

SQL - Funkcje okna (Window functions) - Lab 1-2

Imię i Nazwisko: Daniel Kuc

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 **czzerwone pola**. Preferowane są zrzuty ekranu, **wymagane** komentarze, ale dołącz też polecenia formie tekstowej

Oprogramowanie - co jest potrzebne?

Do wykonania ćwiczenia potrzebne jest następujące oprogramowanie:

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

Oprogramowanie dostępne jest na przygotowanej maszynie wirtualnej

Dokumentacja/Literatura

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

Kilka linków do materiałów które mogą być pomocne

- <https://learn.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql?view=sql-server-ver16>
- <https://www.sqlservertutorial.net/sql-server-window-functions/>
- <https://www.sqlshack.com/use-window-functions-sql-server/>
- <https://www.postgresql.org/docs/current/tutorial-window.html>

- <https://www.postgresqltutorial.com/postgresql-window-function/>
- <https://www.sqlite.org/windowfunctions.html>
- <https://www.sqlitetutorial.net/sqlite-window-functions/>

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

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

Przygotowanie

Uruchom SSMS.

- Skonfiguruj połączenie z bazą Northwind na lokalnym serwerze MS SQL

Uruchom DataGrip (lub Dbeaver)

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

Zadanie 1 - obserwacja

Wykonaj i porównaj wyniki następujących poleceń.

```
-- 1
select avg(unitprice) avgprice
from products p;
-- 2
select avg(unitprice) over () as avgprice
from products p;

-- 3
select categoryid, avg(unitprice) avgprice
from products p
group by categoryid
-- 4
select avg(unitprice) over (partition by categoryid) as avgprice
from products p;
```

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

Wynikiem 1 polecenia jest pojedyncza komórka a wynikiem 2 polecenia jest tabela, której komórki są równe wartości średniej ceny dla produktów w tabeli Products.

W poleceniu 3 jak i 4 dostajemy średnią wartość produktów w danej kategorii tylko w poleceniu trzecim dostajemy wynik "zgrupowany" to jest tabela zawiera pojedyncze rekordy z category id oraz średnią ceną w przeciwieństwie do polecenia 4 które każdemu produktowi przypisuje średnią cenę z kategorii do której należy i w rezultacie dostajemy 77 rekordów w tabeli wynikowej a nie 8 jak w poleceniu 3.

Zadanie 2 - obserwacja

Wykonaj i porównaj wyniki następujących poleceń.

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

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

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

W zapytaniu 1, średnia wartość jest liczona poprzez podzapytanie, które pod uwagę bierze całą tabelę products a dopiero później ogranicza tabelę wynikową do produktów, których id jest mniejsze 10. Polecenie 2 z funkcją okna działa w taki sposób, że najpierw ogranicza tabelę do produktów o id mniejszym od 10 i dopiero wtedy 'pracuje' na tej tabeli i wylicza wartość średnią.

Polecenie równoważne 1) z funkcją okna:

```
with temp as (  
    select ProductID, ProductName, UnitPrice, avg(unitprice) over () as avgprice  
    from Products  
)  
SELECT * from temp WHERE ProductID < 10;
```

Polecenie równoważne 2) z podzapytaniem:

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

Zadanie 3

Baza: Northwind, tabela: products

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

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

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

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

SQLQuery1.sql - GA...nd3 (GABI\rm (62))*

```

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

```

176 %

Results Messages Execution plan

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

select p.productid, p.ProductName, p.unitprice, (select avg(unitprice) from products) as

```

graph LR
    SELECT[SELECT  
Cost: 0 %] --> CS1[Compute Scalar  
Cost: 0 %]
    CS1 --> NL[Nested Loops  
(Inner Join)  
Cost: 5 %  
0.000s  
77 of  
77 (100%)]
    NL --> CS2[Compute Scalar  
Cost: 0 %]
    NL --> CIA[Clustered Index Scan (Clustered)  
[Products].[PK_Products] [p]  
Cost: 47 %  
0.000s  
77 of  
77 (100%)]
    CS2 --> SA[Stream Aggregate  
(Aggregate)  
Cost: 1 %  
0.000s  
1 of  
1 (100%)]

```

W DataGrip użyj opcji Explain Plan/Explain Analyze

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

Services

Tx: > Output Plan >

Operation

- Select
 - Value (Corr)
 - Nested
 - Value
 - Aggre
 - Full Ir

Context Menu:

- Show Context Actions
- Paste ⌘V
- Copy / Paste Special >
- Column Selection Mode ⌘8
- Refactor >
- Folding >
- Save as Live Template...
- Reformat Code ⌘L
- Go To >
- Generate... ⌘N
- Run '.l.sql' ^⌘R
- More Run/Debug >
- Switch Session (w)
- Explain Plan** >
- Execute ⌘↵
- Execute to File
- Open In >

Explain Plan Sub-menu:

- Explain Plan
- Explain Plan (Raw)
- Explain Analyse**
- Explain Analyse (Raw)

	Rows	Act
	77	
	77	
	77	77
	1	
	1	1
[PK_Products];	77	77
[PK_Products];	77	77

```

-- 3 okno
SELECT P.ProductID,
       P.ProductName,
       P.UnitPrice,
       avg(P.UnitPrice) over () AS AvgPrice
FROM Products AS P;

-- 3 podzapytanie
SELECT P.ProductID,
       P.ProductName,
       P.UnitPrice,
       (SELECT avg(UnitPrice) FROM Products) AS AvgPrice
FROM Products AS P;

-- 3 join
WITH temp AS (
  SELECT ProductID, (SELECT avg(UnitPrice) FROM Products) as AVG
  FROM Products AS P
)
SELECT Products.ProductID, ProductName, UnitPrice, temp.AVG
FROM Products
JOIN temp on temp.ProductID = Products.ProductID;

```

Czasy dla Microsoft Servera:

okno - 175 ms

podzapytanie - 167ms

join - 160 ms

Czasy dla Postgresa:

okno - 89 ms

podzapytanie - 84 ms

join - 98 ms

Czasy dla SQLite:

okno - 252 ms

podzapytanie - 270 ms

join - 189 ms

Z analizy planu wynika że jest ono najbardziej rozbudowane dla rozwiązania z joinem (poniważ wymaga zadeklarowania dodatkowej tabeli przy pomocy klauzuli with) co obrazuje również total cost, który znacząco ostaje od pozostałych wartości tj. dla joina ma wartość 0.016 a przykładowo dla funkcji okna ma wartość 0.0038 chodź trzeba wziąć poprawkę że tabela products nie jest zbyt duża.

Zadanie 4

Baza: Northwind, tabela products

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

Napisz polecenie z wykorzystaniem podzapytania, join'a oraz funkcji okna. Porównaj

zapytania. Porównaj czasy oraz plany wykonania zapytań.
Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSQL, SQLite)


```

-- 4 okno
WITH temp as (
SELECT
    P.ProductID,
    P.ProductName,
    P.UnitPrice,
    P.CategoryID,
    avg(UnitPrice) over(partition by CategoryID) AS AvgPrice
FROM Products AS P
)
SELECT * FROM temp
WHERE UnitPrice > AvgPrice
ORDER BY temp.ProductID;

-- 4 podzapytanie
WITH temp as (
SELECT P.ProductID,
    P.ProductName,
    P.UnitPrice,
    P.CategoryID,
    (SELECT avg(UnitPrice)
     FROM Products
     WHERE P.CategoryID = Products.CategoryID
     GROUP BY CategoryID) AS AvgPrice
FROM Products AS P
)
SELECT * FROM temp
WHERE UnitPrice > AvgPrice
ORDER BY temp.ProductID;

-- 4 join
WITH temp as (
SELECT P.CategoryID,
    avg(P.UnitPrice) AS AvgPrice
FROM Products AS P
GROUP BY P.CategoryID
)
SELECT ProductID, ProductName, UnitPrice, Products.CategoryID, temp.AvgPrice
FROM Products
JOIN temp on Products.CategoryID = temp.CategoryID
WHERE UnitPrice > temp.AvgPrice
ORDER BY ProductID;

```

Czasy dla Microsoft Servera:

okno - 193 ms

podzapytanie - 200 ms

join - 215 ms

Czasy dla Postgresa:

okno - 158 ms

podzapytanie - 214 ms

join - 201 ms

Czasy dla SQLite:

okno - 97 ms

podzapytanie - 106 ms

join - 118 ms

Plany zapytań dla powyższych poleceń co do budowy są praktycznie identyczne, obrazują też to całkowite koszty który co do niewielkich wartości jest niemalże identyczny dla każdego polecenia.

Zadanie 5 - przygotowanie

Baza: Northwind

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

Wygeneruj tabelę zawierającą kilka milionów (kilkaset tys.) wierszy

Stwórz tabelę o następującej strukturze:

Skrypt dla SQL Server

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

Wygeneruj przykładowe dane:

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

Skrypt dla SQL Server

```

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

2310000

Zadanie 6

Baza: Northwind, tabela product_history

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

Napisz polecenie, które zwraca: id pozycji, id produktu, nazwę produktu, cenę produktu, średnią cenę produktów w kategorii do której należy dany produkt.

Wyświetl tylko pozycje (produkty) których cena jest większa niż średnia cena.

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

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

```

-- 6 okno
WITH temp as (
SELECT
    P.ProductID,
    P.ProductName,
    P.UnitPrice,
    P.CategoryID,
    avg(UnitPrice) over(partition by CategoryID) AS AvgPrice
FROM product_history AS P
)
SELECT * FROM temp
WHERE UnitPrice > AvgPrice
ORDER BY temp.ProductID;

-- 6 podzapytanie
WITH temp as (
SELECT P.ProductID,
    P.ProductName,
    P.UnitPrice,
    P.CategoryID,
    (SELECT avg(UnitPrice)
     FROM product_history
     WHERE P.CategoryID = product_history.CategoryID
     GROUP BY CategoryID) AS AvgPrice
FROM product_history AS P
)
SELECT * FROM temp
WHERE UnitPrice > AvgPrice
ORDER BY temp.ProductID;

-- 6 join
WITH temp as (
SELECT P.CategoryID,
    avg(P.UnitPrice) AS AvgPrice
FROM product_history AS P
GROUP BY P.CategoryID
)
SELECT ProductID, ProductName, UnitPrice, product_history.CategoryID, temp.AvgPrice
FROM product_history
JOIN temp on product_history.CategoryID = temp.CategoryID
WHERE UnitPrice > temp.AvgPrice
ORDER BY ProductID;

```

Czasy dla Microsoft Servera:

okno - 9 s

podzapytanie - 8 s

join - 5 s

Czasy dla Postgresa:

okno - 4 s

podzapytanie - Podałem się po minucie czekania :/

join - 3 s

Czasy dla SQLite:

okno - 6 s

podzapytanie - Podałem się po minucie czekania :/

join - 5 s

Plany zapytań poleceń z podzapytaniem jak i joinem są dosyć podobne jak i ich koszty tylko lekko odstaje polecenie za pomocą funkcji okna co możemy zaobserwować dla Microsoft Servera. W pozostałych serwerach bazodanowych podzapytanie staje się ZUPEŁNIE nie efektywne, gdzie rozwiązanie za pomocą funkcji okna jak i joina działa podobnie dobrze z lekką przewagą rozwiązania z joinem.

Nasuwa się również ciekawy wniosek ponieważ dane w funkcji okna są przetwarzane w "jednym

Zadanie 7

Baza: Northwind, tabela product_history

Lekka modyfikacja poprzedniego zadania

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

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

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

Porównaj czasy oraz plany wykonania zapytań.

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

```

-- 7 okno
SELECT P.id,
       P.ProductID,
       P.ProductName,
       P.UnitPrice,
       P.CategoryID,
       avg(UnitPrice) over(partition by categoryid) AS AvgPrice,
       sum(value) over (partition by categoryid) AS SumValue,
       avg(unitprice) over (partition by productid, YEAR(date)) as YearAvgPrice,
       sum(unitprice) over (partition by productid, YEAR(date)) as YearSumValue
FROM product_history AS P

-- 7 podzapytanie
SELECT P.id,
       P.ProductID,
       P.ProductName,
       P.UnitPrice,
       P.CategoryID,
       (SELECT avg(unitprice)
        FROM product_history
        WHERE product_history.categoryid = P.categoryid
        GROUP BY categoryid) AS AvgPrice,
       (SELECT sum(unitprice)
        FROM product_history
        WHERE product_history.categoryid = P.categoryid
        GROUP BY categoryid) AS SumValue,
       (SELECT avg(unitprice)
        FROM product_history
        WHERE product_history.categoryid = P.categoryid
        AND YEAR(product_history.date) = YEAR(P.date)
        GROUP BY categoryid) AS YearAvgPrice,
       (SELECT sum(unitprice)
        FROM product_history
        WHERE product_history.categoryid = P.categoryid
        AND YEAR(product_history.date) = YEAR(P.date)
        GROUP BY categoryid) AS YearSumValue
FROM product_history AS P;

-- 7 join
SELECT P.id,
       P.ProductID,
       P.ProductName,
       P.UnitPrice,
       P.CategoryID,
       P1.AvgPrice,
       P1.SumValue,
       P2.YearAvgPrice,
       P2.YearSumValue
FROM product_history AS P
JOIN (SELECT product_history.categoryid, avg(unitprice) as AvgPrice, sum(unitprice) AS SumValue
      FROM product_history
      GROUP BY categoryid) as P1 on P.categoryid = P1.categoryid
JOIN (SELECT productid, YEAR(date) as date, avg(unitprice) as YearAvgPrice, sum(unitprice) AS YearSumValue
      FROM product_history
      GROUP BY productid, YEAR(date)) as P2 on P.productid = P2.productid AND P2.date = YEAR(P.date)

```

Czasy dla Microsoft Servera:

okno - 18 s

podzapytanie - Podałem się po minucie czekania :/

join - 16 s

Czasy dla Postgresa: (Modyfikacja dla polecenia zamiast funkcji YEAR:

```
SELECT date_part('year', date) from product_history as t WHERE P.id = t.id )
```

okno - 18 s

podzapytanie - Podałem się po minucie czekania :/

join - 19 s

15

Czasy dla SQLite: (Modyfikacja dla polecenia zamiast funkcji YEAR:

```
strftime('%Y', date))
```

okno - 12 s

podzapytanie - Podałem się po minucie czekania :/

Zadanie 8 - obserwacja

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

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

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

ROW_NUMBER(): funkcja ta przypisuje unikalny numer do każdego wiersza w grupie, zgodnie z określonym porządkiem sortowania.

RANK(): funkcja ta przypisuje ranking do każdego wiersza w grupie, zgodnie z określonym porządkiem sortowania, przy czym równe wartości otrzymują ten sam ranking, a kolejny ranking jest pomijany.

DENSE_RANK(): funkcja ta przypisuje ranking do każdego wiersza w grupie, zgodnie z określonym porządkiem sortowania, przy czym równe wartości otrzymują ten sam ranking, a kolejny ranking jest przypisywany.

Zadanie

Spróbuj uzyskać ten sam wynik bez użycia funkcji okna


```

-- 8a)
SELECT P1.ProductID, P1.ProductName, P1.CategoryID, P1.UnitPrice,
       (SELECT count(*)
        FROM Products AS P2
        WHERE P2.ProductID <= P1.ProductID AND P2.CategoryID = P1.CategoryID) as row_number
FROM Products AS P1
ORDER BY p1.categoryid, p1.unitprice DESC;

-- 8b)
SELECT P1.ProductID, P1.ProductName, P1.UnitPrice, P1.CategoryID,
       (SELECT COUNT(*) + 1
        FROM Products AS P2
        WHERE P2.UnitPrice > P1.UnitPrice AND P2.CategoryID = P1.CategoryID) as rankprice
FROM Products AS P1
ORDER BY P1.CategoryID, P1.UnitPrice DESC;

-- 8c)
SELECT P1.ProductID, P1.ProductName, P1.UnitPrice, P1.CategoryID,
       (SELECT COUNT(DISTINCT P2.UnitPrice)
        FROM Products AS P2
        WHERE P2.CategoryID = P1.CategoryID AND P2.UnitPrice > P1.UnitPrice) + 1 as rankprice
FROM Products AS P1
ORDER BY P1.CategoryID, P1.UnitPrice DESC;

```

Zadanie 9

Baza: Northwind, tabela product_history

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

rok

id produktu

nazwę produktu

cenę

datę (datę uzyskania przez produkt takiej ceny)

pozycję w rankingu

Uporządkuj wynik wg roku, nr produktu, pozycji w rankingu

```

WITH temp AS (
    SELECT YEAR(p.date) as year, p.productid, p.productname, p.unitprice, p.date,
           rank() over (partition by productid, YEAR(p.date) order by p.unitprice DESC) as rankprice
    FROM product_history as p
)
SELECT *
FROM temp
WHERE temp.rankprice <= 4;

```

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

```

WITH temp AS (SELECT P1.ProductID,
                    P1.ProductName,
                    P1.UnitPrice,
                    P1.CategoryID,
                    (SELECT COUNT(*) + 1
                     FROM product_history AS P2
                     WHERE P2.UnitPrice > P1.UnitPrice
                     AND P2.ProductID = P1.ProductID
                     AND YEAR(P2.date) = YEAR(P1.date)) as rankprice
                FROM product_history AS P1
            )
SELECT *
FROM temp
WHERE temp.rankprice <= 4
ORDER BY temp.productid, temp.unitprice DESC;

```

Nie doczekałem się wyniku polecenia powyżej, gdzie zapytanie z funkcją okna wykonuje się w 1,5 - 3 sekundy w zależności od serwera. Widocznie mnogość podzapytań dobija serwery i nie zwraca wyniku w sensownym czasie.

Zadanie 10 - obserwacja

Funkcje lag(), lead()

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

```

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

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

```

LEAD(): funkcja ta zwraca wartość wiersza, znajdującego się po bieżącym wierszu w określonej kolejności sortowania.

LAG(): funkcja ta zwraca wartość wiersza, znajdującego się przed bieżącym wierszem w określonej kolejności sortowania.

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)

-- Zapytanie dla Microsoft Servera

```
SELECT p.productid,  
       p.productname,  
       p.categoryid,  
       p.date,  
       p.unitprice,  
       (SELECT p2.unitprice  
        FROM product_history as p2  
        WHERE p2.productid = p.productid  
              AND DATEADD(day, +1, p2.date) = p.date  
              AND YEAR(p2.date) = 2022) as previousprodprice,  
       (SELECT p2.unitprice  
        FROM product_history as p2  
        WHERE p2.productid = p.productid  
              AND DATEADD(day, -1, p2.date) = p.date  
              AND YEAR(p2.date) = 2022) as nextprodprice  
FROM product_history as p  
WHERE productid = 1 AND YEAR(date) = 2022  
ORDER BY p.date;
```

-- Zapytanie dla Postgresa

```
SELECT p.productid,  
       p.productname,  
       p.categoryid,  
       p.date,  
       p.unitprice,  
       (SELECT p2.unitprice  
        FROM product_history as p2  
        WHERE p2.productid = p.productid  
              AND (SELECT p2.date + integer '1') = p.date  
              AND extract(YEAR FROM p2.date) = 2022) as previousprodprice,  
       (SELECT p2.unitprice  
        FROM product_history as p2  
        WHERE p2.productid = p.productid  
              AND (SELECT p2.date + integer '-1') = p.date  
              AND extract(YEAR FROM p2.date) = 2022) as nextprodprice  
FROM product_history as p  
WHERE productid = 1 AND extract(YEAR FROM date) = 2022  
ORDER BY p.date;
```

-- Zapytanie dla SQLite

```
SELECT p.productid,  
       p.productname,  
       p.categoryid,  
       p.date,  
       p.unitprice,  
       (SELECT p2.unitprice  
        FROM product_history as p2  
        WHERE p2.productid = p.productid  
              AND (SELECT DATE(p2.date, '+1 days')) = p.date  
              AND (SELECT strftime('%Y', p2.date)) = strftime('%Y', '2022-01-01')) as previousprodprice,  
       (SELECT p2.unitprice  
        FROM product_history as p2  
        WHERE p2.productid = p.productid  
              AND (SELECT DATE(p2.date, '-1 days')) = p.date  
              AND (SELECT strftime('%Y', p2.date)) = strftime('%Y', '2022-01-01')) as nextprodprice  
FROM product_history as p  
WHERE productid = 1 AND strftime('%Y', p.date) = strftime('%Y', '2022-01-01')  
ORDER BY p.date;
```

Czasy dla Microsoft Servera:

okno - 370 ms

bez funkcji okna - 13s

Czasy dla Postgresa:

okno - 790 ms

bez funkcji okna - 1m 14s

Czasy dla SQLite:

okno - 400 ms

bez funkcji okna - 33 s

Wnioski:

Przepaść w czasie wykonania dla funkcji okna jest ogromna, w szczególności dla postgresa. Można to również zaobserwować na planach zapytań, które znów obrazują, że dla funkcji okna przetwarzane są w "jednym kroku" a dla podzapytań te same dane są "mielone" kilka razy.

Warte zauważenie też jest jak dobrze działają podzapytania w Microsoft Serverze w sotsunku do pozostałych serwerów bazodanowych. Taka tendencja powtarza się już w kilku zadaniach.

Zadanie 11

Baza: Northwind, tabele customers, orders, order details

Napisz polecenie które wyświetla inf. o zamówieniach

Zbiór wynikowy powinien zawierać:

nazwę klienta, nr zamówienia,

datę zamówienia,

wartość zamówienia (wraz z opłatą za przesyłkę),

nr poprzedniego zamówienia danego klienta,

datę poprzedniego zamówienia danego klienta,

wartość poprzedniego zamówienia danego klienta.

```
WITH temp AS (
  SELECT C.CompanyName,
         O.OrderID,
         O.OrderDate,
         C.CustomerID,
         (SUM(OD.UnitPrice * OD.Quantity * (1 - OD.Discount)) + (SELECT Freight
                                                                FROM Orders
                                                                WHERE Orders.OrderID = O.OrderID)) as Value
  FROM Customers AS C
  JOIN Orders AS O on C.CustomerID = O.CustomerID
  JOIN [Order Details] AS OD on O.OrderID = OD.OrderID
  GROUP BY C.CompanyName, O.OrderID, O.OrderDate, C.CustomerID
)
SELECT *,
       lag(OrderID) over ( partition by CustomerID order by OrderDate) as PreviousOrderID,
       lag(OrderDate) over ( partition by CustomerID order by OrderDate) as PreviousOrderDate,
       lag(Value) over ( partition by CustomerID order by OrderDate) as PreviousOrderValue
FROM temp;
```

Zadanie 12 - obserwacja

Funkcje first_value(), last_value()

Wykonaj polecenia, zaobserwuj wynik. Jak działają funkcje first_value(), last_value().

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

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

FIRST_VALUE(): funkcja ta zwraca pierwszą wartość w grupie.

LAST_VALUE(): funkcja ta zwraca ostatnią wartość w grupie.

Funkcja first_value wskazuje w tym przypadku najdroższy produkt w danej kategorii tak jak się spodziewaliśmy, jednak funkcja last_value nie działa tak jak oczekujemy, ponieważ wynika to z domyślnych wartości range w funkcji okna, które są ustawione na: od początku do obecnego wiersza (tłumacząc z ang). Aby uzyskać najtańszy produkt w funkcji last_value musimy zmodyfikować zakres range'a tj.

```
last_value(productname) over (partition by categoryid order by unitprice desc range between unbounded preceding and unbounded following)
```

Drugim sposobem na poradzenie sobie z tym problem to użycie funkcji first_value oraz zmiana porządku z malejącego na rosnący.

Zadanie

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

```
SELECT
  P.productid,
  P.productname,
  P.unitprice,
  P.categoryid,
  (SELECT TOP 1 productname
   FROM Products
   WHERE CategoryID = P.CategoryID
   ORDER BY UnitPrice DESC ) as first,
  (SELECT TOP 1 productname
   FROM Products
   WHERE CategoryID = P.CategoryID AND UnitPrice = P.UnitPrice
   ORDER BY ProductID DESC ) as last
FROM Products AS P
ORDER BY CategoryID, UnitPrice DESC;
```

Czas dla Microsoft Servera: 150 ms

Czas dla Postgresa: 110 ms (zmiana TOP 1 na limit 1)

Czas dla SQLite: 112 ms (zmiana TOP 1 na limit 1)

Bardzo zbliżone dla powyższego rozwiązania jak i rozwiązania z funkcją okna ale tabela Products nie jest zbyt miarodajna, ponieważ ma stosunkowo małą liczbę rekordów.

Zadanie 13

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

```
WITH temp AS (
  SELECT O.CustomerID,
         O.OrderID,
         O.OrderDate,
         (SUM(OD.UnitPrice * OD.Quantity * (1 - OD.Discount)) + (SELECT Freight
                                                                FROM Orders
                                                                WHERE Orders.OrderID = O.OrderID)) as Value
  FROM Orders AS O
  JOIN [Order Details] AS OD on O.OrderID = OD.OrderID
  GROUP BY O.CustomerID, O.OrderID, O.OrderDate
)
SELECT *,
       first_value(OrderID) over (partition by CustomerID ORDER BY Value) as IdLowestValueOrder,
       first_value(OrderDate) over (partition by CustomerID ORDER BY Value) as DateLowestValueOrder,
       first_value(Value) over (partition by CustomerID ORDER BY Value) as ValueLowestValueOrder,
       first_value(OrderID) over (partition by CustomerID ORDER BY Value DESC) as IdHighestValueOrder,
       first_value(OrderDate) over (partition by CustomerID ORDER BY Value DESC) as DateHighestValueOrder,
       first_value(Value) over (partition by CustomerID ORDER BY Value DESC) as ValueHighestValueOrder
FROM temp;
```

Zadanie 14

Baza: Northwind, tabela product_history

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

Zbiór wynikowy powinien zawierać:

id pozycji

id produktu

datę

wartość sprzedaży produktu w danym dniu

wartość sprzedaży produktu narastające od początku miesiąca

```
SELECT ph.id,
       ph.productid,
       ph.date,
       ph.value,
       sum(value) over (partition by productid, YEAR(date), MONTH(date) order by date) as GrownValue
FROM product_history as ph
ORDER BY productid, 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)

```

SELECT ph.id,
       ph.productid,
       ph.date,
       ph.value,
       (SELECT sum(value)
        FROM product_history
        WHERE date <= ph.date
          AND YEAR(date) = YEAR(ph.date)
          AND MONTH(date) = MONTH(ph.date)
          AND productid = ph.productid
        GROUP BY productid) as GrownValue
FROM product_history as ph
ORDER BY productid, date;

```

Czas dla Microsoft Servera:

- z funkcją okna: 1m 10s
- powyższe rozwiązanie: po 5 min brak rezultatu :/

Czas dla Postgresa:

- z funkcją okna: 8s
- powyższe rozwiązanie: po 5 min brak rezultatu :/

Czas dla SQLite:

- z funkcją okna: 13s
- powyższe rozwiązanie: po 5 min brak rezultatu :/

Tutaj możemy zaobserwować dość ciekawy rezultat. Po pierwsze rozwiązanie z podzapytaniami jest wysoce nieefektywne jak i ciekawe zjawisko występuje dla funkcji okna, która w Microsoft Serverze liczy się około 6/7 krotnie dłużej niż w pozostałych serwerach. Wykonywałem polecenie kilka razy na różnych łączach oraz analizowałem plany zapytań ale wciąż nie mogę dojść do wniosku czemu akurat dla sumowania jest taka duża różnica.

Zadanie 15

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

Modyfikacja zadania 12, gdzie zamiast pierwszej wartości zwracamy wartość 2. Idealnie nadaje się do tego funkcja okna NTH_Value ale niestety wersja serwera microsoft na którym pracuje jest mniejsza niż 19 przez co musiałem zbudować tą funkcję przy pomocy row_number.

Funkcja okna:

```
with temp as (  
select row_number() over(partition by categoryid order by unitprice desc) as 'row', ProductName, CategoryID  
from Products  
)  
SELECT P.productid,  
       P.ProductName,  
       P.UnitPrice,  
       P.CategoryID,  
       (SELECT ProductName FROM temp as t WHERE t.CategoryID = P.CategoryID AND t.row = 2) as second  
FROM Products AS P  
ORDER BY CategoryID, UnitPrice DESC;
```

Bez funkcji okna:

```
SELECT  
  P.productid,  
  P.productname,  
  P.unitprice,  
  P.categoryid,  
  (SELECT TOP 1 ProductName  
   FROM (SELECT TOP 2 productname, UnitPrice  
         FROM Products  
         WHERE CategoryID = P.CategoryID  
         ORDER BY UnitPrice DESC ) as pUP  
   ORDER BY UnitPrice) as second  
FROM Products AS P  
ORDER BY CategoryID, UnitPrice DESC;
```

Z racji tego, że tabela products posiada małą ilość wierszy to czasy są mocno zbliżone dla każdego z serwera jak i polecenia. Różnice dopiero możemy zaobserwować na planie, który jest nieco 'bogatszy' ze względu na liczbę podzapytań, które "miały" te same dane dla drugiego rozwiązania. Różnica byłaby jeszcze wydatniejsza jeżeli w pierwszym rozwiązaniu użylibyśmy wspomnianej funkcji NTH_Value.

NTH_VALUE(expression, N): funkcja ta zwraca wartość określonego wiersza w grupie, zgodnie z określoną kolejnością sortowania.

WNIOSKI PO CAŁYM LABOLATORIUM:

Efektywność zapytania: Zapytania z użyciem funkcji okna są zazwyczaj bardziej efektywne niż równoważne zapytania z użyciem podzapytań lub join, ponieważ funkcje okna przetwarzają dane tylko raz, podczas gdy zapytania z podzapytaniami i join muszą wykonywać te same obliczenia wiele razy, co prowadzi do większego obciążenia systemu.

Wydażność przetwarzania danych: Funkcje okna pozwalają na efektywniejsze przetwarzanie danych niż inne metody, ponieważ dane są przetwarzane w jednym kroku, co pozwala na minimalizację liczby przetwarzanych rekordów.

Wielkość danych: W przypadku dużych zbiorów danych, zapytania z użyciem funkcji okna są zazwyczaj szybsze niż zapytania z podzapytaniami lub join, ponieważ funkcje okna przetwarzają tylko niezbędne dane, co pozwala na znaczne zmniejszenie czasu przetwarzania.

Punktacja

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