



Chapter 5: Advanced SQL

Database System Concepts, 7th Ed.

©Silberschatz, Korth and Sudarshan

See www.db-book.com for conditions on re-use



Advanced Aggregation Features



Ranking

- Ranking is done in conjunction with an order by specification.
- Suppose we are given a relation
student_grades(ID, GPA)
giving the grade-point average of each student
- Find the rank of each student.
- **select ID, rank() over (order by GPA desc) as s_rank
from student_grades**
- An extra **order by** clause is needed to get them in sorted order
**select ID, rank() over (order by GPA desc) as s_rank
from student_grades
order by s_rank**
- Ranking may leave gaps: e.g. if 2 students have the same top GPA, both have rank 1, and the next rank is 3
 - **dense_rank** does not leave gaps, so next dense rank would be 2



Ranking

- Ranking can be done using basic SQL aggregation, but resultant query is very inefficient

```
select ID, (1 + (select count(*)  
                    from student_grades B  
                    where B.GPA > A.GPA)) as s_rank  
from student_grades A  
order by s_rank;
```



Ranking (Cont.)

- Ranking can be done within partition of the data.
- “Find the rank of students within each department.”

```
select ID, dept_name,  
       rank () over (partition by dept_name order by GPA desc)  
       as dept_rank  
from dept_grades  
order by dept_name, dept_rank;
```

- Multiple **rank** clauses can occur in a single **select** clause.
- Ranking is done *after* applying **group by** clause/aggregation
- Can be used to find top-n results
 - More general than the **limit** *n* clause supported by many databases, since it allows top-n within each partition



Ranking (Cont.)

- Other ranking functions:
 - **percent_rank** (within partition, if partitioning is done)
 - **cume_dist** (cumulative distribution)
 - fraction of tuples with preceding values
 - **row_number** (non-deterministic in presence of duplicates)
- SQL:1999 permits the user to specify **nulls first** or **nulls last**
select *ID*,
 rank () over (order by GPA desc nulls last) as s_rank
from *student_grades*



Ranking (Cont.)

- For a given constant n , the ranking the function $ntile(n)$ takes the tuples in each partition in the specified order, and divides them into n buckets with equal numbers of tuples.
- E.g.,
`select ID, ntile(4) over (order by GPA desc) as quartile`
`from student_grades;`



Windowing

- Used to smooth out random variations.
- E.g., **moving average**: “Given sales values for each date, calculate for each date the average of the sales on that day, the previous day, and the next day”
- **Window specification** in SQL:
 - Given relation *sales(date, value)*
select *date*, **sum**(*value*) **over**
 (**order by** *date* **between rows** 1 **preceding** and 1 **following**)
from *sales*



Windowing

- Examples of other window specifications:
 - **between rows unbounded preceding and current**
 - **rows unbounded preceding**
 - **range between 10 preceding and current row**
 - All rows with values between current row value -10 to current value
 - **range interval 10 day preceding**
 - Not including current row



Windowing (Cont.)

- Can do windowing within partitions
- E.g., Given a relation *transaction* (*account_number*, *date_time*, *value*), where *value* is positive for a deposit and negative for a withdrawal
 - “Find total balance of each account after each transaction on the account”

```
select account_number, date_time,  
       sum (value) over  
         (partition by account_number  
          order by date_time  
          rows unbounded preceding)  
       as balance  
from transaction  
order by account_number, date_time
```



OLAP



Data Analysis and OLAP

- **Online Analytical Processing (OLAP)**
 - Interactive analysis of data, allowing data to be summarized and viewed in different ways in an online fashion (with negligible delay)
- Data that can be modeled as dimension attributes and measure attributes are called **multidimensional data**.
 - **Measure attributes**
 - measure some value
 - can be aggregated upon
 - e.g., the attribute *number* of the *sales* relation
 - **Dimension attributes**
 - define the dimensions on which measure attributes (or aggregates thereof) are viewed
 - e.g., attributes *item_name*, *color*, and *size* of the *sales* relation



Example sales relation

<i>item_name</i>	<i>color</i>	<i>clothes_size</i>	<i>quantity</i>
skirt	dark	small	2
skirt	dark	medium	5
skirt	dark	large	1
skirt	pastel	small	11
skirt	pastel	medium	9
skirt	pastel	large	15
skirt	white	small	2
skirt	white	medium	5
skirt	white	large	3
dress	dark	small	2
dress	dark	medium	6
dress	dark	large	12
dress	pastel	small	4
dress	pastel	medium	3
dress	pastel	large	3
dress	white	small	2
dress	white	medium	3
dress	white	large	0
shirt	dark	small	2
shirt	dark	medium	4

...

... ...



Cross Tabulation of sales by *item_name* and *color*

clothes_size **all**

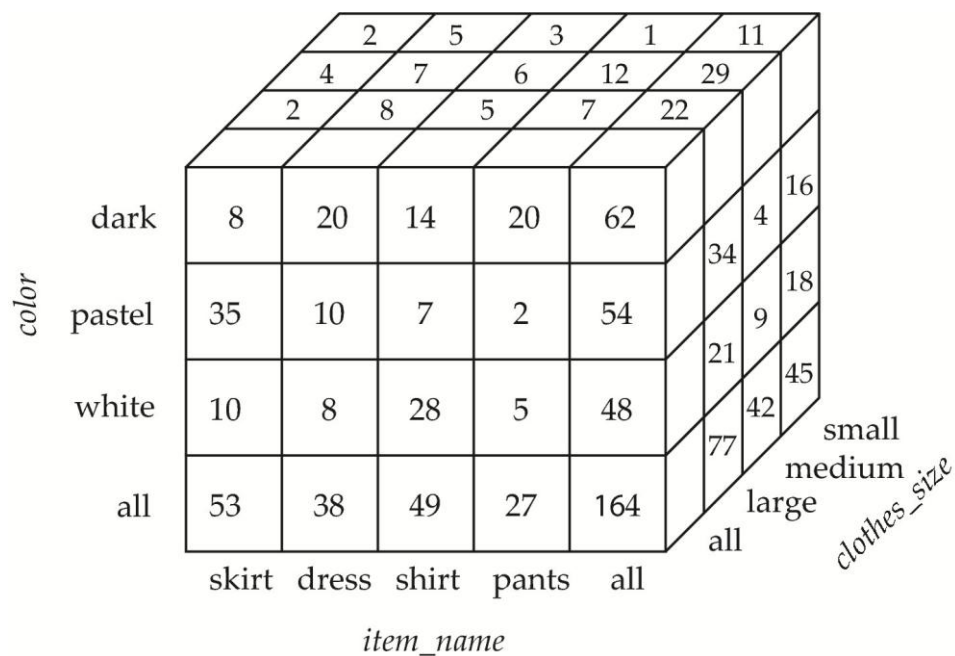
		<i>color</i>			
		dark	pastel	white	total
<i>item_name</i>	skirt	8	35	10	53
	dress	20	10	5	35
	shirt	14	7	28	49
	pants	20	2	5	27
total		62	54	48	164

- The table above is an example of a **cross-tabulation** (**cross-tab**), also referred to as a **pivot-table**.
 - Values for one of the dimension attributes form the row headers
 - Values for another dimension attribute form the column headers
 - Other dimension attributes are listed on top
 - Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.



Data Cube

- A **data cube** is a multidimensional generalization of a cross-tab
- Can have n dimensions; we show 3 below
- Cross-tabs can be used as views on a data cube





Cross Tabulation With Hierarchy

- Cross-tabs can be easily extended to deal with hierarchies
 - Can drill down or roll up on a hierarchy

clothes_size: **all**

		<i>color</i>				
<i>category</i>	<i>item_name</i>	dark	pastel	white	total	
womenswear	skirt	8	35	10	53	88
	dress	20	10	5	35	
	subtotal	28	45	15		
menswear	pants	14	7	28	49	76
	shirt	20	2	5	27	
	subtotal	34	9	33		
total		62	54	48		164



Relational Representation of Cross-tabs

- Cross-tabs can be represented as relations
 - We use the value **all** is used to represent aggregates.
 - The SQL standard actually uses null values in place of **all** despite confusion with regular null values.

<i>item_name</i>	<i>color</i>	<i>clothes_size</i>	<i>quantity</i>
skirt	dark	all	8
skirt	pastel	all	35
skirt	white	all	10
skirt	all	all	53
dress	dark	all	20
dress	pastel	all	10
dress	white	all	5
dress	all	all	35
shirt	dark	all	14
shirt	pastel	all	7
shirt	White	all	28
shirt	all	all	49
pant	dark	all	20
pant	pastel	all	2
pant	white	all	5
pant	all	all	27
all	dark	all	62
all	pastel	all	54
all	white	all	48
all	all	all	164



Extended Aggregation to Support OLAP

- The **cube** operation computes union of **group by**'s on every subset of the specified attributes
- Example relation for this section
sales(item_name, color, clothes_size, quantity)
- E.g., consider the query

```
select item_name, color, size, sum(number)
from sales
group by cube(item_name, color, size)
```

This computes the union of eight different groupings of the *sales* relation:

```
{ (item_name, color, size), (item_name, color),
  (item_name, size),        (color, size),
  (item_name),              (color),
  (size),                   ( ) }
```

where () denotes an empty **group by** list.

- For each grouping, the result contains the null value for attributes not present in the grouping.



Online Analytical Processing Operations

- Relational representation of cross-tab that we saw earlier, but with *null* in place of **all**, can be computed by
- ```
select item_name, color, sum(number)
from sales
group by cube(item_name, color)
```
- The function **grouping()** can be applied on an attribute
  - Returns 1 if the value is a null value representing all, and returns 0 in all other cases.

```
select item_name, color, size, sum(number),
 grouping(item_name) as item_name_flag,
 grouping(color) as color_flag,
 grouping(size) as size_flag,
from sales
group by cube(item_name, color, size)
```



# Online Analytical Processing Operations

- Can use the function **decode()** in the **select** clause to replace such nulls by a value such as **all**
  - E.g., replace *item\_name* in first query by  
**decode( grouping(*item\_name*), 1, 'all', *item\_name*)**



# Extended Aggregation (Cont.)

- The **rollup** construct generates union on every prefix of specified list of attributes

- E.g.,

```
select item_name, color, size, sum(number)
from sales
group by rollup(item_name, color, size)
```

- Generates union of four groupings:

```
{ (item_name, color, size), (item_name, color), (item_name), () }
```

- Rollup can be used to generate aggregates at multiple levels of a hierarchy.
- E.g., suppose table *itemcategory*(*item\_name*, *category*) gives the category of each item. Then

```
select category, item_name, sum(number)
from sales, itemcategory
where sales.item_name = itemcategory.item_name
group by rollup(category, item_name)
```

would give a hierarchical summary by *item\_name* and by *category*.



# Extended Aggregation (Cont.)

- Multiple rollups and cubes can be used in a single group by clause
  - Each generates set of group by lists, cross product of sets gives overall set of group by lists

- E.g.,

```
select item_name, color, size, sum(number)
from sales
group by rollup(item_name), rollup(color, size)
```

generates the groupings

$$\{item\_name, ()\} \times \{(color, size), (color), ()\}$$
$$= \{ (item\_name, color, size), (item\_name, color), (item\_name), (color, size), (color), () \}$$



# Online Analytical Processing Operations

- **Pivoting:** changing the dimensions used in a cross-tab is called
- **Slicing:** creating a cross-tab for fixed values only
  - Sometimes called **dicing**, particularly when values for multiple dimensions are fixed.
- **Rollup:** moving from finer-granularity data to a coarser granularity
- **Drill down:** The opposite operation - that of moving from coarser-granularity data to finer-granularity data



# OLAP Implementation

- The earliest OLAP systems used multidimensional arrays in memory to store data cubes, and are referred to as **multidimensional OLAP (MOLAP)** systems.
- OLAP implementations using only relational database features are called **relational OLAP (ROLAP)** systems
- Hybrid systems, which store some summaries in memory and store the base data and other summaries in a relational database, are called **hybrid OLAP (HOLAP)** systems.





# OLAP Implementation (Cont.)

- Early OLAP systems precomputed *all* possible aggregates in order to provide online response
  - Space and time requirements for doing so can be very high
    - $2^n$  combinations of **group by**
  - It suffices to precompute some aggregates, and compute others on demand from one of the precomputed aggregates
    - Can compute aggregate on  $(item\_name, color)$  from an aggregate on  $(item\_name, color, size)$ 
      - For all but a few “non-decomposable” aggregates such as *median*
      - is cheaper than computing it from scratch
- Several optimizations available for computing multiple aggregates
  - Can compute aggregate on  $(item\_name, color)$  from an aggregate on  $(item\_name, color, size)$
  - Can compute aggregates on  $(item\_name, color, size)$ ,  $(item\_name, color)$  and  $(item\_name)$  using a single sorting of the base data



# End of Chapter 5