Filip Strózik 260377

Hurtownie Danych - laboratorium
Lista 3

Podstawy SQL: funkcje grupujące i okienkowe

Wstęp teoretyczny

Funkcje grupujące

Klauzula instrukcji `SELECT` dzieli wynik zapytania na grupy wierszy, zwykle w celu wykonania jednej lub więcej agregacji dla każdej grupy. Polecenie `SELECT – GROUP BY` zwraca jeden wiersz wyniku na grupę.
Stosuje się następującą składnię polecenia `GROUP BY`:

```
SELECT ...
FROM ...
WHERE ...
GROUP BY {
      column-expression
    | ROLLUP ( <group_by_expression> [ ,...n ] )
    | CUBE ( <group_by_expression> [ ,...n ] )
    | GROUPING SETS ( <grouping_set> [ ,...n ]  )
    | () --calculates the grand total
} [ ,...n ]
```

Różnice pomiędzy `ROLLUP`, `CUBE` i `GROUPING SETS`:
GROUP BY ROLLUP

GROUP BY ROLLUP (col_1, col_2, col_3, col_4, ..., col_n)
Wynik – podsumowania dla następujących zestawień kolumn:

- col_1, col_2, ..., col_n
- col_1, col_2, ..., col_{n-1}, NULL
- col_1, ..., col_{n-2}, NULL, NULL
- ...
- col_1, col_2, NULL, ..., NULL
- col_1, NULL, NULL, ..., NULL
- NULL, NULL, ..., NULL

GROUP BY CUBE

GROUP BY CUBE (col_1, col_2, col_3, col_4, ..., col_n)
Wynik – podsumowania dla wszystkich możliwych kombinacji kolumn
Ile jest możliwych podzbiorów zbioru n-elementowego?

GROUP BY GROUPING SETS

GROUP BY GROUPING SETS (...)
Wynik – podsumowania dla wszystkich wymienionych kombinacji kolumn

Funkcje okienkowe

Klauzula OVER określa podział i kolejność zestawu wierszy przed zastosowaniem powiązanej funkcji okna. Klauzula OVER definiuje okno (określony przez użytkownika zestaw wierszy), a następnie wylicza wartość dla każdego wiersza w oknie.

```
OVER (
       [ <PARTITION BY clause> ]
       [ <ORDER BY clause> ]
       [ <ROW or RANGE clause> ]
      )
```

Klauzulę OVER stosuje się do obliczania wartości zagregowanych, np. średnie ruchome, sumy bieżące, wyniki dla ostatnich *n* wierszy w każdej grupie.

PARTITION BY clause
Dzieli zbiór wyników zapytania na określone podgrupy. Funkcja okna jest stosowana do każdej podgrupy osobno i obliczenia uruchamiane są ponownie dla każdej podgrupy.

ORDER BY clause
Definiuje logiczną kolejność wierszy w każdej podgrupie wynikowego zbioru. Dla każdej podgrupy określa logiczną kolejność wykonywania obliczeń funkcji okna.

ROW or RANGE clause
Ograniczają wiersze w partycji, określając granice:
- ROWS – fizyczne ograniczenie liczby wierszy, np.
  - BETWEEN 2 PRECEEDING AND CURRENT ROW
- RANGE – logiczne ograniczenie liczby wierszy, np.
  - BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING

Obie klauzule wymagają klauzuli ORDER BY i na podstawie ustalonej kolejności determinują wynik.

Funkcje szeregujące pozwalają tworzyć ranking, przypisując odpowiednie miejsca kolejnym wynikom. Najczęściej stosowane są następujące funkcje szeregujące:

- **ROW_NUMBER** – nr pozycji
- **RANK** – ranking (to samo miejsce dla tych samych wartości)
- **DENSE_RANK** – ranking, numerowanie ciągłe
- **NTILE** – grupuje rekordy poprzez przypisanie tej samej wartości szeregującej członkom grupy

Źródła:
https://docs.microsoft.com/en-us/sql/t-sql/queries/select-group-by-transact-sql?view=sql-server-ver15
https://docs.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql?view=sql-server-ver15

Zadania do wykonania

Baza danych: **AdventureWorks**

Rozwiązania:

Zad. 1. Wykorzystanie funkcji grupujących (rollup, cube, grouping sets)

1. Przygotować zestawienie przedstawiające, ile pieniędzy wydali klienci na zamówienia na przestrzeni poszczególnych lat.
   Wykonaj zestawienie przy użyciu poleceń rollup, cube, grouping sets.

```
--1.1.1
SELECT
	ISNULL(CONCAT(per.FirstName,' ',per.LastName), ' ') AS Klient ,
	ISNULL(CONVERT(varchar, YEAR(soh.OrderDate)), ' ') AS Rok,
	SUM([TotalDue]) AS Kwota
FROM
	Sales.SalesOrderHeader soh
	JOIN Sales.Customer cust ON cust.CustomerID = soh.CustomerID
	JOIN Person.Person per ON per.BusinessEntityID = cust.PersonID
GROUP BY ROLLUP (YEAR(soh.OrderDate), CONCAT([FirstName], ' ' ,[LastName]))
ORDER BY 1;
```

	Klient	Rok	Kwota
1		2012	37675700.312
2		2013	48965887.9632
3		2011	14155699.525
4		2014	22419498.3157
5			123216786.1159
6	A. Leonetti	2014	1586.6583
7	A. Leonetti	2013	1814.1819
8	Aaron Adams	2013	130.3458
9	Aaron Alexander	2014	77.339
10	Aaron Allen	2012	3756.989
11	Aaron Baker	2014	1934.8329
12	Aaron Bryant	2013	148.0258
13	Aaron Butler	2014	16.5529
14	Aaron Campbell	2014	1276.8054
15	Aaron Carter	2014	44.1779

```sql
--1.1.2
SELECT
	ISNULL(CONCAT(per.FirstName,' ',per.LastName), ' ') AS Klient,
	ISNULL(CONVERT(varchar, YEAR(soh.OrderDate)), ' ') AS Rok,
	SUM([TotalDue]) AS Kwota
FROM Sales.SalesOrderHeader soh
JOIN Sales.Customer cust ON cust.CustomerID = soh.CustomerID
JOIN Person.Person per ON per.BusinessEntityID = cust.PersonID
GROUP BY CUBE (YEAR([OrderDate]),CONCAT([FirstName], ' ' ,[LastName]))
ORDER BY 1;
```

	Klient	Rok	Kwota
1			123216786.1159
2		2013	48965887.9632
3		2014	22419498.3157
4		2011	14155699.525
5		2012	37675700.312
6	A. Leonetti	2013	1814.1819
7	A. Leonetti	2014	1586.6583
8	A. Leonetti		3400.8402
9	Aaron Adams	2013	130.3458
10	Aaron Adams		130.3458
11	Aaron Alexander	2014	77.339
12	Aaron Alexander		77.339
13	Aaron Allen	2012	3756.989
14	Aaron Allen		3756.989
15	Aaron Baker	2014	1934.8329

```sql
--1.1.3
SELECT
	ISNULL(CONCAT(per.FirstName,' ',per.LastName), ' ') AS Klient,
	ISNULL(CONVERT(varchar, YEAR(soh.OrderDate)), ' ') AS Rok,
	SUM([TotalDue]) AS Kwota
FROM Sales.SalesOrderHeader soh
JOIN Sales.Customer cust ON cust.CustomerID = soh.CustomerID
JOIN Person.Person per ON per.BusinessEntityID = cust.PersonID
GROUP BY GROUPING SETS
(
	CONCAT([FirstName], ' ' ,[LastName]),
	YEAR([OrderDate]),
	(CONCAT([FirstName], ' ' ,[LastName]), YEAR([OrderDate])),
	()
)
ORDER BY 1;
```

	Klient	Rok	Kwota
1		2012	37675700.312
2		2013	48965887.9632
3		2011	14155699.525
4		2014	22419498.3157
5			123216786.1159
6	A. Leonetti	2014	1586.6583
7	A. Leonetti	2013	1814.1819
8	A. Leonetti		3400.8402
9	Aaron Adams	2013	130.3458
10	Aaron Adams		130.3458
11	Aaron Alexander		77.339
12	Aaron Alexander	2014	77.339
13	Aaron Allen	2012	3756.989
14	Aaron Allen		3756.989
15	Aaron Baker		1934.8329

2. Przygotować zestawienie przedstawiające łączną kwotę zniżek z podziałem na kategorie, produkty oraz lata.

```
SELECT
       pc.Name AS Kategoria,
       p.Name AS Produkt,
       ISNULL(CONVERT(varchar,YEAR(soh.OrderDate)),'') AS Rok,
    CAST(SUM(sod.LineTotal * sod.UnitPriceDiscount) AS DECIMAL(10,2)) AS Kwota
FROM
    Sales.SalesOrderHeader soh
    JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
    JOIN Production.Product p ON sod.ProductID = p.ProductID
    JOIN Production.ProductSubcategory psc ON p.ProductSubcategoryID = psc.ProductSub-
categoryID
    JOIN Production.ProductCategory pc ON psc.ProductCategoryID = pc.ProductCategoryID
GROUP BY ROLLUP (
       pc.Name,
       p.Name,
       YEAR(soh.OrderDate));
```

	Kategoria	Produkt	Rok	Kwota
1	Accessories	All-Purpose Bike Stand	2013	0.00
2	Accessories	All-Purpose Bike Stand	2014	0.00
3	Accessories	All-Purpose Bike Stand		0.00
4	Accessories	Bike Wash - Dissolver	2013	79.92
5	Accessories	Bike Wash - Dissolver	2014	26.94
6	Accessories	Bike Wash - Dissolver		106.87
7	Accessories	Cable Lock	2012	19.89
8	Accessories	Cable Lock	2013	3.41
9	Accessories	Cable Lock		23.30
10	Accessories	Fender Set - Mountain	2013	0.00
11	Accessories	Fender Set - Mountain	2014	0.00
12	Accessories	Fender Set - Mountain		0.00
13	Accessories	Hitch Rack - 4-Bike	2013	1846.55
14	Accessories	Hitch Rack - 4-Bike	2014	379.42
15	Accessories	Hitch Rack - 4-Bike		2225.98

Zad. 2. Wykorzystanie funkcji okienkowych (over, over partition by, row_number, rank, dense_rank, ntile)

1. Dla kategorii ‘Bikes’ przygotuj zestawienie prezentujące procentowy udział kwot sprzedaży produktów tej kategorii w poszczególnych latach w stosunku do łącznej kwoty sprzedaży dla tej kategorii. W zadaniu wykorzystaj funkcje okna.

   Wykonaj podobne zestawienia dla pozostałych kategorii.

```
SELECT
  Kategoria,
  [2011], [2012], [2013], [2014]
FROM
  (
    SELECT
      C.Name AS Kategoria,
      YEAR(H.OrderDate) AS Rok,
      ROUND(
        SUM(H.TotalDue) OVER(
          PARTITION BY C.Name, YEAR(H.OrderDate)
        ) * 100 / SUM(H.TotalDue) OVER(PARTITION BY C.Name),
        2
      ) AS Procent
    FROM
      Production.Product P
      JOIN Sales.SalesOrderDetail D on P.ProductID = D.ProductID
      JOIN Production.ProductSubcategory S on P.ProductSubcategoryID = S.ProductSub-
categoryID
      JOIN Production.ProductCategory C on S.ProductCategoryID = C.ProductCategoryID
      JOIN Sales.SalesOrderHeader H on D.SalesOrderID = H.SalesOrderID
    WHERE
      C.Name IN ('Accessories', 'Bikes', 'Clothing', 'Components')
  ) AS src
PIVOT (
  MAX(Procent)
  FOR Rok IN ([2011], [2012], [2013], [2014])
) AS piv
ORDER BY Kategoria;
```

	Kategoria	2011	2012	2013	2014
1	Accessories	5.62	26.20	48.52	19.66
2	Bikes	7.93	34.51	42.60	14.96
3	Clothing	4.62	36.11	45.82	13.45
4	Components	3.93	31.92	49.37	14.78

2. Przygotuj zestawienie prezentujące narastającą liczbę zamówień wykonanych przez klienta w poszczególnych latach. W zadaniu wykorzystaj funkcje okna.
   Ogranicz wynik do 10 najlepszych klientów.

```
SELECT TOP (100) *, DENSE_RANK() OVER(ORDER BY Razem DESC) AS Ranking
FROM (
SELECT
       CONCAT(P.FirstName, ' ', P.LastName) Klient,
       YEAR(H.OrderDate) Rok,
       COUNT(H.SalesOrderID) OVER (PARTITION BY C.CustomerID) Razem,
```

```
      COUNT(H.SalesOrderID) OVER (PARTITION BY C.CustomerID
            ORDER BY YEAR(H.OrderDate) RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT
ROW) Narastająco
FROM Sales.SalesOrderHeader H JOIN Sales.Customer C ON H.CustomerID = C.CustomerID
 JOIN Person.Person P ON P.BusinessEntityID = C.PersonID
 )
AS zamowienia
PIVOT (
MAX(Narastająco)
FOR Rok IN ("2011", "2012", "2013", "2014")
) Lata;
```

	Klient	Razem	2011	2012	2013	2014	Ranking
1	Dalton Perez	28	NULL	NULL	15	28	1
2	Mason Roberts	28	NULL	NULL	15	28	1
3	Ashley Henderson	27	NULL	NULL	9	27	2
4	Charles Jackson	27	NULL	NULL	14	27	2
5	Daniel Davis	27	NULL	NULL	11	27	2
6	Fernando Barnes	27	NULL	NULL	13	27	2
7	Hailey Patterson	27	NULL	NULL	15	27	2
8	Henry Garcia	27	NULL	NULL	16	27	2
9	Jason Griffin	27	NULL	NULL	15	27	2
10	Jennifer Simmons	27	NULL	NULL	11	27	2

Gdy nie ograniczymy to widać narastanie:

	Klient	Razem	2011	2012	2013	2014	Ranking
34	Sarah Simmons	16	NULL	NULL	6	16	5
35	Sierra Young	16	NULL	NULL	7	16	5
36	Aaron Con	12	2	7	11	12	6
37	Aidan Delaney	12	2	6	10	12	6
38	Andrew Dixon	12	2	6	10	12	6
39	Bernard Duerr	12	2	7	11	12	6
40	Blaine Dockter	12	2	7	11	12	6
41	Brannon Jones	12	2	7	11	12	6
42	Brian Goldstein	12	2	7	11	12	6
43	Carla Eldridge	12	2	7	11	12	6
44	Cheryl Herring	12	2	6	10	12	6
45	Conor Cunningh...	12	2	6	10	12	6
46	David Liu	12	2	7	11	12	6
47	Della Demott Jr	12	2	7	11	12	6
48	Diane Tibbott	12	2	7	11	12	6

3. Przygotuj zestawienie dla sprzedawców z podziałem na lata i miesiące prezentujące liczbę obsłużonych przez nich zamówień w ciągu roku, w ciągu roku narastająco oraz sumarycznie w obecnym i poprzednim miesiącu. W zadaniu wykorzystaj funkcje okna.

```
SELECT
  *,
  SUM("W miesiącu") OVER (
    PARTITION BY "Imię i nazwisko",
```

```
    Rok
    ORDER BY
      Miesiąc ROWS BETWEEN 1 PRECEDING
      AND CURRENT ROW
  ) "Obecny i poprzedni miesiąc"
FROM
  (
    SELECT
      DISTINCT CONCAT(per.FirstName, ' ', per.LastName) "Imię i nazwisko",
      YEAR(soh.OrderDate) Rok,
      MONTH(soh.OrderDate) Miesiąc,
      COUNT(soh.SalesOrderID) OVER (
        PARTITION BY CONCAT(per.FirstName, ' ', per.LastName),
        YEAR(soh.OrderDate),
        MONTH(soh.OrderDate)
      ) "W miesiącu",
      COUNT(soh.SalesOrderID) OVER (
        PARTITION BY CONCAT(per.FirstName, ' ', per.LastName),
        YEAR(soh.OrderDate)
      ) "W roku",
      COUNT(soh.SalesOrderID) OVER (
        PARTITION BY CONCAT(per.FirstName, ' ', per.LastName),
        YEAR(soh.OrderDate)
        ORDER BY
          MONTH(soh.OrderDate) RANGE BETWEEN UNBOUNDED PRECEDING
          AND CURRENT ROW
      ) "W roku narastająco"
    FROM
      Sales.SalesOrderHeader soh
      JOIN Sales.SalesPerson sp ON soh.SalesPersonID = sp.BusinessEntityID
      JOIN HumanResources.Employee E ON sp.BusinessEntityID = E.BusinessEntityID
      JOIN Person.Person per ON per.BusinessEntityID = E.BusinessEntityID
  ) AS Sprzedawcy
ORDER BY
        [Imię i nazwisko], Rok, Miesiąc;
```

	Imię i nazwisko	Rok	Miesiąc	W miesiącu	W roku	W roku narastająco	Obecny i poprzedni miesiąc
1	Amy Alberts	2012	6	3	7	3	3
2	Amy Alberts	2012	9	2	7	5	5
3	Amy Alberts	2012	12	2	7	7	4
4	Amy Alberts	2013	1	1	29	1	1
5	Amy Alberts	2013	2	1	29	2	2
6	Amy Alberts	2013	3	1	29	3	2
7	Amy Alberts	2013	4	2	29	5	3
8	Amy Alberts	2013	5	1	29	6	3
9	Amy Alberts	2013	6	5	29	11	6
10	Amy Alberts	2013	7	3	29	14	8
11	Amy Alberts	2013	8	1	29	15	4
12	Amy Alberts	2013	9	4	29	19	5
13	Amy Alberts	2013	10	4	29	23	8
14	Amy Alberts	2013	11	1	29	24	5
15	Amy Alberts	2013	12	5	29	29	6
16	Amy Alberts	2014	1	1	3	1	1
17	Amy Alberts	2014	2	1	3	2	2
18	Amy Alberts	2014	3	1	3	3	2
19	David Campbell	2011	5	5	28	5	5
20	David Campbell	2011	7	3	28	8	8
21	David Campbell	2011	8	7	28	15	10

4. Przygotuj zestawienie dla kategorii produktów prezentujące sumę maksymalnych cen produktów w poszczególnych podkategoriach należących danej tej kategorii. W zadaniu wykorzystaj funkcje okna.

```
--4
WITH pokategorie AS (
    SELECT
        pc.Name AS Kategoria,
        ps.Name AS Podkategoria,
        MAX(p.ListPrice) AS MaxPrice
    FROM
        Production.Product p
        JOIN Production.ProductSubcategory ps ON p.ProductSubcategoryID = ps.Prod-
uctSubcategoryID
        JOIN Production.ProductCategory pc ON ps.ProductCategoryID = pc.ProductCatego-
ryID
    GROUP BY
        pc.Name,
        ps.Name
)
SELECT
    Kategoria,
    ROUND(SUM(MaxPrice),2) AS 'Suma maks. cen'
FROM
    pokategorie
GROUP BY Kategoria;
```

	Kategoria	Suma maks. cen
1	Accessories	663.88
2	Bikes	9362.33
3	Clothing	408.94
4	Components	5539.77

5. Przygotuj ranking klientów w zależności od liczby zakupionych produktów. Porównaj rozwiązania uzyskane przez funkcje `rank` i `dense_rank`.

```
SELECT CONCAT(p.FirstName, ' ', p.LastName) AS Klient,
    SUM(sod.OrderQty) AS 'Liczba zakupionych produktów',
    RANK() OVER (ORDER BY SUM(sod.OrderQty) DESC) AS 'rank',
    DENSE_RANK() OVER (ORDER BY SUM(sod.OrderQty) DESC) AS 'dense rank'
FROM
    Sales.SalesOrderHeader soh
    JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
    JOIN Sales.Customer c ON soh.CustomerID = c.CustomerID
    JOIN Person.Person p ON c.PersonID = p.BusinessEntityID
GROUP BY CONCAT(p.FirstName, ' ', p.LastName);
```

	Klient	Liczba zakupionych produktów	rank	dense rank
1	Reuben D'sa	2737	1	1
2	Kevin Liu	2554	2	2
3	Marcia Sultan	2350	3	3
4	Holly Dickson	2313	4	4
5	Mandy Vance	2129	5	5
6	Richard Lum	2076	6	6
7	Della Demott Jr	1963	7	7
8	Sandra Maynard	1951	8	8
9	Anton Kirilov	1946	9	9
10	Ryan Calafato	1931	10	10
11	John Evans	1887	11	11
12	Yale Li	1843	12	12
13	Helen Dennis	1784	13	13
14	Margaret Vanderkamp	1782	14	14
15	Lola McCarthy	1776	15	15
16	Robert Vessa	1736	16	16
17	Joseph Castellucio	1708	17	17
18	Donna Carreras	1695	18	18
19	Kirk DeGrasse	1688	19	19

	Klient	Liczba zakupionych produktów	rank	dense rank
22	Nieves Vargas	1627	22	22
23	Robert Bernacchi	1606	23	23
24	Thierry D'Hers	1605	24	24
25	Shawn Demicell	1588	25	25
26	John Arthur	1579	26	26
27	Richard Bready	1578	27	27
28	Johnny Caprio	1578	27	27
29	Roger Harui	1558	29	28
30	Mary Gimmi	1545	30	29
31	Scott Culp	1532	31	30
32	Blaine Dockter	1488	32	31
33	Barbara Calone	1471	33	32
34	Judy Lew	1410	34	33
35	Robin McGuigan	1408	35	34
36	Mitch Kennedy	1386	36	35
37	François Ferrier	1382	37	36
38	Raul Casts	1363	38	37
39	Krishna Sunkammurali	1351	39	38
40	Stacey Cereghino	1344	40	39

6. Przygotuj ranking produktów w zależności od średniej liczby sprzedanych sztuk. Wyróżnij 3 (prawie równoliczne) grupy produktów: sprzedających się najlepiej, średnio i najsłabiej.

```
SELECT
  "Produkt",
  DENSE_RANK() OVER (
    ORDER BY
      Srednia DESC
  ) Miejsce,
  CASE NTILE(3) OVER (
    ORDER BY
      Srednia DESC
  ) WHEN 1 THEN 'najlepiej' WHEN 2 THEN 'średnio' WHEN 3 THEN 'najsłabiej' END AS
Grupa,
  Srednia
FROM
  (
    SELECT
      DISTINCT P.Name "Produkt",
      AVG(
        CAST(D.OrderQty AS FLOAT)
      ) OVER (PARTITION BY D.ProductID) Srednia
    FROM
      Sales.SalesOrderDetail D
      JOIN Production.Product P ON D.ProductID = P.ProductID
  ) AS ListaProduktow;
```

	Produkt	Miejsce	Grupa	Srednia
1	Full-Finger Gloves, L	1	najlepiej	9.28021978021978
2	Classic Vest, S	2	najlepiej	6.22727272727273
3	Full-Finger Gloves, M	3	najlepiej	6.14484679665738
4	ML Headset	4	najlepiej	5.99090909090909
5	Mountain Bike Socks, M	5	najlepiej	5.88829787234043
6	Women's Mountain Shorts, S	6	najlepiej	5.08641975308642
7	Women's Tights, S	7	najlepiej	4.90995260663507
8	Women's Tights, L	8	najlepiej	4.8140589569161
9	Women's Mountain Shorts, L	9	najlepiej	4.58840169731259
10	Men's Bib-Shorts, M	10	najlepiej	4.57790368271955
11	Men's Sports Shorts, M	11	najlepiej	4.50883392226148
12	Short-Sleeve Classic Jersey, XL	12	najlepiej	4.27433628318584
13	Minipump	13	najlepiej	4.23220973782772
14	Cable Lock	14	najlepiej	4.18076923076923
15	Classic Vest, M	15	najlepiej	4.11531531531532
16	Racing Socks, L	16	najlepiej	4.09437086092715
17	Long-Sleeve Logo Jersey, L	17	najlepiej	4.03180428134557
18	Hitch Rack - 4-Bike	18	najlepiej	3.97738693467337
19	ML Mountain Seat/Saddle	19	najlepiej	3.8695652173913

	Produkt	Miejsce	Grupa	Srednia
82	Touring-3000 Yellow, 62	82	najlepiej	2.64576802507837
83	LL Road Frame - Black, 52	83	najlepiej	2.635
84	LL Mountain Frame - Black, 42	84	najlepiej	2.61111111111111
85	Racing Socks, M	85	najlepiej	2.60876897133221
86	Mountain-100 Silver, 44	86	najlepiej	2.5793991416309
87	Road-350-W Yellow, 40	87	najlepiej	2.57766143106457
88	Road-450 Red, 58	88	najlepiej	2.57522123893805
89	Touring-1000 Yellow, 60	89	najlepiej	2.57274826789838
90	Hydration Pack - 70 oz.	90	srednio	2.5707635009311
91	HL Road Pedal	91	srednio	2.56060606060606
92	Touring-1000 Blue, 60	92	srednio	2.55125284738041
93	ML Road Frame-W - Yellow, 48	93	srednio	2.54
94	Mountain-100 Black, 48	94	srednio	2.53497942386831
95	Road-650 Red, 52	95	srednio	2.52727272727273
96	LL Bottom Bracket	96	srednio	2.52
97	ML Road Frame-W - Yellow, 38	97	srednio	2.51404494382022
98	LL Touring Frame - Yellow, 44	98	srednio	2.51282051282051
99	LL Touring Frame - Blue, 50	99	srednio	2.50793650793651
100	Bike Wash - Dissolver	100	srednio	2.50113036925396

Wnioski:

Zadanie 1

Funkcje grupujące w SQL, takie jak ROLLUP, CUBE i GROUPING BY SETS, pozwalają na grupowanie i agregowanie danych w różne sposoby. Dodatkowo, unikamy pisania wielu zapytań o różne dane, które łączmy za pomocą UNION.

Funkcja ROLLUP tworzy hierarchię sumaryczną, agregując dane według kolejnych poziomów grupowania. Po wykonaniu funkcji ROLLUP, wynikowe dane są pogrupowane według każdej z kolumn z podsumowaniem wartości dla każdej grupy. Na końcu wynikowe dane są dodatkowo pogrupowane według wszystkich kolumn z podsumowaniem całkowitym.

Funkcja CUBE tworzy wszystkie możliwe kombinacje grupowania, tworząc tabele przestawne z wszystkimi możliwymi sumarycznymi wartościami. Po wykonaniu funkcji CUBE, wynikowe dane są pogrupowane według każdej z kolumn, a następnie wyświetla sumę wartości dla każdej kombinacji.

GROUPING BY SETS umożliwia grupowanie danych na podstawie różnych zbiorów kolumn, co pozwala na tworzenie bardziej skomplikowanych zapytań agregujących.
Dzięki tym funkcjom programiści SQL mogą tworzyć bardziej elastyczne i złożone zapytania, które mogą skutecznie analizować duże ilości danych w bazach danych.

Analizując wyniki pierwszej kwerendy można zauważyć, że od roku 2011 do 2013 łącznie klienci wydawali coraz więcej, była zauważalna tendencja wzrostowa, natomiast w 2014 roku łączne wydatki klientów znacząco zmalały. Pogrupowanie wydatków ze względu na klienta oraz rok daje dokładniejszy pogląd na wydatki danego klienta w sklepie a podsumowanie wskazuje na punkt odniesienia wydatków danego klienta.

Kwerenda pokazująca zestawienie łącznej kwoty zniżek z podziałem na poszczególne kategorie produkty i lata przedstawia informację na jakie produkty zostały wprowadzone zniżki oraz czy zniżki wraz z upływem lat były coraz niższe czy coraz wyższe.

Zadanie 2

Funkcje okienkowe w SQL pozwalają na wykonywanie operacji analitycznych na grupach wierszy, bez konieczności grupowania ich za pomocą klauzuli GROUP BY. Funkcje okienkowe umożliwiają na wykonywanie bardziej złożonych analiz na danych, które wcześniej były trudne lub niemożliwe do wykonania w SQL. Pozwalają na wykonanie różnych operacji analitycznych na danych, takich jak sumowanie, obliczanie średniej, znajdowanie wartości maksymalnej lub minimalnej itp., a wszystko to bez konieczności stosowania dodatkowych kroków, takich jak tworzenie podzapytań. Dodatkowo zwiększją wydajność, ponieważ wszytsko odbywa się w ramach jednego połączenia z bazą.

1. Biorąc pod uwagę wszystkie kategorie i lata to widać, że rok 2013 był najlepszym rokiem dla każdej kategorii użycie PIVOT'a pozwala też łatwo sprecyzować która Kategoria była najlepsza lub najgorsza w danym roku.
2. Jeżeli bierzemy pod uwagę tylko top 10 najlepszych klientów, którzy sumarycznie najwięcej złożyli zamówień to możemy dojść do spostrzeżenia że są to klienci, którzy zaczęli kupować w sklepie po jego wstępnym rozwinięciu, mianowicie po 2012 roku. Z poprzednich zadań wiemy, że rok 2013 był najlepszym rokiem dlatego, też można się domyśleć, że sklep w tym czasie zdobył bardzo dobrych klientów.
3. Wyniki analizy danych wykazały, że pracownicy firmy odnotowują wzrost sprzedaży z roku na rok. Jednakże, zauważono, że liczba pracowników w firmie w poszczególnych miesiącach jest różna, co zostało również odnotowane w poprzednich raportach. Właściciele firmy mogą zastanawiać się, jakie działania podjąć w tej sytuacji, np. zezwolić pracownikom na udzielanie większych rabatów na produkty w okresach, w których liczba zamówień jest mniejsza.

4. Zestawienie jasno wskazuje że sklep zajmujący się sprzedażą rowerów będzie sumarycznie najdroższe miał rowery, po czym komponenty do tych rowerów, na przykład części zamienne, następnie akcesoria, a na końcu ubrania.
5. Zestawienie w jasny sposób pokazuje ranking najlepszych klientów ze względu na liczbę zamówionych produktów widać różnicę między liczeniem Kolejności w rankingu za pomocą RANK(), a za pomocą DENSE_RANK(). RANK() w przypadku wystąpienia wierszy o równych wartościach przypisuje wierszom ten sam numer porządkowy a kolejny numer zostaje pominięty. DENSE_RANK() w przypadku wystąpienia wierszy o równych wartościach przypisuje wierszom ten sam numer porządkowy a kolejny numer nie zostaje pominięty.
6. Kolejne zestawienie pokazuje ranking średnie ilości sprzedanych produktów z podziałem na czy równoliczne grupy, najlepiej sprzedających się średnio sprzedających się i najsłabiej sprzedających się. wyprosty sposób można byłoby planować kolejne zamówienia w celu zaopatrzenia sklepu, w taki sposób aby większy priorytet nadawać produktom z grupy średnio oraz najlepiej sprzedających się, a rozważyć ograniczenie ilości zamówionych produktów z grupy najsłabiej sprzedających się.

Uwaga!

- Sprawozdanie, bez wniosków podsumowujących aspekt zagadnień analizowanych na zajęciach laboratoryjnych i zawartych w sprawozdaniu, jest automatycznie oceniane negatywnie!