SQL - Funkcje okna (Window functions)

Lab 1-2

Imię i nazwisko:

- **Szymon Budziak**
- **Piotr Ludynia**

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

Swoje odpowiedzi wpisuj w miejsca oznaczone jako:

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

Ważne/wymagane są komentarze.

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

Zwróć uwagę na formatowanie kodu

Oprogramowanie - co jest potrzebne?

Do wykonania ćwiczenia potrzebne jest następujące oprogramowanie:

- MS SQL Server - wersja 2019, 2022
- PostgreSQL - wersja 15/16
- SQLite
- Narzędzia do komunikacji z bazą danych
    - SSMS - Microsoft SQL Managment Studio
    - DtataGrip 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

---

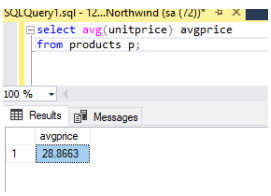
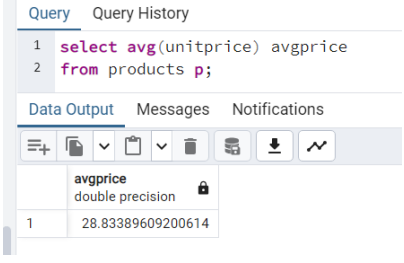
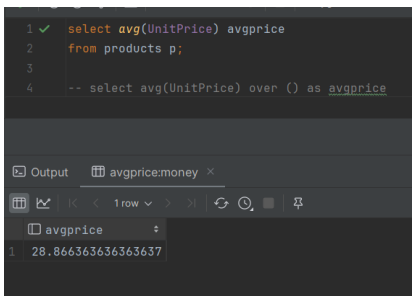
Zadanie 1 - obserwacja

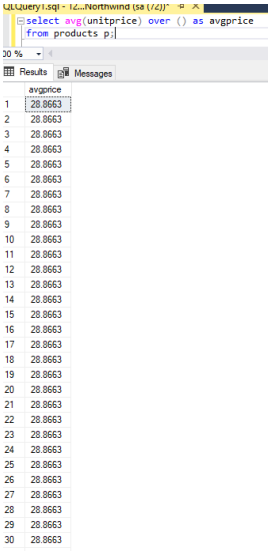
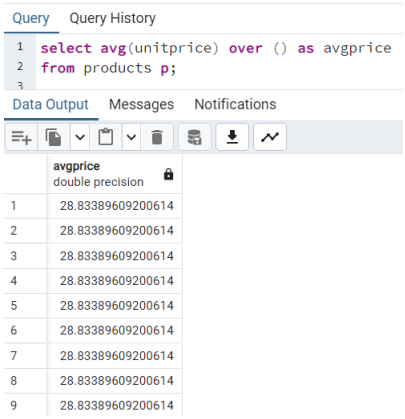
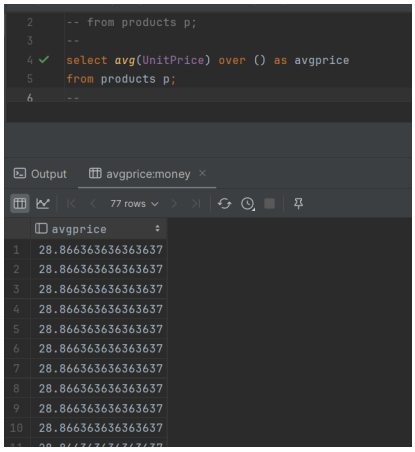
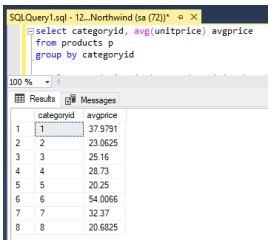
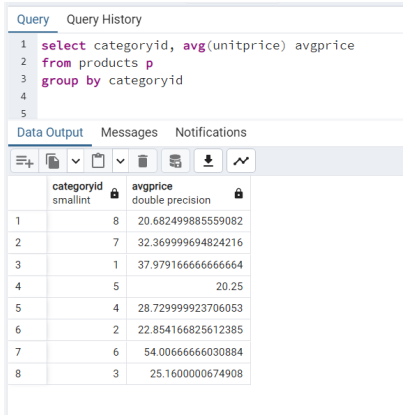
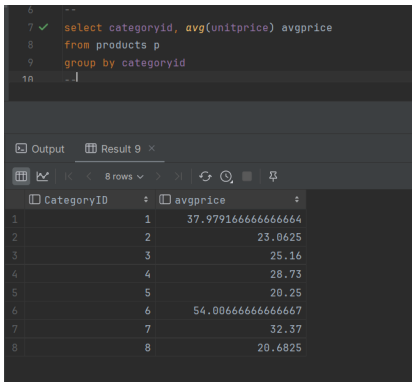
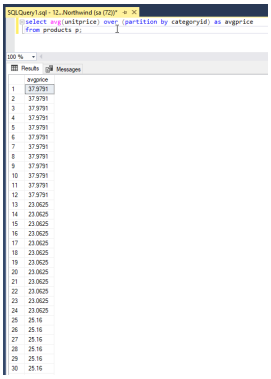
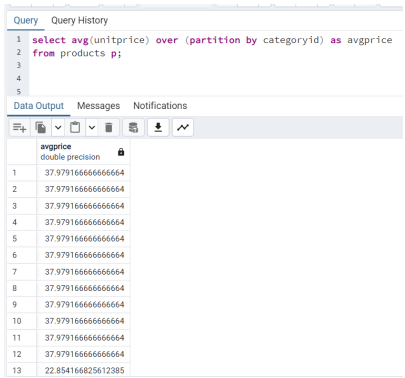
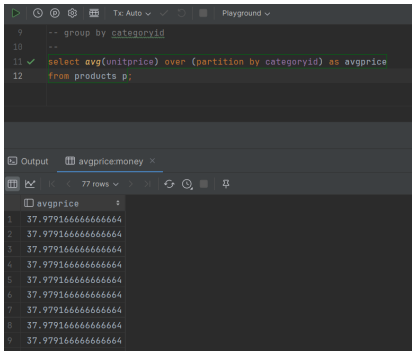
Wykonaj i porównaj wyniki następujących poleceń.

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

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

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

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

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

Widzimy, że funkcje okna przypisują obliczoną wartość każdemu wierszowi danych. Grupowanie automatycznie agreguje wartości do grup.

Zapytanie	MySQL	Postgres	SQLite
1			

Zapytanie	MySQL	Postgres	SQLite
2			
3			
4			

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
```

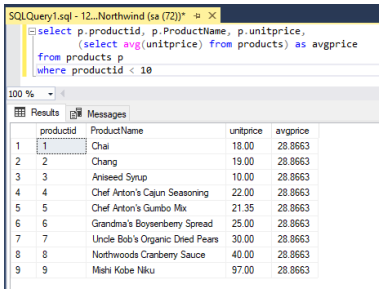
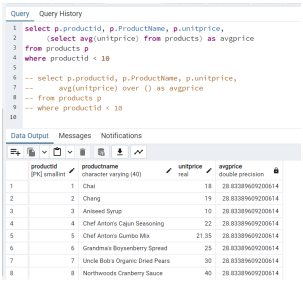
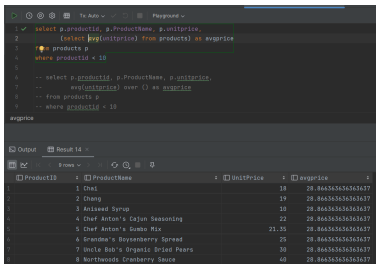
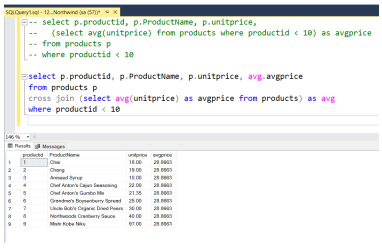
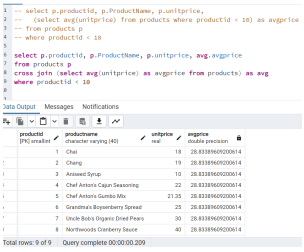
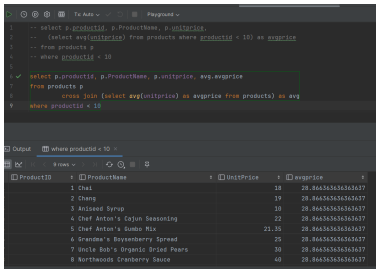
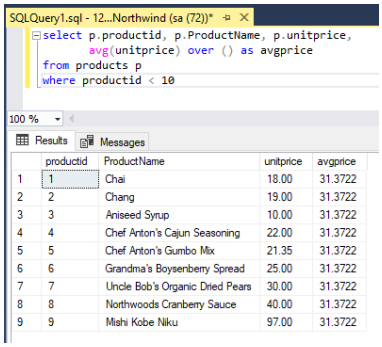
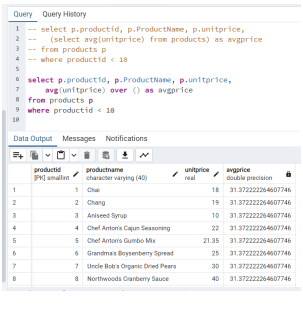
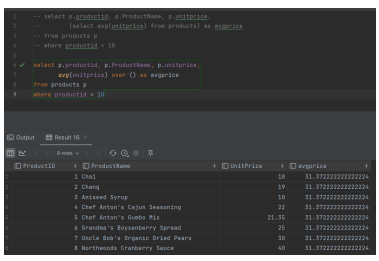
```
where productid < 10

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

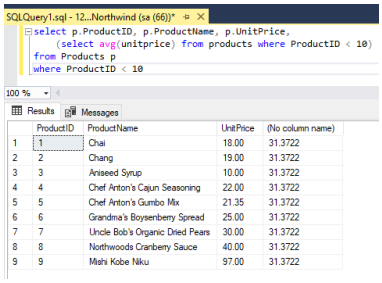
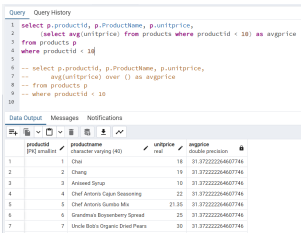
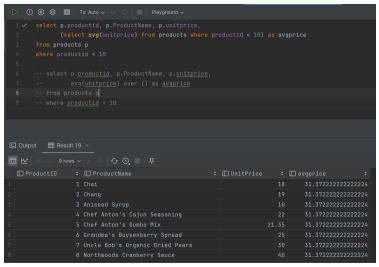
Jaka jest różnica? Czego dotyczy warunek w każdym z przypadków? Napisz polecenie równoważne

1. z wykorzystaniem podzapytania
2. z wykorzystaniem funkcji okna. Napisz polecenie równoważne

---

1. Pierwsze zapytanie używa `podzapytania w klauzuli SELECT` do obliczenia średniej ceny wszystkich produktów. Warunek productid < 10 jest używany do filtracji wyników na podstawie identyfikatora produktu mniejszego niż 10.
2. Drugie zapytanie wykorzystuje `funkcję okna avg(unitprice) over ()` do obliczenia średniej ceny wszystkich produktów, ale bez potrzeby podzapytania. Warunek productid < 10 również jest używany do filtracji wyników na podstawie identyfikatora produktu mniejszego niż 10.

Zapytanie	MySQL	Postgres	SQLite
1 oryginalne			
1 równoważne			
2 oryginalne			

Zapytanie	MySQL	Postgres	SQLite
2 równoważne			

Zadanie 3

Baza: Northwind, tabela: products

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

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

- Polecenie z wykorzystaniem podzapytania

```
SELECT p.ProductID,
       p.ProductName,
       p.UnitPrice,
       (SELECT AVG(p2.UnitPrice) FROM Products AS p2) AS AveragePrice
FROM Products AS p
```

- Polecenie z wykorzystaniem joina

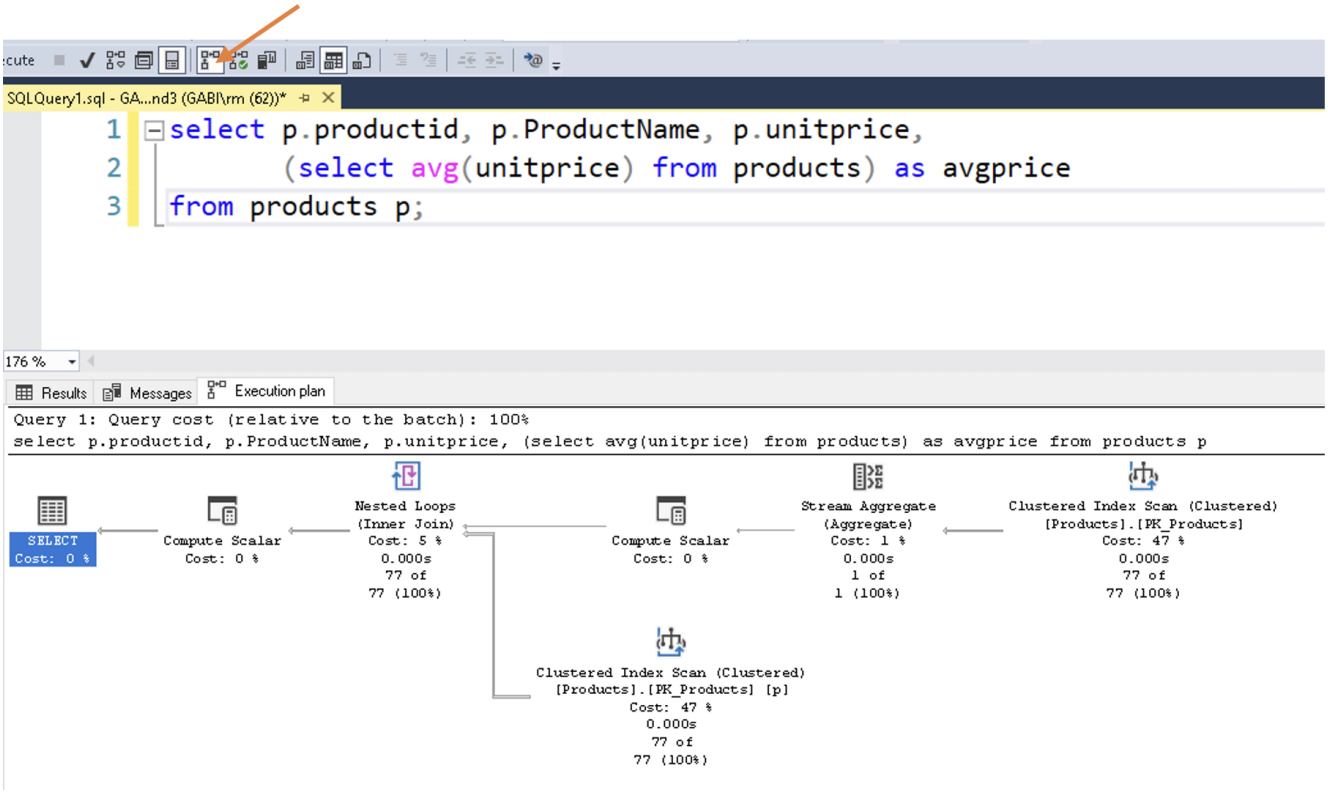
```
SELECT p.ProductID,
       p.ProductName,
       p.UnitPrice,
       AVG(p2.UnitPrice) AS AveragePrice
FROM Products p
         JOIN Products p2 ON 1 = 1
GROUP BY p.ProductID, p.ProductName, p.UnitPrice
```

- Polecenie z wykorzystaniem funkcji okna

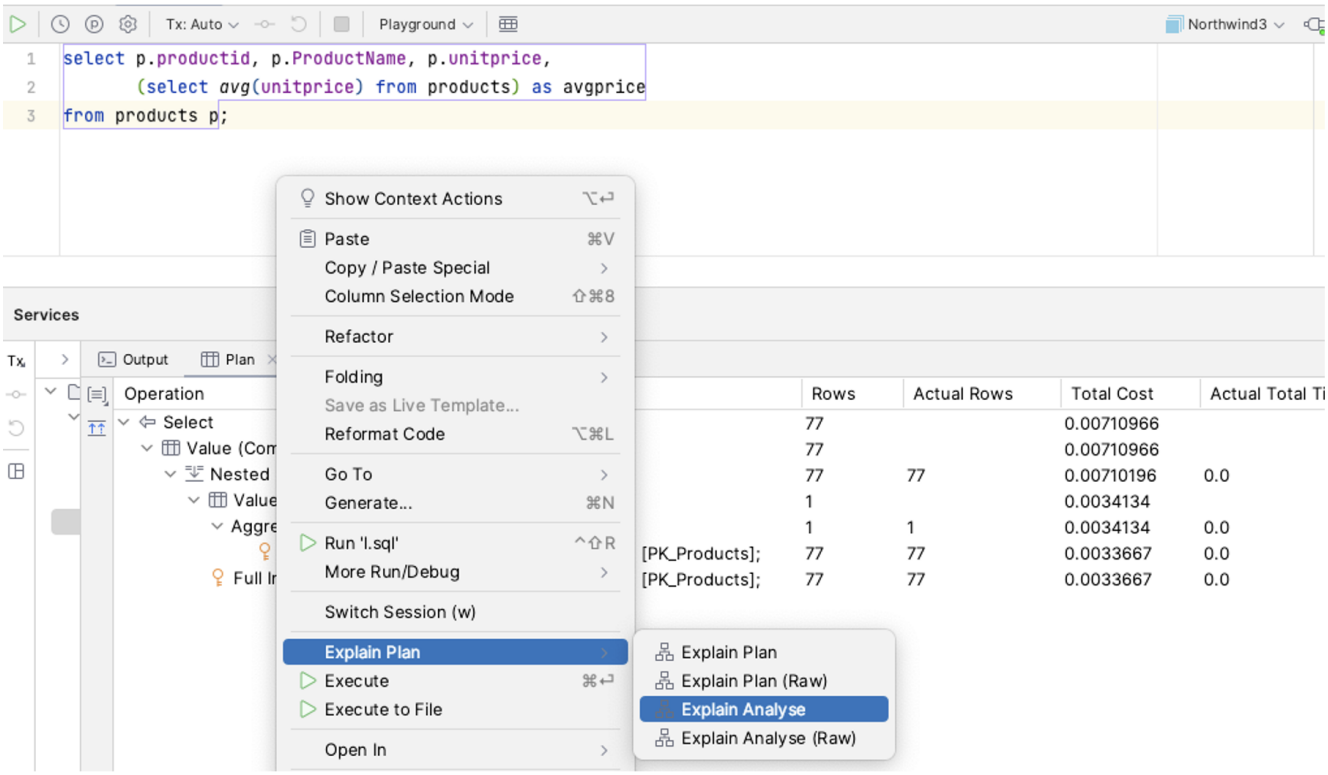
```
SELECT p.ProductID,
       p.ProductName,
       p.UnitPrice,
       AVG(UnitPrice) OVER() AS AveragePrice
FROM Products AS p
```

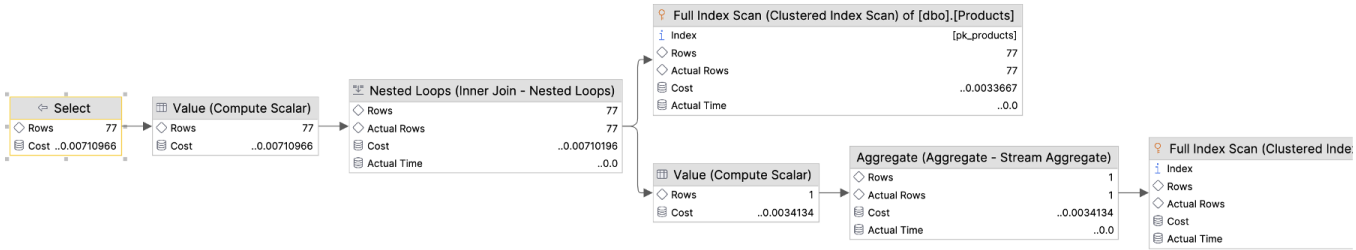
Przetestuj działanie w różnych SZBD (MS SQL Server, PostgreSql, SQLite)

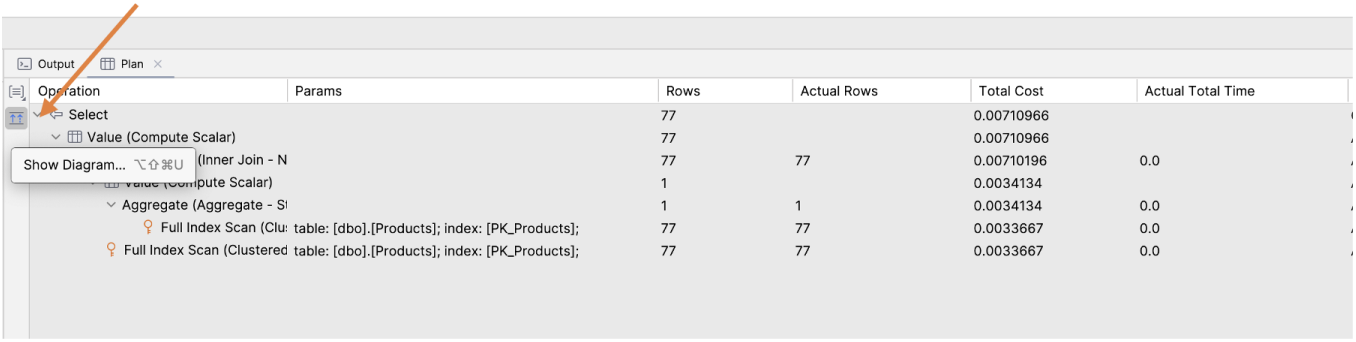
W SSMS włącz dwie opcje: Include Actual Execution Plan oraz Include Live Query Statistics

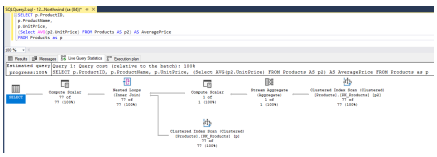
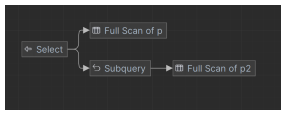
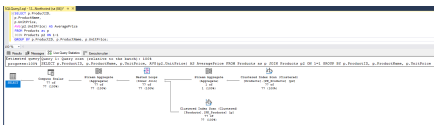
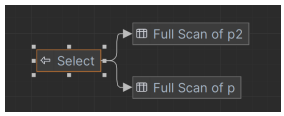
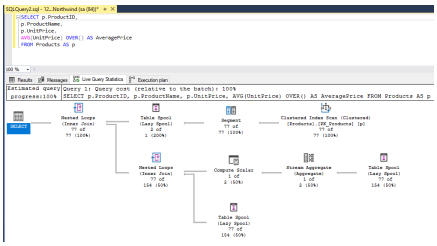
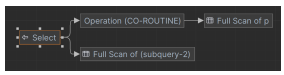


W DataGrip użyj opcji Explain Plan/Explain Analyze

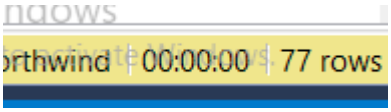
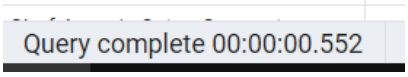
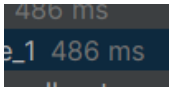
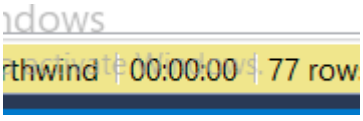
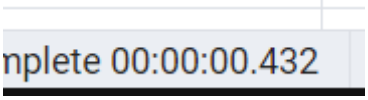
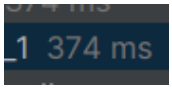
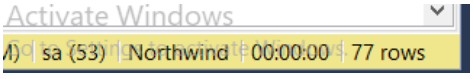
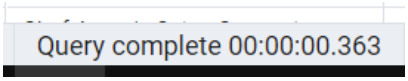
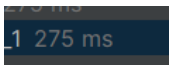




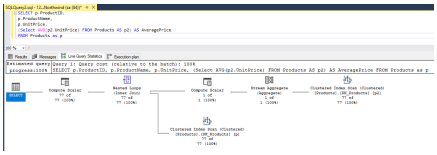
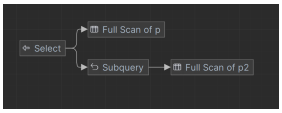
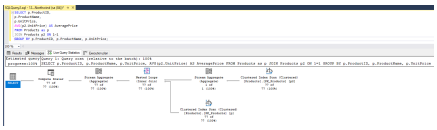
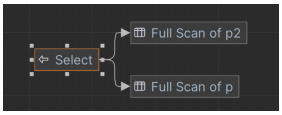
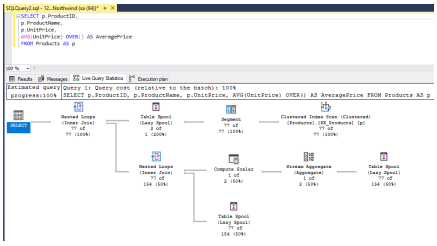
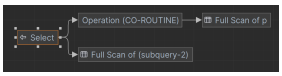


Zapytanie	MySQL	Postgres	SQLite
Podzapytanie			
Join			
Funkcja okna			

Porównanie czasów wykonania

Zapytanie	MySQL	Postgres	SQLite
Podzapytanie			
Join			
Funkcja okna			

Porównanie planów wykonania

Zapytanie	MySQL	Postgres	SQLite
Podzapytanie			
Join			
Funkcja okna			

Sqlite nie daje pełnej możliwości zwizualizowania planu. Datagrip pozwala jednak na zrobienie tego z Postgresem.

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.

- Polecenie z wykorzystaniem podzapytania

```
SELECT p.ProductID,
       p.ProductName,
       p.UnitPrice,
       (SELECT AVG(p2.UnitPrice)
        FROM Products AS p2
        WHERE p2.CategoryID = p.CategoryID) AS AvgCategoryPrice
FROM Products p
WHERE p.UnitPrice > (SELECT AVG(p3.UnitPrice)
                     FROM Products p3
                     WHERE p3.CategoryID = p.CategoryID)
```

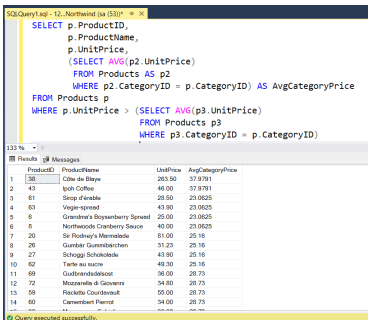
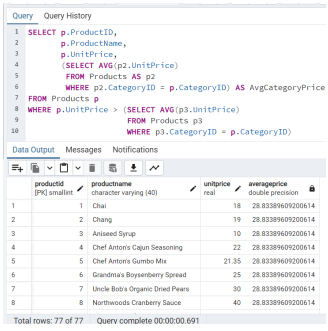
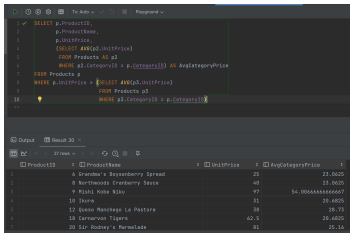
- Polecenie z wykorzystaniem joina

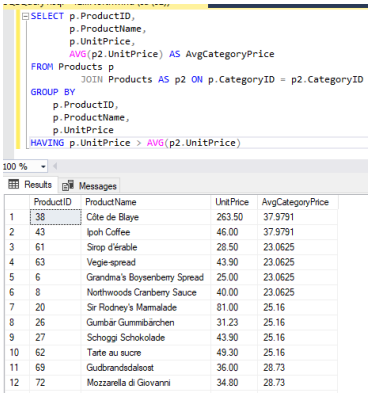
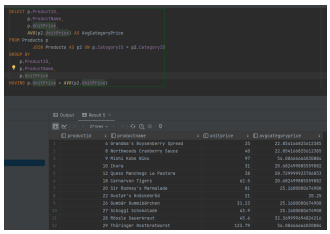
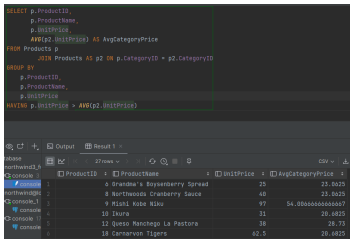
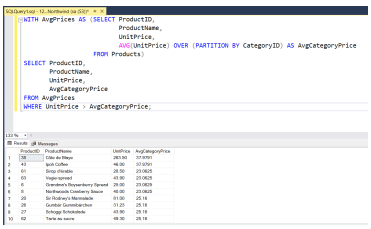
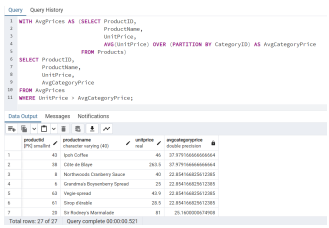
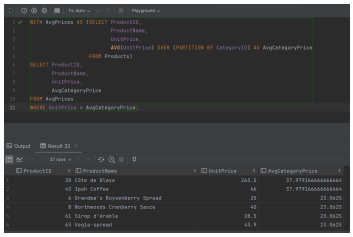
```
SELECT p.ProductID,
       p.ProductName,
       p.UnitPrice,
       AVG(p2.UnitPrice) AS AvgCategoryPrice
FROM Products p
         JOIN Products AS p2 ON p.CategoryID = p2.CategoryID
GROUP BY
    p.ProductID,
    p.ProductName,
    p.UnitPrice
HAVING p.UnitPrice > AVG(p2.UnitPrice)
```

- Polecenie z wykorzystaniem funkcji okna

```
WITH AvgPrices AS (SELECT ProductID,
                          ProductName,
                          UnitPrice,
                          AVG(UnitPrice) OVER (PARTITION BY CategoryID) AS
AvgCategoryPrice
                   FROM Products)
SELECT ProductID,
       ProductName,
       UnitPrice,
       AvgCategoryPrice
FROM AvgPrices
WHERE UnitPrice > AvgCategoryPrice;
```

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

Zapytanie	MySQL	Postgres	SQLite
Podzapytanie			

Zapytanie	MySQL	Postgres	SQLite
Join			
Funkcja okna			

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 Srerver

```
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

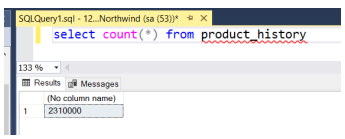
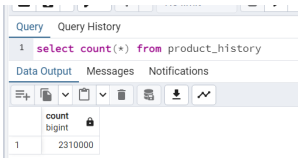
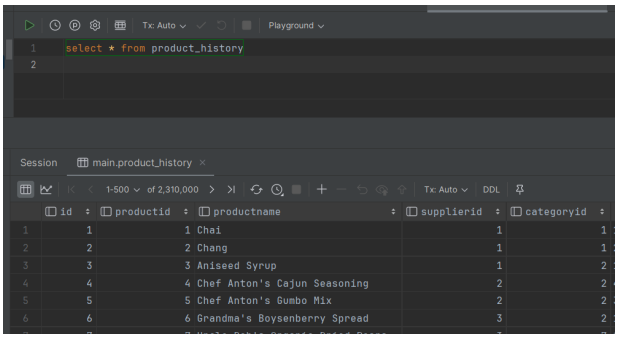
```
create table product_history
(
    id              int generated always as identity not null
        constraint pkproduct_history primary key,
    productid       int,
    productname     varchar(40)                      not null,
    supplierid      int null,
    categoryid      int null,
    quantityperunit 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

MySQL	Postgres	SQLite
		

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.

(przykłady poniżej)

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

- Polecenie z wykorzystaniem podzapytania

```
SELECT p.ProductID,
       p.ProductName,
       p.UnitPrice,
       (SELECT AVG(ph.UnitPrice)
        FROM product_history ph
        WHERE ph.CategoryID = p.CategoryID) as avgprice
FROM product_history as p
WHERE p.UnitPrice >
      (SELECT AVG(ph.UnitPrice)
       FROM product_history ph
       WHERE ph.CategoryID = p.CategoryID)
```

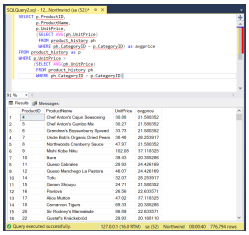
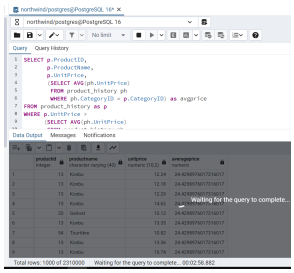
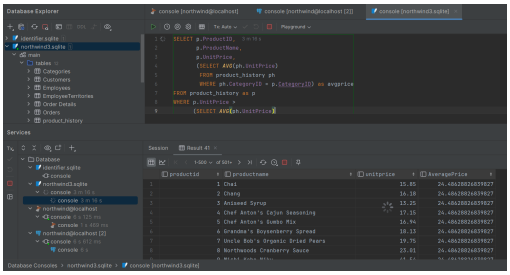
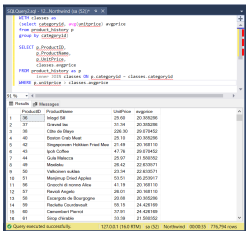
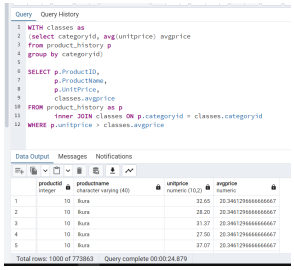
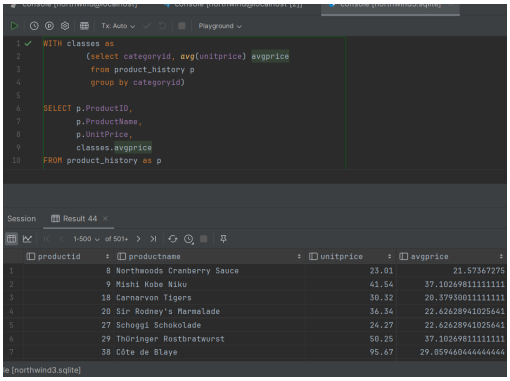
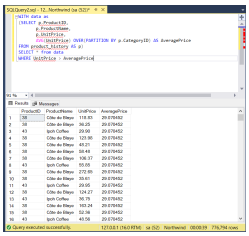
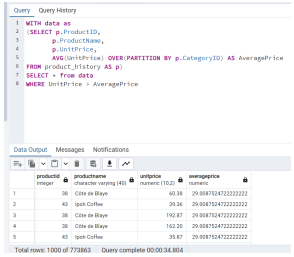
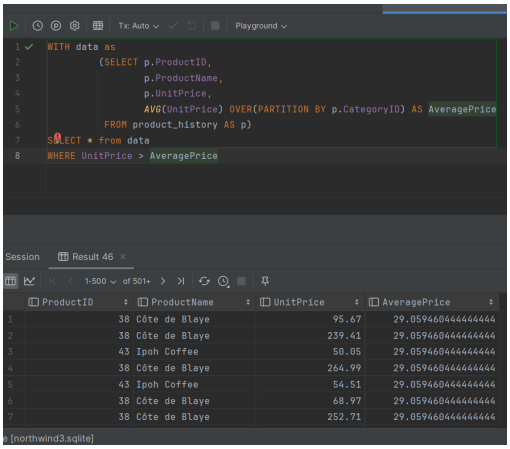
- Polecenie z wykorzystaniem joina

```
WITH classes as
(select categoryid, avg(unitprice) avgprice
from product_history p
group by categoryid)

SELECT p.ProductID,
       p.ProductName,
       p.UnitPrice,
       classes.avgprice
FROM product_history as p
       inner JOIN classes ON p.categoryid = classes.categoryid
WHERE p.unitprice > classes.avgprice
```

- Polecenie z wykorzystaniem funkcji okna

```
WITH data as
(SELECT p.ProductID,
       p.ProductName,
       p.UnitPrice,
       AVG(UnitPrice) OVER(PARTITION BY p.CategoryID) AS AveragePrice
FROM product_history AS p)
SELECT * from data
WHERE UnitPrice > AveragePrice
```

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

Zapytanie	MySQL	Postgres	SQLite
Podzapytanie			
Join			
Funkcja okna			

Możemy zaobserwować, że zapytanie z użyciem joina dla postgres nie wykonało się nawet po 8 minutach od wywołania. SSMS dla MySQL nie rozpoznaje nowo utworzonej tabeli, ale potrafi wykonać na niej zapytanie

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

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

- Polecenie z wykorzystaniem podzapytania

```
SELECT ph.id,
       ph.ProductID,
       ph.ProductName,
       ph.UnitPrice,
       (select avg(ph2.unitprice) from product_history as ph2 where
ph.categoryid = ph2.categoryid) as AveragePrice,
       (select sum(ph3.value) from product_history as ph3 where
ph.categoryid = ph3.categoryid) as TotalSale,
       (select avg(ph4.unitprice) from product_history as ph4 where
ph.productid = ph4.productid and YEAR(ph.date) = YEAR(ph4.date)) as
AveragePriceOverYear
FROM product_history AS ph
```

- Polecenie z wykorzystaniem joina

```
SELECT
    ph.id,
    ph.ProductID,
    ph.ProductName,
    ph.UnitPrice,
    AVG(ph2.unitprice) AS AveragePrice,
    SUM(ph3.value) AS TotalSale,
    AVG(ph4.unitprice) AS AveragePriceOverYear
FROM
    product_history AS ph
JOIN
    product_history AS ph2 ON ph.categoryid = ph2.categoryid
JOIN
    product_history AS ph3 ON ph.categoryid = ph3.categoryid
JOIN
    product_history AS ph4 ON ph.productid = ph4.productid AND
YEAR(ph.date) = YEAR(ph4.date)
GROUP BY
    ph.id,
    ph.ProductID,
    ph.ProductName,
    ph.UnitPrice;
```

- Polecenie z wykorzystaniem funkcji okna

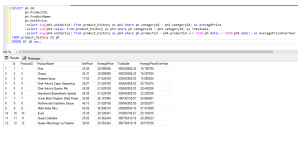
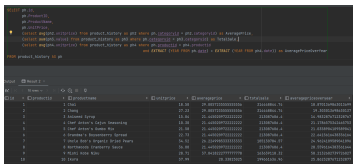
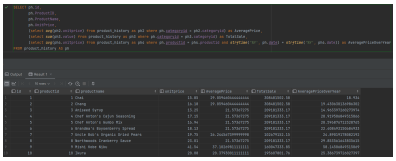
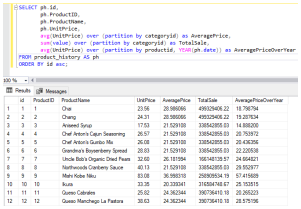
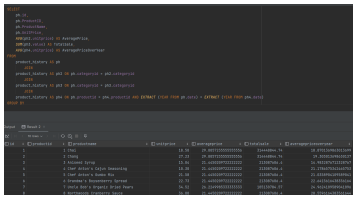
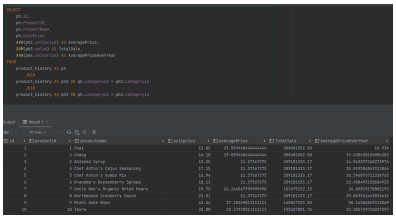
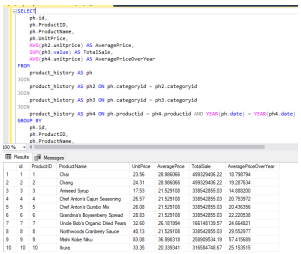
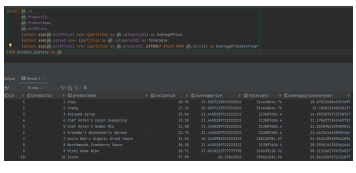
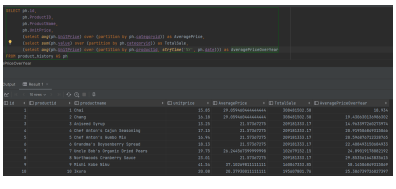
```
SELECT ph.id,
       ph.ProductID,
       ph.ProductName,
       ph.UnitPrice,
       (select avg(ph.UnitPrice) over (partition by ph.categoryid)) as
AveragePrice,
       (select sum(ph.value) over (partition by ph.categoryid)) as
TotalSale,
       (select avg(ph.UnitPrice) over (partition by ph.productid,
YEAR(ph.date))) as AveragePriceOverYear
FROM product_history AS ph
```

Różnice w:

- SQLite: należy zmienić `YEAR(ph.date)` na `strftime('%Y', ph.date)`
- Postgres: należy zamienić `YEAR(ph.date)` na `EXTRACT (YEAR FROM ph.date)`

Porównaj czasy oraz plany wykonania zapytań.

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

Zapytanie	MySQL	Postgres	SQLite
Podzapytanie			
Join			
Funkcja okna			

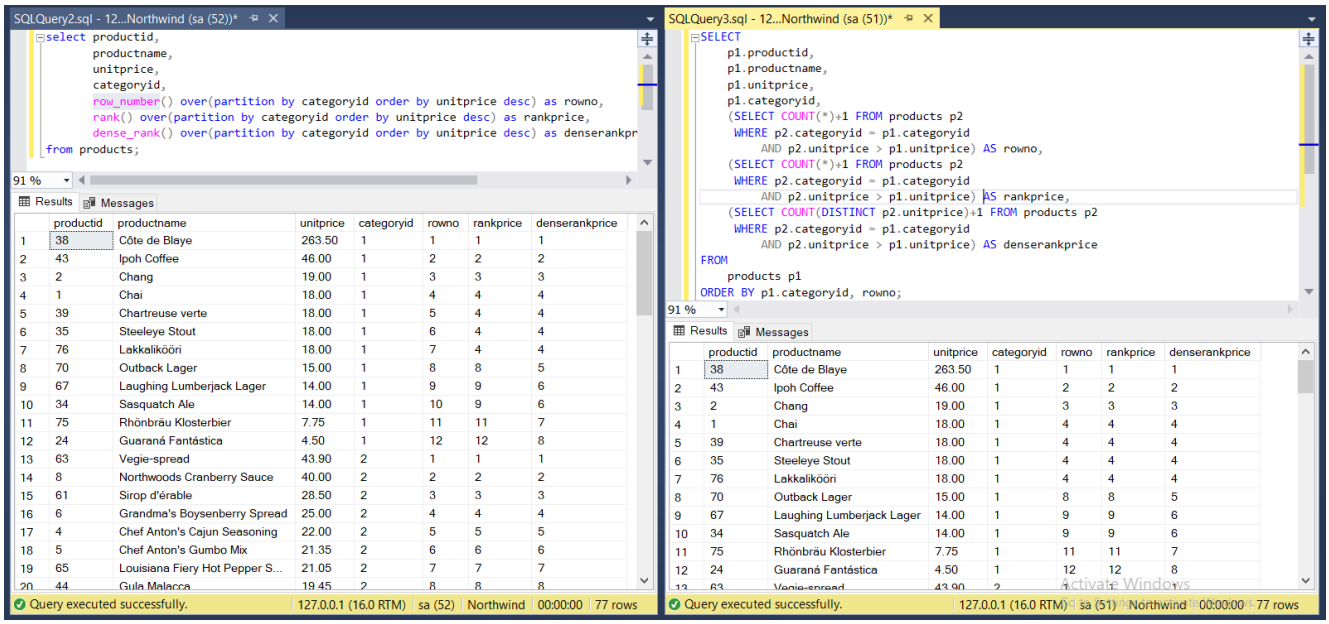
Zadanie 8 - obserwacja

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;
```

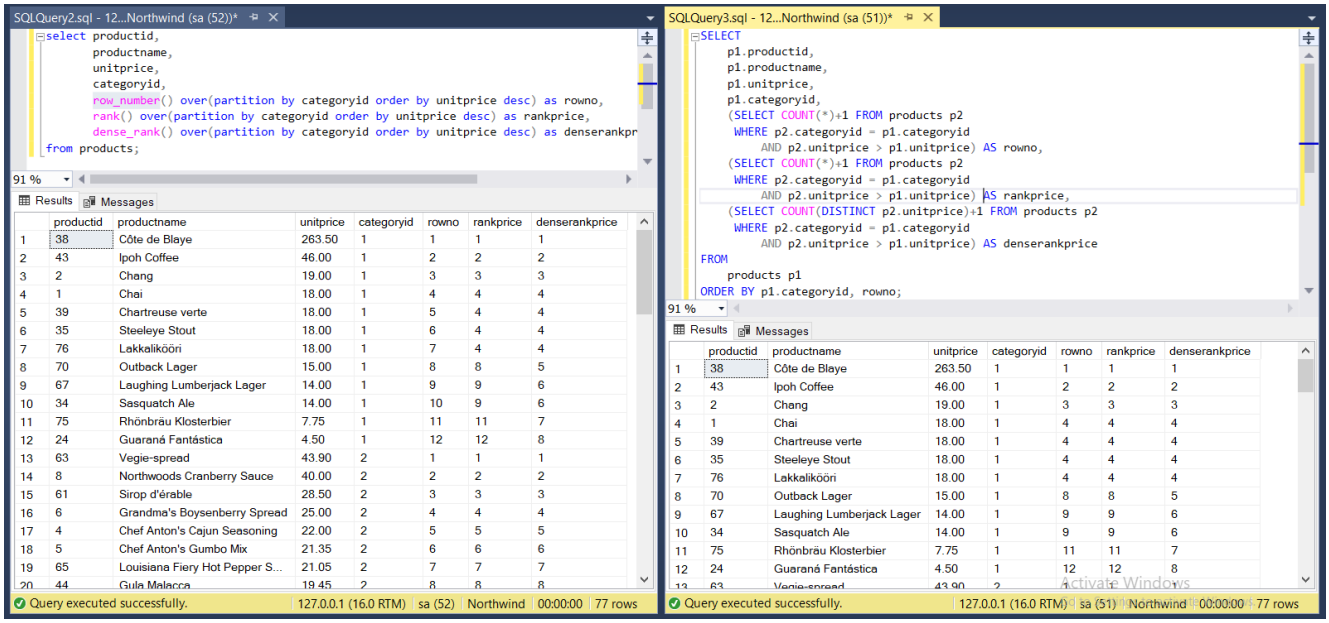
MySQL



Spróbuj uzyskać ten sam wynik bez użycia funkcji okna

```
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;
```

MySQL



Widać zasadniczą różnicę działania dla kolumny `rowno` zwiększanie wartość musiałaby rosnąć dla zbioru produktów o tej samej cenie. Nie jest to trywialne bez funkcji okna (przynajmniej dla studenta który się uczy)

Zadanie 9

Baza: Northwind, tabela product_history

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

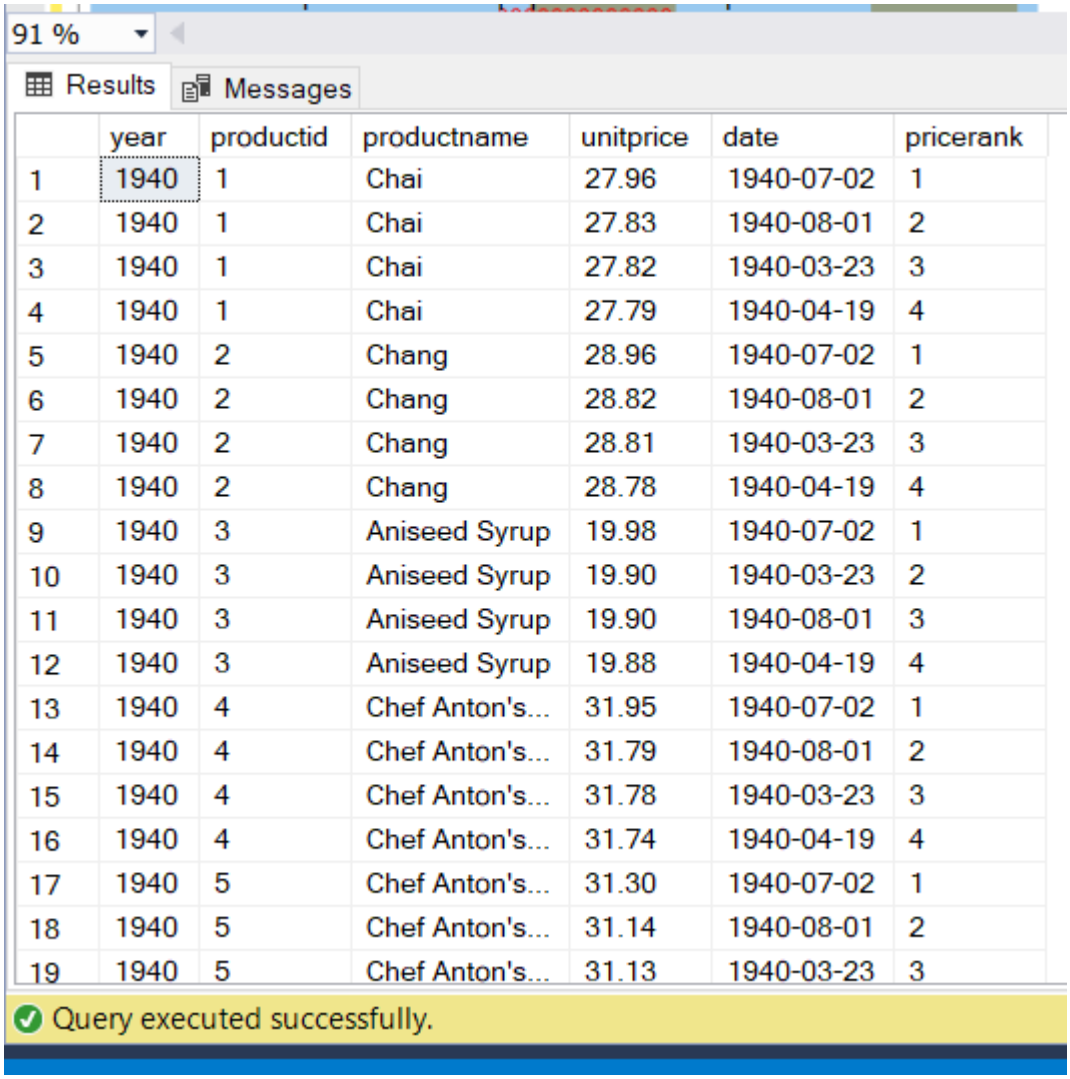
- rok
- id produktu
- nazwę produktu
- cenę
- datę (datę uzyskania przez produkt takiej ceny)
- pozycję w rankingu

Uporządkuj wynik wg roku, nr produktu, pozycji w rankingu

```
WITH ranked_prices AS (
  SELECT
    YEAR(date) AS year, p.productid, p.ProductName, p.unitprice, p.date,
    ROW_NUMBER() OVER(PARTITION BY p.productid, YEAR(date) ORDER BY
p.unitprice DESC) AS pricerank
  FROM product_history as p
  INNER JOIN products ON p.productid = products.ProductID
)

SELECT year, productid, productname, unitprice, date, pricerank
```

```
FROM ranked_prices
WHERE pricerank <= 4
ORDER BY year, productid, pricerank;
```

MySQL



91 %

Results Messages

	year	productid	productname	unitprice	date	pricerank
1	1940	1	Chai	27.96	1940-07-02	1
2	1940	1	Chai	27.83	1940-08-01	2
3	1940	1	Chai	27.82	1940-03-23	3
4	1940	1	Chai	27.79	1940-04-19	4
5	1940	2	Chang	28.96	1940-07-02	1
6	1940	2	Chang	28.82	1940-08-01	2
7	1940	2	Chang	28.81	1940-03-23	3
8	1940	2	Chang	28.78	1940-04-19	4
9	1940	3	Aniseed Syrup	19.98	1940-07-02	1
10	1940	3	Aniseed Syrup	19.90	1940-03-23	2
11	1940	3	Aniseed Syrup	19.90	1940-08-01	3
12	1940	3	Aniseed Syrup	19.88	1940-04-19	4
13	1940	4	Chef Anton's...	31.95	1940-07-02	1
14	1940	4	Chef Anton's...	31.79	1940-08-01	2
15	1940	4	Chef Anton's...	31.78	1940-03-23	3
16	1940	4	Chef Anton's...	31.74	1940-04-19	4
17	1940	5	Chef Anton's...	31.30	1940-07-02	1
18	1940	5	Chef Anton's...	31.14	1940-08-01	2
19	1940	5	Chef Anton's...	31.13	1940-03-23	3

Query executed successfully.

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 ranked_prices AS (
  SELECT
    YEAR(date) AS year,
    p.productid,
    p.ProductName,
    p.unitprice,
    p.date,
    (SELECT COUNT(*) + 1 FROM product_history p2
     WHERE p2.productid = p.productid
         AND p2.unitprice > p.unitprice AND YEAR(p.date) = YEAR(p2.date))
as pricerank
  FROM product_history as p
  INNER JOIN products ON p.productid = products.ProductID
```

```
)

SELECT
  year,
  productid,
  productname,
  unitprice,
  date,
  pricerank
FROM
  ranked_prices
WHERE
  pricerank <= 4
ORDER BY
  year,
  productid,
  pricerank;
```

Czas działania dla mysql był bardzo długi i nie uzyskaliśmy wyniku w rozsądnym czasie.

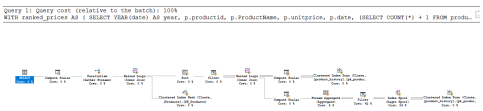
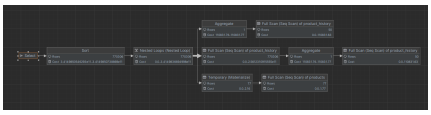
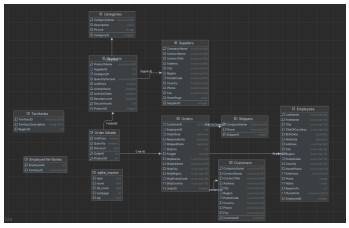
Dla postgres musieliśmy użyć innej składni by wywołać zapytanie, jednak dalej jego czas obliczeń nie był skończony w rozsądnym czasie

```
WITH ranked_prices AS (
  SELECT
    extract ("YEAR" from (p.date)) AS year,
    p.productid,p.ProductName,p.unitprice,p.date,
    (SELECT COUNT(*) + 1 FROM product_history p2
     WHERE p2.productid = p.productid
         AND p2.unitprice > p.unitprice AND extract ("YEAR" from (p.date))
= extract ("YEAR" from (p2.date))) as pricerank
  FROM product_history as p
  INNER JOIN products ON p.productid = products.ProductID
)

SELECT
  year,
  productid,productname,unitprice,date,pricerank
FROM
  ranked_prices
WHERE
  pricerank <= 4
ORDER BY
  year,productid,pricerank;
```

Składnia wyboru roku różniła się również dla sqlite. Również czas nie był zadowalający i nie dostaliśmy wyniku

```

```

```
WITH ranked_prices AS (
    SELECT
        strftime('%Y',p.date) AS year,
        p.productid,p.ProductName,p.unitprice,p.date,
        (SELECT COUNT(*) + 1 FROM product_history p2
         WHERE p2.productid = p.productid
           AND p2.unitprice > p.unitprice AND strftime('%Y',p.date) =
strftime('%Y',p2.date)) as pricerank
    FROM product_history as p
             INNER JOIN products ON p.productid = products.ProductID
)

SELECT
    year,productid,productname,unitprice,date,pricerank
FROM
    ranked_prices
WHERE
    pricerank <= 4
ORDER BY
    year,productid,pricerank;
```

MySQL	Postgres	SQLite
		

Zadanie 10 - obserwacja

Funkcje `lag()`, `lead()`

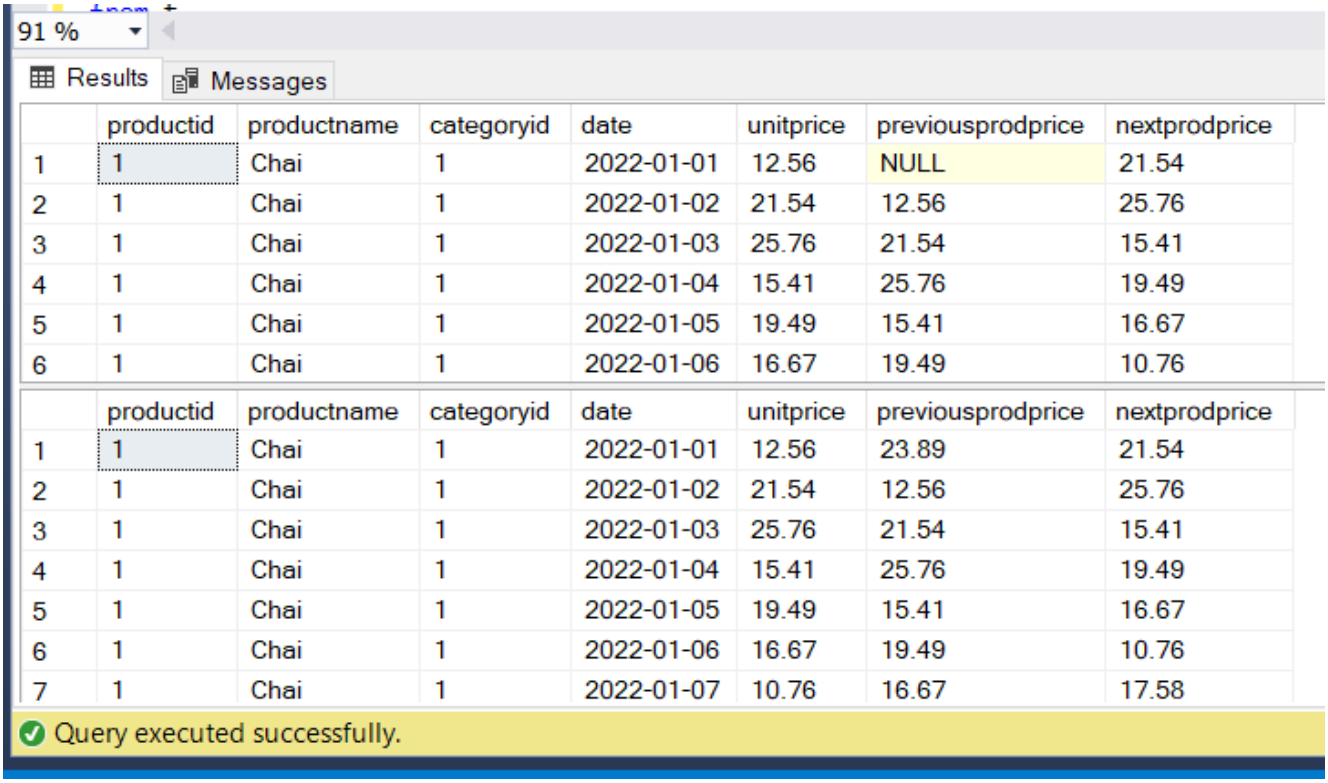
Wykonaj polecenia, zaobserwuj wynik. Jak działają funkcje `lag()`, `lead()`

```
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;
```

Funkcje lead i follow zwracają przesuniętą kolumnę. odpowiednio w dół lub w górę. tzn. używając lag dostaniemy wartość która w wybranej kolumnie pojawiła się 1 rząd wyżej

MySQL



	productid	productname	categoryid	date	unitprice	previousprodprice	nextprodprice
1	1	Chai	1	2022-01-01	12.56	NULL	21.54
2	1	Chai	1	2022-01-02	21.54	12.56	25.76
3	1	Chai	1	2022-01-03	25.76	21.54	15.41
4	1	Chai	1	2022-01-04	15.41	25.76	19.49
5	1	Chai	1	2022-01-05	19.49	15.41	16.67
6	1	Chai	1	2022-01-06	16.67	19.49	10.76

	productid	productname	categoryid	date	unitprice	previousprodprice	nextprodprice
1	1	Chai	1	2022-01-01	12.56	23.89	21.54
2	1	Chai	1	2022-01-02	21.54	12.56	25.76
3	1	Chai	1	2022-01-03	25.76	21.54	15.41
4	1	Chai	1	2022-01-04	15.41	25.76	19.49
5	1	Chai	1	2022-01-05	19.49	15.41	16.67
6	1	Chai	1	2022-01-06	16.67	19.49	10.76
7	1	Chai	1	2022-01-07	10.76	16.67	17.58

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)

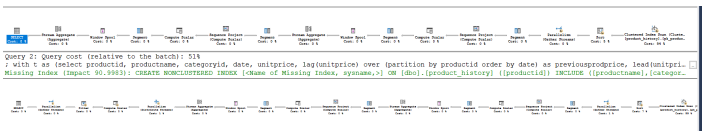
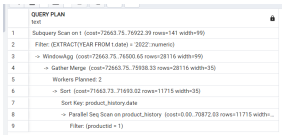
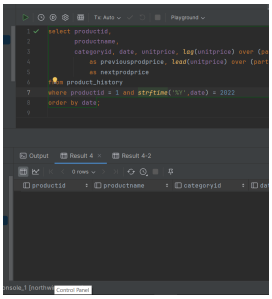
```
-- wyniki ...
```

Wykonanie zapytania bez funkcji okna okazało się być zbyt trudne dlatego możemy dokonać porównania funkcji okna dla różnych SZBD

SSMS nie zapełnia możliwości wykonania czytelnego zrzutu ekranu dla planu wykonania. Jest on jestnak sekwencyjny. Czas trwania jest niemal natychmiastowy

dla postgres czas wykonania wyniósł ponad sekundę. Dodatkowu znowu należało zamienić `year (date)` na `extract ("YEAR" from date)`

Sqlite zwrócił puste tabele pomimo użycia funkcji `strftime('%Y',date)` do wydobycia daty

MySQL	Postgres	SQLite
		

Zadanie 11

Baza: Northwind, tabele customers, orders, order details

Napisz polecenie które wyświetla inf. o zamówieniach

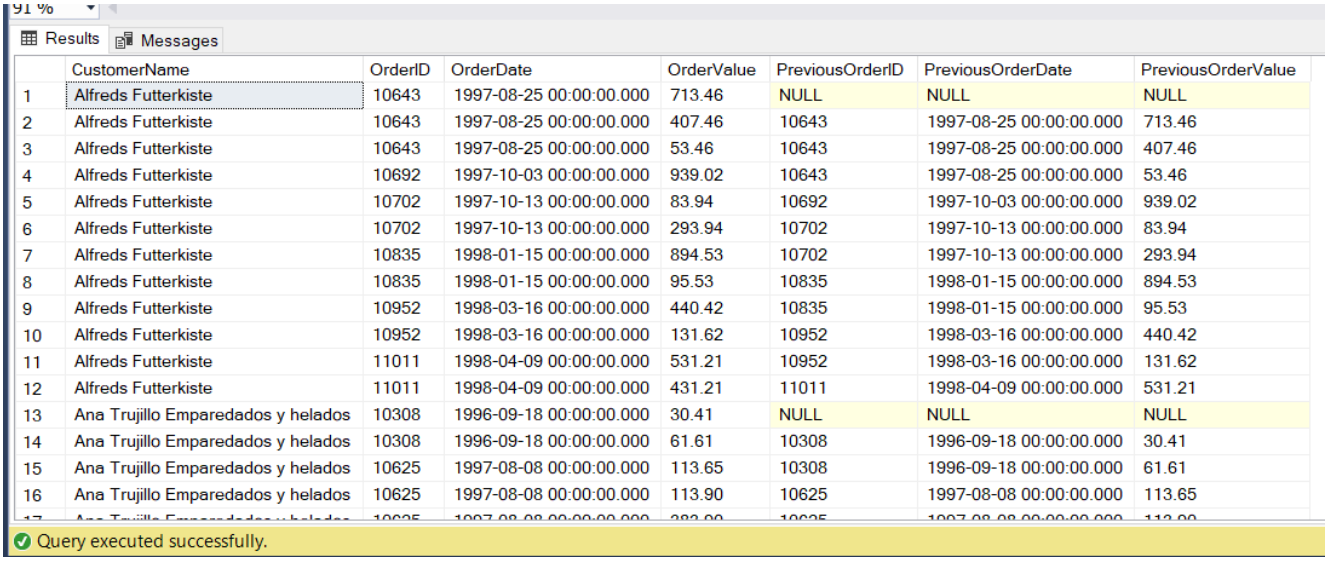
Zbiór wynikowy powinien zawierać:

- 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.

```
SELECT
        c.CompanyName AS CustomerName,
        o.OrderID,
        o.OrderDate,
        od.UnitPrice * od.Quantity + o.freight AS OrderValue,
        LAG(o.OrderID) OVER (PARTITION BY o.CustomerID ORDER BY
o.OrderDate) AS PreviousOrderID,
        LAG(o.OrderDate) OVER (PARTITION BY o.CustomerID ORDER BY
o.OrderDate) AS PreviousOrderDate,
        LAG(od.UnitPrice * od.Quantity + o.freight) OVER (PARTITION BY
o.CustomerID ORDER BY o.OrderDate) AS PreviousOrderValue
    FROM
        Orders o
    INNER JOIN
        Customers c ON o.CustomerID = c.CustomerID
    INNER JOIN
        OrderDetails od ON o.OrderID = od.OrderID
```

MySQL

MySQL



Results Messages

	CustomerName	OrderID	OrderDate	OrderValue	PreviousOrderID	PreviousOrderDate	PreviousOrderValue
1	Alfreds Futterkiste	10643	1997-08-25 00:00:00.000	713.46	NULL	NULL	NULL
2	Alfreds Futterkiste	10643	1997-08-25 00:00:00.000	407.46	10643	1997-08-25 00:00:00.000	713.46
3	Alfreds Futterkiste	10643	1997-08-25 00:00:00.000	53.46	10643	1997-08-25 00:00:00.000	407.46
4	Alfreds Futterkiste	10692	1997-10-03 00:00:00.000	939.02	10643	1997-08-25 00:00:00.000	53.46
5	Alfreds Futterkiste	10702	1997-10-13 00:00:00.000	83.94	10692	1997-10-03 00:00:00.000	939.02
6	Alfreds Futterkiste	10702	1997-10-13 00:00:00.000	293.94	10702	1997-10-13 00:00:00.000	83.94
7	Alfreds Futterkiste	10835	1998-01-15 00:00:00.000	894.53	10702	1997-10-13 00:00:00.000	293.94
8	Alfreds Futterkiste	10835	1998-01-15 00:00:00.000	95.53	10835	1998-01-15 00:00:00.000	894.53
9	Alfreds Futterkiste	10952	1998-03-16 00:00:00.000	440.42	10835	1998-01-15 00:00:00.000	95.53
10	Alfreds Futterkiste	10952	1998-03-16 00:00:00.000	131.62	10952	1998-03-16 00:00:00.000	440.42
11	Alfreds Futterkiste	11011	1998-04-09 00:00:00.000	531.21	10952	1998-03-16 00:00:00.000	131.62
12	Alfreds Futterkiste	11011	1998-04-09 00:00:00.000	431.21	11011	1998-04-09 00:00:00.000	531.21
13	Ana Trujillo Emparedados y helados	10308	1996-09-18 00:00:00.000	30.41	NULL	NULL	NULL
14	Ana Trujillo Emparedados y helados	10308	1996-09-18 00:00:00.000	61.61	10308	1996-09-18 00:00:00.000	30.41
15	Ana Trujillo Emparedados y helados	10625	1997-08-08 00:00:00.000	113.65	10308	1996-09-18 00:00:00.000	61.61
16	Ana Trujillo Emparedados y helados	10625	1997-08-08 00:00:00.000	113.90	10625	1997-08-08 00:00:00.000	113.65

Query executed successfully.

Zadanie 12 - obserwacja

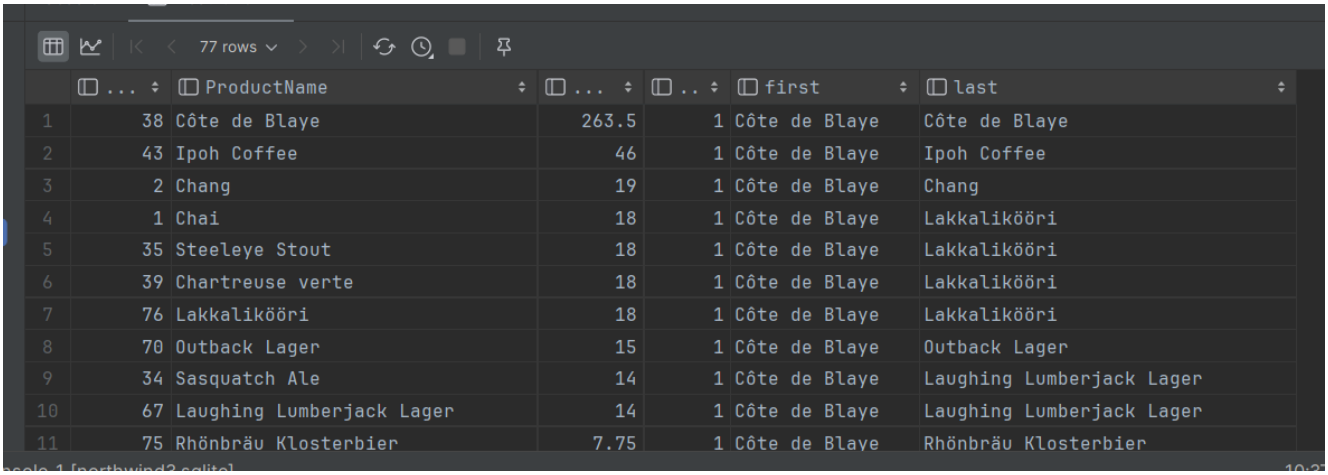
Funkcje `first_value()`, `last_value()`

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;
```

SQLite

SQLite



77 rows

	...	ProductName	...	..	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
10	67	Laughing Lumberjack Lager	14	1	Côte de Blaye	Laughing Lumberjack Lager
11	75	Rhönbräu Klosterbier	7.75	1	Côte de Blaye	Rhönbräu Klosterbier

Funkcje first_value, oraz last_value zwracają odpowiednio pierwszy i ostatni element w danej partycji/grupie, a przynajmniej powinny. Okzuje się, że last_value nie działa tak jak powinno. Wygląda na to, że działa błędnie.

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 unitprice ASC) as last
FROM products p
ORDER BY p.categoryid, p.unitprice DESC;
```

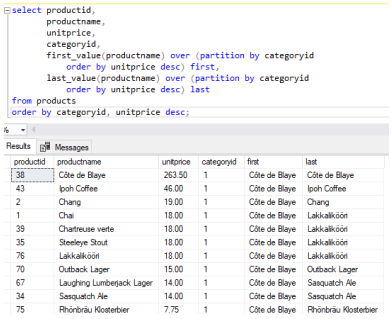
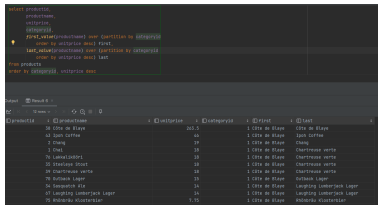
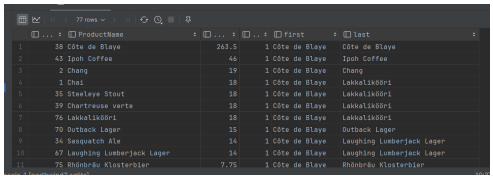
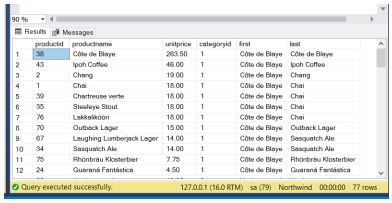
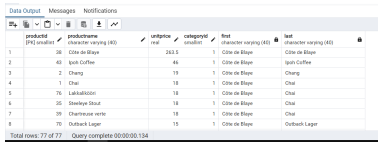
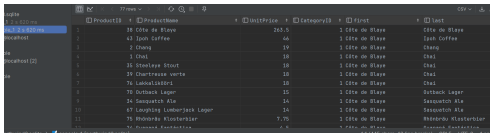
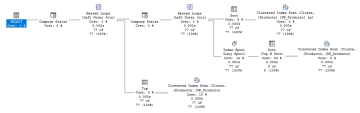
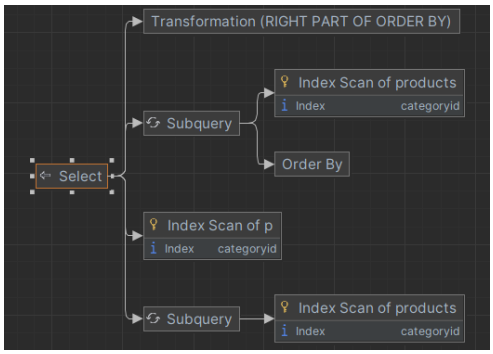
Dla Postgresa i SQLite zamiast top 1 trzeba użyć limit 1

```
SELECT p.productid,
       p.productname,
       p.unitprice,
       p.categoryid,
       (SELECT  productname
        FROM products
        WHERE categoryid = p.categoryid
        ORDER BY unitprice DESC limit 1) AS first,
       (SELECT productname
        FROM products
```

```
        WHERE categoryid = p.categoryid and unitprice = p.unitprice
        ORDER BY unitprice ASC limit 1) as last
FROM products p
ORDER BY p.categoryid, p.unitprice DESC;
```

Bez użycia funkcji okna możemy uzyskać wyniki których potrzebujemy.

Wyniki i czasy

MySQL	Postgres	SQLite
		
		
		

Zadanie 13

Baza: Northwind, tabele orders, order details

Napisz polecenie które wyświetla inf. o zamówieniach

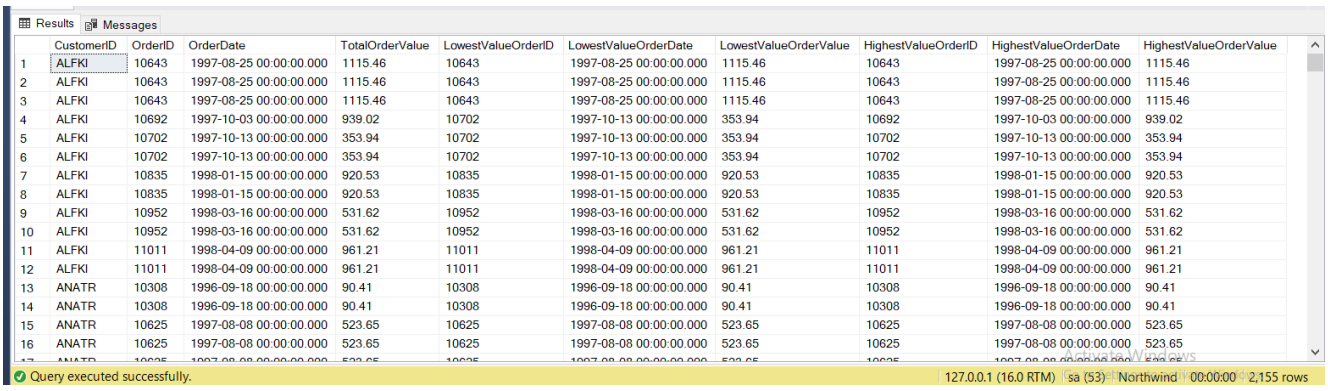
Zbiór wynikowy powinien zawierać:

- 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

```
SELECT
    CustomerID,
    OrderID,
    OrderDate,
    TotalOrderValue,
    FIRST_VALUE(OrderID) OVER (PARTITION BY CustomerID, YEAR(OrderDate),
MONTH(OrderDate) ORDER BY TotalOrderValue ASC) AS LowestValueOrderID,
    FIRST_VALUE(OrderDate) OVER (PARTITION BY CustomerID, YEAR(OrderDate),
MONTH(OrderDate) ORDER BY TotalOrderValue ASC) AS LowestValueOrderDate,
    FIRST_VALUE(TotalOrderValue) OVER (PARTITION BY CustomerID,
YEAR(OrderDate), MONTH(OrderDate) ORDER BY TotalOrderValue ASC) AS
LowestValueOrderValue,
    LAST_VALUE(OrderID) OVER (PARTITION BY CustomerID, YEAR(OrderDate),
MONTH(OrderDate) ORDER BY TotalOrderValue DESC) AS HighestValueOrderID,
    LAST_VALUE(OrderDate) OVER (PARTITION BY CustomerID, YEAR(OrderDate),
MONTH(OrderDate) ORDER BY TotalOrderValue DESC) AS HighestValueOrderDate,
    LAST_VALUE(TotalOrderValue) OVER (PARTITION BY CustomerID,
YEAR(OrderDate), MONTH(OrderDate) ORDER BY TotalOrderValue DESC) AS
HighestValueOrderValue
FROM (
    SELECT
        Orders.CustomerID,
        Orders.OrderID,
        Orders.OrderDate,
        (SUM(od.UnitPrice * od.Quantity) OVER (PARTITION BY
Orders.OrderID)) + Orders.freight AS TotalOrderValue
    FROM Orders
    JOIN [Order Details] as od ON Orders.OrderID = od.OrderID
) AS OrderSummary
```

MySQL



	CustomerID	OrderID	OrderDate	TotalOrderValue	LowestValueOrderID	LowestValueOrderDate	LowestValueOrderValue	HighestValueOrderID	HighestValueOrderDate	HighestValueOrderValue
1	ALFKI	10643	1997-08-25 00:00:00.000	1115.46	10643	1997-08-25 00:00:00.000	1115.46	10643	1997-08-25 00:00:00.000	1115.46
2	ALFKI	10643	1997-08-25 00:00:00.000	1115.46	10643	1997-08-25 00:00:00.000	1115.46	10643	1997-08-25 00:00:00.000	1115.46
3	ALFKI	10643	1997-08-25 00:00:00.000	1115.46	10643	1997-08-25 00:00:00.000	1115.46	10643	1997-08-25 00:00:00.000	1115.46
4	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
5	ALFKI	10702	1997-10-13 00:00:00.000	353.94	10702	1997-10-13 00:00:00.000	353.94	10702	1997-10-13 00:00:00.000	353.94
6	ALFKI	10702	1997-10-13 00:00:00.000	353.94	10702	1997-10-13 00:00:00.000	353.94	10702	1997-10-13 00:00:00.000	353.94
7	ALFKI	10835	1998-01-15 00:00:00.000	920.53	10835	1998-01-15 00:00:00.000	920.53	10835	1998-01-15 00:00:00.000	920.53
8	ALFKI	10835	1998-01-15 00:00:00.000	920.53	10835	1998-01-15 00:00:00.000	920.53	10835	1998-01-15 00:00:00.000	920.53
9	ALFKI	10952	1998-03-16 00:00:00.000	531.62	10952	1998-03-16 00:00:00.000	531.62	10952	1998-03-16 00:00:00.000	531.62
10	ALFKI	10952	1998-03-16 00:00:00.000	531.62	10952	1998-03-16 00:00:00.000	531.62	10952	1998-03-16 00:00:00.000	531.62
11	ALFKI	11011	1998-04-09 00:00:00.000	961.21	11011	1998-04-09 00:00:00.000	961.21	11011	1998-04-09 00:00:00.000	961.21
12	ALFKI	11011	1998-04-09 00:00:00.000	961.21	11011	1998-04-09 00:00:00.000	961.21	11011	1998-04-09 00:00:00.000	961.21
13	ANATR	10308	1996-09-18 00:00:00.000	90.41	10308	1996-09-18 00:00:00.000	90.41	10308	1996-09-18 00:00:00.000	90.41
14	ANATR	10308	1996-09-18 00:00:00.000	90.41	10308	1996-09-18 00:00:00.000	90.41	10308	1996-09-18 00:00:00.000	90.41
15	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
16	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

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
    id,
    productid,
    date,
    value,
    SUM(value) OVER (PARTITION BY productid, YEAR(date), MONTH(date) ORDER
BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS
cumulative_value
    from product_history
ORDER by
    productid,
    YEAR(date),
    MONTH(date),
    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(ph_inner.value)
     FROM product_history ph_inner
     WHERE ph_inner.productid = ph.productid
     AND YEAR(ph_inner.date) = YEAR(ph.date)
     AND MONTH(ph_inner.date) = MONTH(ph.date)
     AND ph_inner.date <= ph.date) AS cumulative_value
FROM
    product_history ph
ORDER BY
```

```
    ph.productid,
    YEAR(ph.date),
    MONTH(ph.date),
    ph.date;
```

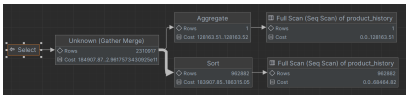
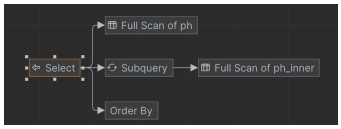
wersja dla Postgres

```
SELECT
    ph.id,
    ph.productid,
    ph.date,
    ph.value,
    (SELECT SUM(ph_inner.value)
     FROM product_history ph_inner
     WHERE ph_inner.productid = ph.productid
     AND extract (year from ph_inner.date) = extract (year from ph.date)
     AND extract (month from ph_inner.date) = extract (month from ph.date)
     AND ph_inner.date <= ph.date) AS cumulative_value
FROM
    product_history ph
ORDER BY
    ph.productid,
    extract (year from ph.date),
    extract (month from ph.date),
    ph.date;
```

Wersja dla SQLite

```
SELECT
    ph.id,
    ph.productid,
    ph.date,
    ph.value,
    (SELECT SUM(ph_inner.value)
     FROM product_history ph_inner
     WHERE ph_inner.productid = ph.productid
       AND strftime('%Y',ph_inner.date) = strftime('%Y',ph.date)
       AND strftime('%M',ph_inner.date) = strftime('%M',ph.date)
       AND ph_inner.date <= ph.date) AS cumulative_value
FROM
    product_history ph
ORDER BY
    ph.productid,
    strftime('%Y',ph.date),
    strftime('%M',ph.date),
    ph.date;
```

Dla żadnego SZBD zapytanie nie skończyło się w rozsądnym czasie więc przedstawiy analizę planu wykonania

MySQL	Postgres	SQLite
		

Jak widzimy plan wykonania dla mysql jest znacznie bardziej rozbudowany niż dla sqlite i postgres

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.

`Funkcje agregujące + order by`

Napisz polecenie które pokaże id produktu, nazwę produktu, cenę produktu, kategorię produktu, oraz maksymalną cenę produktu w danej kategori.

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

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

- Polecenie z wykorzystaniem podzapytania

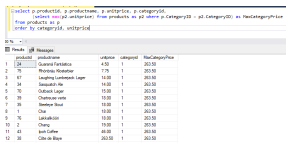
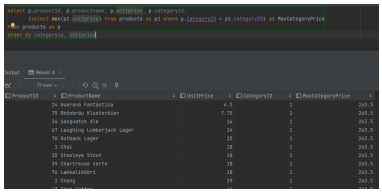
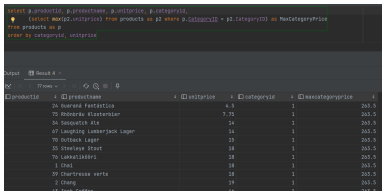
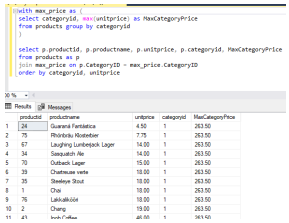
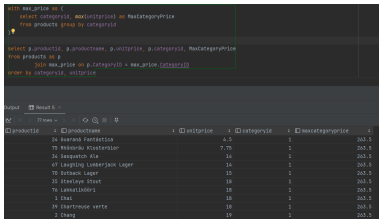
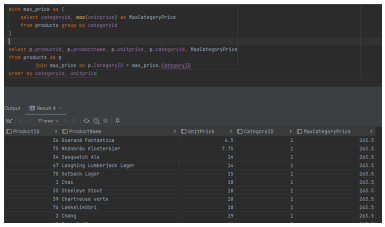
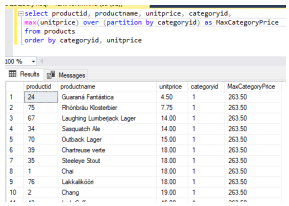
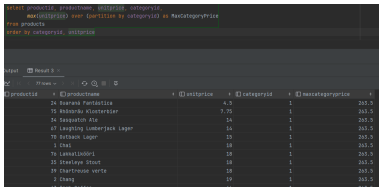
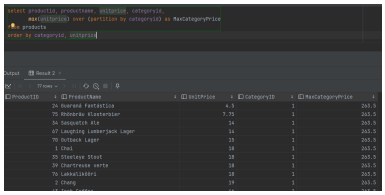
```
SELECT p.productid, p.productname, p.unitprice, p.categoryid,
       (SELECT MAX(p2.unitprice) FROM products p2 WHERE p2.categoryid =
p.categoryid) AS maxprice
FROM products p
ORDER BY p.categoryid, p.unitprice;
```

- Polecenie z wykorzystaniem join

```
with max_price as (
    select categoryid, max(unitprice) as MaxCategoryPrice
    from products
    group by categoryid
)

SELECT p.productid, p.productname, p.unitprice, p.categoryid,
MaxCategoryPrice
FROM products p
         JOIN max_prices ON p.categoryid = max_price.categoryid
ORDER BY p.categoryid, p.unitprice;
```

- Polecenie z wykorzystaniem funkcji okna

```
SELECT productid, productname, unitprice, categoryid,
       max(unitprice) OVER (PARTITION BY categoryid) AS MaxCategoryPrice
FROM products
ORDER BY categoryid, unitprice;
```

Zapytanie	MySQL	Postgres	SQLite
Podzapytanie			
Join			
Funkcja okna			

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