATE
   , PRODUCTNAME
   , CATEGORYID
   , UNITPRICE
   , SPLIT_PART(STATS_CATEGORY, '|', 1)
   , SPLIT_PART(STATS_CATEGORY, '|', 2)
   , SPLIT_PART(STATS_DATE, '|', 1)
   , SPLIT_PART(STATS_DATE, '|', 2)
FROM T;
```

Wyniki

1

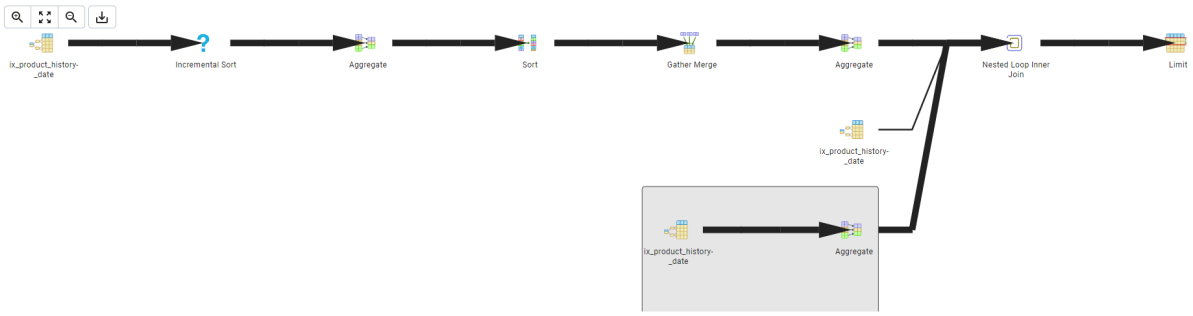
Wyniki MsSQL

1. z funckją okna
   - Czas wykonania 16.3s
   

   -
2. bez funkcji okna
   - Czas wykonanie 12s [DLA 1000]
   
   -

Wyniki PostgreSQL

1. z funckją okna
   - Czas wykonania 5,8s
   

   -
2. bez funkcji okna
   - Czas wykonanie > 100s [DLA 1000]
   

   -

2 Niestety nie, liczyło się ponad 5razy dłużej i nie skończyło

Wnioski z całego labolatorium

Poszczególne wnioski z zadan są zamieszczone bezposrednio pod nimi

- wykorzystywanie funkcji okna jest bardzo wygodne oraz czytelne
- w większości wypadków również pod względem efektywności są porównywalne bądź lepsze

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