ITBA
Instituto Tecnológico de Buenos Aires

Diseño Lógico de DW

SQL para OLAP

Funciones de ventana

Alejandro Vaisman
Instituto Tecnológico de Buenos Aires, Argentina
avaisman@itba.edu.ar

El Cubo de Datos Relacional

- El modelo relacional no es lo mejor para representar datos multidimensionales
- La pregunta es: ¿Cómo representamos el data cube utilizando tablas (2 dimensiones)?
- Ej: un cubo de dos dimensiones, Cliente y Producto, con todas las posibles agregaciones: (Cliente), (Producto), (Cliente, Producto), ()

	c1	c2	c3	TotalBy Product
p1	100	105	100	305
p2	70	60	40	170
p3	30	40	50	120
TotalBy Customer	200	205	190	595

ProductKey	CustomerKey	SalesAmount
p1	c1	100
p1	c2	105
p1	c3	100
p2	c1	70
p2	c2	60
p2	c3	40
p3	c1	30
p3	c2	40
p3	c3	50

- El fact de la derecha no permite almacenar agregaciones. **Cómo representamos el cubo de la izquierda?**

El Cubo de Datos Relacional

- Utilizar consultas SQL para obtener una tabla como la de la derecha

	c1	c2	c3	TotalBy Product
p1	100	105	100	305
p2	70	60	40	170
p3	30	40	50	120
TotalBy Customer	200	205	190	595

ProductKey	CustomerKey	SalesAmount
p1	c1	100
p2	c1	70
p3	c1	30
NULL	c1	200
p1	c2	105
p2	c2	60
p3	c2	40
NULL	c2	205
p1	c3	100
p2	c3	40
p3	c3	50
NULL	c3	190
p1	NULL	305
p2	NULL	170
p3	NULL	120
NULL	NULL	595

El Cubo de Datos Relacional

- Conceptualmente: unión de todas las posibles sentencias SQLque definen las posibles vistas del cubo
- Los SGBD no ejecutan cada sentencia individualmente, hacen algo más inteligente

```
SELECT    ProductKey, CustomerKey, SalesAmount
FROM      Sales
    UNION
SELECT    ProductKey, NULL, SUM(SalesAmount)
FROM      Sales
GROUP BY ProductKey
    UNION
SELECT    NULL, CustomerKey, SUM(SalesAmount)
FROM      Sales
GROUP BY CustomerKey
    UNION
SELECT    NULL, NULL, SUM(SalesAmount)
FROM      Sales
```

ProductKey	CustomerKey	SalesAmount
p1	c1	100
p2	c1	70
p3	c1	30
NULL	c1	200
p1	c2	105
p2	c2	60
p3	c2	40
NULL	c2	205
p1	c3	100
p2	c3	40
p3	c3	50
NULL	c3	190
p1	NULL	305
p2	NULL	170
p3	NULL	120
NULL	NULL	595

El Cubo de Datos en SQL

- SQL tiene sentencias para expresar el cálculo del cubo de datos en forma concisa

```
SELECT    ProductKey, CustomerKey, SalesAmount
FROM      Sales
    UNION
SELECT    ProductKey, NULL, SUM(SalesAmount)
FROM      Sales
GROUP BY ProductKey
    UNION
SELECT    NULL, CustomerKey, SUM(SalesAmount)
FROM      Sales
GROUP BY CustomerKey
    UNION
SELECT    NULL, NULL, SUM(SalesAmount)
FROM      Sales
```

```
SELECT    ProductKey, CustomerKey, SUM(SalesAmount)
FROM      Sales
GROUP BY CUBE(ProductKey, CustomerKey)
```

```
SELECT    ProductKey, CustomerKey, SUM(SalesAmount)
FROM      Sales
GROUP BY GROUPING SETS((ProductKey, CustomerKey),(ProductKey),(CustomerKey),())
```

El Cubo de Datos en SQL

- También se pueden computar agregados que permiten realizer acumulaciones parciales. Las siguientes expresiones son equivalentes:

```
SELECT    ProductKey, CustomerKey, SUM(SalesAmount)
FROM      Sales
GROUP BY ROLLUP(ProductKey, CustomerKey)
```

```
SELECT    ProductKey, CustomerKey, SUM(SalesAmount)
FROM      Sales
GROUP BY GROUPING SETS((ProductKey,CustomerKey),(ProductKey),())
```

Group BY ROLLUP

ProductKey	CustomerKey	SalesAmount
p1	c1	100
p1	c2	105
p1	c3	100
p1	NULL	305
p2	c1	70
p2	c2	60
p2	c3	40
p2	NULL	170
p3	c1	30
p3	c2	40
p3	c3	50
p3	NULL	120
NULL	NULL	595

Group BY CUBE

ProductKey	CustomerKey	SalesAmount
p1	c1	100
p2	c1	70
p3	c1	30
NULL	c1	200
p1	c2	105
p2	c2	60
p3	c2	40
NULL	c2	205
p1	c3	100
p2	c3	40
p3	c3	50
NULL	c3	190
NULL	NULL	595
p1	NULL	305
p2	NULL	170
p3	NULL	120

Actividad

- Utilizar la base de datos de MusicBrainz disponible en campus
- Elegir algunos atributos y definer un cubo, por ejemplo

```
SELECT position, artist_credit, medium, count(*)
FROM  track1
GROUP BY CUBE (position, artist_credit, medium)
```

1. Utilizando EXPLAIN, verificar que efectivamente, Postgres calcula el cubo en forma más eficiente que la simple union de comandos.
2. Comprobarlo experimentalmente, ejecutando la consulta como UNION de los GROUP BYs
3. Repetir lo anterior para otros cubos (se puede realizar el join con la table artistcredit para generar más cubos)

SQL Windows Functions

- Permiten realizar operaciones sobre cubos en forma eficiente y flexible
- Típico uso: en dashboards, para calculary
  - YTD
  - Promedios móviles
  - Ránkings

- Tres tipos de funciones:
  - Window Partitioning
  - Window Ordering
  - Window Framing

Window Partitioning

- Permite comparar datos detallados con valores agregados
- Ej. : Calcular la relevancia de cada cliente con respecto a las ventas de cada producto

Query SQL

```
SELECT ProductKey, CustomerKey, SalesAmount,
   MAX(SalesAmount) OVER (PARTITION BY  ProductKey)
FROM Sales
```

- Tres primeras columnas salen de Sales
- 4ta columna:
  - Para cada tupla se define la partición con todas las tuplas del mismo producto
  - SalesAmount se agrega sobre esta Ventana usando MAX

Resultado

ProductKey	CustomerKey	SalesAmount	MaxAmount
p1	c1	100	105
p1	c2	105	105
p1	c3	100	105
p2	c1	70	70
p2	c2	60	70
p2	c3	40	70
p3	c1	30	50
p3	c2	40	50
p3	c3	50	50

Window Partitioning

- Permite comparar datos detallados con valores agregados
- Ej. : Calcular la relevancia de cada cliente con respecto a las ventas de cada producto

Query SQL

```
SELECT ProductKey, CustomerKey, SalesAmount,
  MAX(SalesAmount) OVER  w
FROM Sales
WINDOW w AS (PARTITION BY productkey) ;
```

- Forma alternativa de escribir esta consulta

Resultado

ProductKey	CustomerKey	SalesAmount	MaxAmount
p1	c1	100	105
p1	c2	105	105
p1	c3	100	105
p2	c1	70	70
p2	c2	60	70
p2	c3	40	70
p3	c1	30	50
p3	c2	40	50
p3	c3	50	50

Window Partitioning

- Otro ejemplo comparando con totales

Query SQL

```
SELECT customerkey, productkey,
ROUND(Avg(salesamount::numeric)  OVER everything,2) as total,
ROUND(Avg(salesamount::numeric)  OVER bycust ,2) as  bycust,
ROUND(Avg(salesamount::numeric) OVER byprod,2) as byprod
FROM Sales
WINDOW everything AS (),
bycust  AS (PARTITION BY customerkey),
byprod AS (PARTITION by productkey);
```

Resultado

Data Output | Explain | Messages | Notifications

	customerkey integer	productkey integer	total numeric	bycust numeric	byprod numeric
1	975	24	593.04	515.22	89.35
2	975	51	593.04	515.22	1076.68
3	975	70	593.04	515.22	287.10
4	975	57	593.04	515.22	323.43
5	975	74	593.04	515.22	187.12
6	975	40	593.04	515.22	431.21
7	975	59	593.04	515.22	1313.17
8	975	24	593.04	515.22	89.35
9	975	11	593.04	515.22	335.05
10	975	71	593.04	515.22	473.26

Window Ordering

- Permite ordenar las filas dentro de una partición
- Útil para computar ránkings
- Ej. : ¿Cómo rankea cada producto en las ventas de cada cliente?

Query SQL

```
SELECT ProductKey, CustomerKey, SalesAmount,
 ROW_NUMBER() OVER (PARTITION BY CustomerKey
ORDER BY SalesAmount DESC)
FROM Sales
```

- Se evalúa primero una Ventana con todas las tuplas de c1 ordenadas por las ventas a c1
- El producto de mayos venta a c1 es p1

Resultado

Product Key	Customer Key	Sales Amount	RowNo
p1	c1	100	1
p2	c1	70	2
p3	c1	30	3
p1	c2	105	1
p2	c2	60	2
p3	c2	40	3
p1	c3	100	1
p3	c3	50	2
p2	c3	40	3

Window Ordering

- Otro ejemplo con quartiles

Query SQL

```
SELECT DISTINCT  customerkey,
ntile(4) OVER (PARTITION BY customerkey ORDER BY
                    salesamount) as quartile, salesamount
FROM sales
ORDER BY quartile desc;
```

Resultado

Data Output | Explain | Messages | Notifications

	customerkey integer	quartile integer	salesamount money
1	975	4	$877.50
2	975	4	$912.00
3	975	4	$931.00
4	975	4	$950.00
5	975	4	$1,060.00
6	975	4	$1,104.00
7	975	4	$1,650.60
8	975	4	$1,925.00
9	975	4	$2,640.00

Window Ordering

- Computando ventas del mes anterior

Query SQL

```
WITH MonthSales AS
  (SELECT   D.Monthnumber as month, D.year as year,
            SUM(SalesAmount) as monthTot
   FROM Sales S join  Date D on d.datekey=s.orderdatekey
   GROUP BY D.Monthnumber, D.year
   ORDER BY D.year,  D.Monthnumber)
SELECT month, Year, monthTot,
          Lag(monthTot, 1) OVER (ORDER BY  Year, Month)
FROM MonthSales
```

Resultado

Data Output | Explain | Messages | Notifications

	month smallint	year integer	monthtot money	lag money
1	7	2016	$27,246.10	[null]
2	8	2016	$23,104.98	$27,246.10
3	9	2016	$20,582.40	$23,104.98
4	10	2016	$33,991.57	$20,582.40
5	11	2016	$44,365.42	$33,991.57
6	12	2016	$42,559.41	$44,365.42
7	1	2017	$57,187.26	$42,559.41
8	2	2017	$36,275.14	$57,187.26
9	3	2017	$37,100.85	$36,275.14
10	4	2017	$47,209.05	$37,100.85
11	5	2017	$49,093.30	$47,209.05
12	6	2017	$32,907.67	$49,093.30
13	7	2017	$49,758.38	$32,907.67

Window Framing

- Define el tamaño de la partición
- Usado para computar estadísticas sobre series de tiempo
- Ej. : Calcular el promedio móvil de tres meses de las ventas por producto

Query SQL

```
SELECT ProductKey, Year, Month,  SalesAmount ,
AVG(SalesAmount)  OVER (PARTITION BY ProductKey
   ORDER BY Year, Month ROWS 2  PRECEDING) AS
   MovAvg
FROM Sales
```

- Para cada tupla abre una Ventana con las tuplas del producto actual
- Luego, ordena la Ventana por año y mes, computa el promedio sobre la tuple actual y las dos anteriores (ROWS TWO PRECEDING), si existen

Resultado

Product Key	Year	Month	Sales Amount	MovAvg
p1	2011	10	100	100
p1	2011	11	105	102.5
p1	2011	12	100	101.67
p2	2011	12	60	60
p2	2012	1	40	50
p2	2012	2	70	56.67
p3	2012	1	30	30
p3	2012	2	50	40
p3	2012	3	40	40

Window Framing

- En general se requiere un preproceso
- Por ejemplo, Northwind, promedio móvil de 3 días, de ventas por cliente
- Hay varias compras del mismo cliente el mismo día, hay que agregarlas primero

Query SQL

```
WITH dailySales AS
(SELECT  S.customerkey as customer, D.date as day,
SUM(SalesAmount) as dayTot
FROM Sales S join  Date D on d.datekey=s.orderdatekey
GROUP BY S.customerkey, D.date
ORDER BY  S.customerkey,D.date)

SELECT customer, day, daytot, avg(daytot) over w
FROM dailySales
WINDOW w AS (PARTITION BY customer ORDER BY
day ROWS 2 PRECEDING);
```

Resultado

57

Data Output | Explain | Messages | Notifications

	customer integer	day date	daytot money	sum money
1	975	2016-12...	$1,119.90	$1,119.90
2	975	2016-12...	$2,122.92	$3,242.82
3	975	2017-03...	$180.48	$3,423.30
4	975	2017-03...	$1,272.00	$3,575.40
5	975	2017-03...	$3,163.20	$4,615.68
6	975	2017-04...	$155.00	$4,590.20
7	975	2017-05...	$880.50	$4,198.70
8	975	2017-07...	$2,285.00	$3,320.50
9	975	2017-07...	$353.20	$3,518.70
10	975	2017-08...	$2,054.00	$4,692.20
11	975	2017-10...	$378.00	$2,785.20
12	975	2017-12...	$2,146.86	$4,578.86

Read-only column

Window Framing

- Otro ejemplo: YTD
- Ej. : Calcular las ventas Year-To-Date por producto

Query SQL

```
SELECT ProductKey, Year, Month, SalesAmount ,
  SUM (SalesAmount) OVER (PARTITION BY ProductKey,
  Year ORDER BY Month ROWS UNBOUNDED
  PRECEDING) AS YTD
FROM Sales
```

- Para cada tupla abre una Ventana con las tuplas del product y año actuals, ordenado por mes
- Áplica la función SUM a todas las tuplas anteriores a la actual, dentro de la partición (ROWS TWO PRECEDING)

Resultado

Product Key	Year	Month	Sales Amount	YTD
p1	2011	10	100	100
p1	2011	11	105	205
p1	2011	12	100	305
p2	2011	12	60	60
p2	2012	1	40	40
p2	2012	2	70	110
p3	2012	1	30	30
p3	2012	2	50	80
p3	2012	3	40	120

Actividad 1

- Utilizar la base de datos de Northwind DW
- Modificar las consultas de los ejemplos de Window Functions, de acuerdo al esquema de NowrthwindDW, y ejecutarlas.

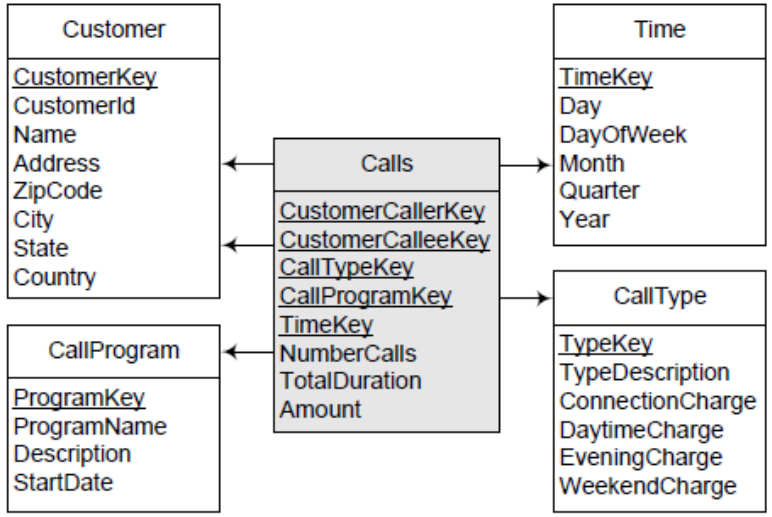
Actividad 2

- Realizar el TP-03

Actividad 3

Dado el DW Estrella de la figura, escribir en SQL las siguientes consultas

- Monto total por programa en 2018
- Duración total de las llamadas realizadas por clientes de Bruselas en 2019
- Cantidad total de llamadas de fin de semana realizadas por clientes de Bruselas a clientes en Amberes en 2018.
- Duración total de las llamadas internacionales iniciadas por clientes en Bélgica en 2019.



Actividad 4

Dado el DW Snowflake de la figura, escribir en SQL las siguientes consultas

- Cantidad total de kilómetros realizados por los trenes Alstom durante 2018 partiendo de estaciones francesas o belgas.
- Duración total de los viajes internacionales durante 2018, es decir, viajes que salen de una estación ubicada en un país y llegan a una estación ubicada en otro país.