VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Aggregate Functions

Lecturer : Dr. Pavle Mogin

SWEN 432
Advanced Database Design and Implementation

Plan for Aggregate Functions Topic

- Motivation to discuss aggregate functions
- A classification of aggregates
  - Distributive aggregates
  - Algebraic aggregates
  - Holistic aggregates

Motivation

- OLAP queries predominantly rely on aggregates
- Computing aggregates is costly since it requires sorting
- Reusing finer grained aggregates to compute coarser grained aggregates is a nice idea
- It particularly applies to computing roll-ups
- But this idea is not always applicable
- Understanding mechanisms in computing aggregates can help to extend usefulness of this idea

A Two Dimensional Set of Values

- Consider a two-dimensional set of values

$$V = \{X_{ij} \mid i = 1,\ldots, n; j = 1,\ldots, m_i\}$$

- The set V contains n subsets and each of these contains m_i values

- Suppose we need to compute:
  - m sub-aggregates by applying an aggregate function on each subset $\{X_i \mid i = 1,\ldots, n\}$, and
  - A global aggregate of the same kind by applying an appropriate aggregate functions (possibly different from one applied to compute sub-aggregates) on sub-aggregates

An Example Set of Values

- Suppose:
  - i is *City*,
  - j is *ProdType*,
  - $n = 3$,
  - $m_1 = 3$,
  - $m_2 = 2$,
  - $m_3 = 4$
- For i = *Auckland* and j = *Jeans*, $X_{ij} = 10$
- For i = *Wellington* and j = *Socks*, $X_{ij} = 3$

Sale_By_City_ProdType

City	ProdType	Amnt
Auckland	Jeans	10
Auckland	Shoes	11
Auckland	Pajamas	12
Christchurch	Jeans	5
Christchurch	Shoes	6
Wellington	Jeans	7
Wellington	Shoes	8
Wellington	Pajamas	9
Wellington	Socks	3

Distributive Aggregate Functions

- An aggregate function $F(\,)$ is distributive if there is a function $G(\,)$ such that

$$F(\{X_{ij}\}) = G(\{F(\{X_j \mid j = 1, \ldots, m_i\})_i \mid i = 1, \ldots, n\})$$

- The formula above says that function $F(\,)$ is applied on each of n subsets producing aggregates

$$Y_i = F(\{X_j \mid j = 1, \ldots, m_i)\}),$$

and then is $G(\,)$ applied on the n intermediate results

$$\{Y_i \mid i = 1, \ldots, n\}$$

to produce the aggregate $F(\{X_{ij}\})$ of a coarser granularity

- `SUM(), COUNT( ), MIN( ), MAX( )` are distributive aggregate functions
  - SUM(X_{ij}) = SUM(SUM(X_j)$_i$)
  - COUNT(X_{ij}) = SUM(COUNT(X_j)$_i$)
  - MAX(X_{ij}) = MAX(MAX(X_j)$_i$)

Examples of Distributive Functions

```
CREATE MATERIALIZED VIEW Sale_By_City AS
SELECT City, SUM(Amnt) AS SubTot
FROM Sale_By_City_ProdType
GROUP BY City;
```

Sale_By_City

Auck	*33*
Chr	*11*
Well	*27*

```
SELECT SUM(SubTot) AS Total_Sale
FROM Sale_By_City;
```

Total_Sale

71

SubTot

```
CREATE TABLE Max_By_City AS
SELECT City, MAX(Amnt) SubMax
FROM Sale_By_City_ProdType
GROUP BY City;
```

Max_By_City

Auck	*12*
Chr	*6*
Well	*9*

```
SELECT MAX(SubMax) AS Max_Sale
FROM Max_By_City;
```

Max_Sale

12

SubMax

Algebraic Aggregate Functions

- An aggregate function $F\,()$ is algebraic if there is a p-tuple function $G()$ and a function $H\,()$ such that

$$F(\{X_{ij}\}) = H\,(\{G(\{X_j \mid j = 1,\ldots, m_i\,)_i\})\mid i = 1,\ldots, n\,\})$$

- Function $G(\,)$ computes a p-tuple for each of n subsets, and then is function $H()$ applied on all these p-tuples to produce the courser aggregate
- `AVG( ), STDEV( )`, center_of_mass() are algebraic functions
- For each operation, p is a constant
- Each p-tuple is produced by computing a sub-aggregate and all are used to compute the global aggregate

Average as an Algebraic Function

```
CREATE MATERIALIZED View Sale_By_City_Avg AS
SELECT City, COUNT(*)AS my_count,
AVG(Amnt)AS my_avg
FROM Sale_By_City_ProdType
GROUP BY City;
```

Sale_By_City_Avg

City	*my_count*	*my_avg*
Auck	*3*	*11.00*
Chr	*2*	*5.50*
Well	*4*	*6.75*

```
SELECT SUM(my_avg*my_count)/SUM(my_count)
AS Total_Avg
FROM Sale_By_City_Avg;
```

Total_Avg

7.89

- Note, for the average function, *p = 2*,
- When computing `AVG` sub-aggregates, we have to produce and store a two tuple:
  - Either an intermediate (`SUM, COUNT`), or
  - An intermediate (`AVG, COUNT`)

Holistic Aggregate Functions

- An aggregate function $F()$ is holistic if there is no constant storage needed to describe a sub-aggregate
- There is no constant k ($k \geq 1$), such that a k - tuple characterizes the computation of sub-aggregates

$$F(\{X_j \mid j = 1, \ldots, m_i\})_i$$

- Usually, each sub-aggregate is characterized by m_i values,

  hence all values are needed to compute the super-aggregate

- Most common holistic aggregate functions are:
  – Median(),
  – MostFrequent() (called also Mode()), and
  – Rank()
  – MaxN(), or TopN()

Median As a Holistic Aggregate Ffunction

Median_By_City

City	List of values	Med
Auck	10, 11,12	11.0
Chr	6, 7	6.5
Well	3,7,8,9	7.5

Global_Median

8.0

To compute global median, one needs all Amnt values

RANK Function

- The RANK function returns the position of a row within its partition
- Rows are ordered in the partition according to a ranking criteria
- Rank of a row is expressed as an ordinal number of the row within the partition
  - All rows with the same value of the ranking criteria are designated the same rank
  - If n (≥ 1) rows are ranked at position r, then the first next row is ranked at position $r + n$
  - DENSE_RANK function generates ranks without gaps
- To compute the overall ranks, we need all source data, hence it is holistic

An Example Set of Values

Sale_By_City_ProdType

City	ProdType	Amnt	City	ProdType	Amnt
Auckland	Jeans	10	Christchurch	Hats	12
Auckland	Shoes	11	Wellington	Jeans	7
Auckland	Pajamas	12	Wellington	Shoes	8
Auckland	Socks	12	Wellington	Pajamas	13
Auckland	Jackets	6	Wellington	Socks	13
Auckland	Hats	12	Wellington	Jackets	7
Auckland	Pullovers	8	Wellington	Hats	4
Christchurch	Jeans	5	Wellington	Pullovers	7
Christchurch	Shoes	6	Wellington	Trousers	7

A Rank Query

- Retrieve the **third** best selling product type, use **`DENSE_RANK`**

City	The third best selling product type
Auckland	Jeans 10
Christchurch	Jeans 5
Wellington	Jackets 7, Pullovers 7, Jeans 7, Trousers 7

Overall			
ProdType	Amnt	ProdType	Amnt
Hats	28	Jeans	22
Shoes	25	Pullovers	15
Pajamas	25	Trousers	7
Socks	25		

Queries of the Type "Top N"

- Data analysts often require just top *N* best ranked elements of a dimension with regard to the measure
- Example:
  - Show the 10 best sold products for a given location *l* and time unit *t*

A Top N Query

- Retrieve the three best selling product types

City	The three best selling product types
Auckland	Pajamas 12, Socks 12, Hats 12
Christchurch	Hats 12, Shoes 6, Jens 5
Wellington	Pajamas 13, Socks 13, Shoes 8

Overall			
ProdType	Amnt	ProdType	Amnt
Hats	28	Jeans	22
Shoes	25	Pullovers	15
Pajamas	25	Trousers	7
Socks	25		

Computing All for Holistic Agg Funcs

- To compute overall aggregate we need all source values
  - In the case of the median, we needed to look at all source values to find out the overall median
  - In the case of the rank and topN functions, we needed all source values to compute overall numbers for each product type by adding sales numbers of the same product in different cities
  - Even if we wanted to find the third best selling product in any of cities, we would need all source values, since it is
    - Shoes (in Auckland)
- So, these are really holistic functions

A Problem with Rank() and TopN()

- Example:
  - Show the 10 best sold products for a given location *l* and time unit *t*
- The SQL statement

```
SELECT p.ProdId, p.ProdName, s.Sale
FROM Product p NATURAL JOIN Sales s
WHERE s.LocId = l AND s.TimeId = t
ORDER BY s.Sale DESC
```

will return all existing products (possibly thousands of them) and consume a lot of time to compute, although only ten first were needed

A Problem with Rank() and TopN()

- Some DBMSs allow defining the clause

  `OPTIMIZE FOR N ROWS`

  in the line below the clause **`ORDER BY`** of the SQL statement
- PostgreSQL supports the clauses
  - `LIMIT {COUNT | ALL}` and
  - `OFFSET` *`start`* (convenient for rank)
- This way, SQL processor is informed when to start and stop displaying the result
- But result is already computed
- How to inform SQL processor to perform computation just for first N items?
  - Think about!

Summary

- Distributive aggregates can be easily used to compute aggregates of coarser granularity
  - Distributive: `SUM(), COUNT(), MIN(), MAX()`
- Algebraic aggregates can be used to compute aggregates of the coarser granularity if the corresponding p-tuple is also produced
  - Algebraic: `AVG(), STDEV(), VAR(),`
- Holistic aggregates can be hardly used to compute aggregates of coarser granularity
  - Holistic: `MEDIAN(), MODE(), RANK(), TopN()`