Database Lab
Queries

Fall Term 2023

Dr. Andreas Geppert
geppert@acm.org

Topics

- conceptual design
- logical design
- consistency constraints
- data manipulation
- **queries**
- transactions
- views
- stored procedures and user-defined functions
- triggers
- security
- (database applications with Java (JDBC))

Queries

- joins
- left ¦ right ¦ full outer joins
- subqueries
- (not) exists subqueries
- aggregates
- case (conditional)
- Expressions
  - Access to attributes of structured types
  - Strings
  - Dates (comparison, arithmetics, current date and time)

Queries 2

- group by and aggregates
- finding the tuples with extreme values
- subquery in from-clause
- temporary views ("with")
- also available in the meantime: super groups (grouping sets, cube, rollup)
- OLAP (analytic) functions/local grouping
- recursive queries

Temporary Views

- view is defined only for the execution of a query
- Example: the location with the biggest map (in terms of number of bytes)

```
with maps(shortname, len) as
    (select shortname, length(map) from location)
select *
 from maps
 where len = (select max(len) from maps);
```

Analytic Queries

- conventional aggregates in SQL are avg, sum, count, min, max
- but there is a need for further, new aggregation operators
  - for instance ranking: assign a “rank”to each tuple whhich corresponds to its position in an ordered list
  - example: bestselling books, charts
- conventional grouping partitions all the tuples in a (intermediate) relation and performs aggregate on entire partitions
- in advanced analysis, more flexible,”local” grouping is required
  - example: moving average over a three/months period

Analytic Queries (2)

- analytic operators: over-clause
- local partitioning
  - forming of groups (partition by)
  - sorting (order by)
  - window definition (rows oder range)
- select year, month, sum(sales) over(partition by year) as cumsales
  from sales;
- select year, month,
    sum(sales) over(partition by year order by month) as cumsales
  from sales;

Analytic Queries (3)

- select year, month, sales,
  avg(sales) over(partition by year
  order by month
  rows between 1 preceding and 1 following) as mvgavg
  from sales;
- new operators:
  - Rank, denserank, row_number
  - Ntile
  - Lag, lead, nth, first_value, last_value
  - Cume_dist
  - min, max, avg, sum, count are possible as well

Grouping in SQL

- traditionally:
  - group-by clause
  - Per query, there is a fixed set of grouping criteria
- Suboptimal for flexible grouping
  - Along multiple dimension
  - On multiple levels of a dimension hierarchy
- all combinations of grouping attributes
  - 2^n queries with corresponding grouping criteria
  - (product, store, date)
    ⇒ [(product, store, date), (store, date),
    (product, date), (product, store)
    (product), (store), (date), ()]

➢Super groups: grouping sets, rollup, cube

brand	size	sales
Foo	L	10
Foo	M	20
Bar	M	15
Bar	L	5
Bar		3
	L	2

Grouping Sets

- Groups along multiple grouping criteria
- In a single query!
  - Grouping sets
  - Explicit listing of all grouping criteria
- Example

```
SELECT brand, size, sum(sales)
  FROM items_sold
GROUP BY GROUPING SETS ((brand), (size), ());
```

The Grouping Function: Example

- Similar query as above

```
SELECT (case when grouping(brand) = 1 then 'ALL'
                                  else brand end) as brand,
       (case when grouping(size) = 1 then 'ALL'
                                  else size end) as size,
       sum(sales)
  FROM items_sold
 GROUP BY GROUPING SETS
       ((brand), (size), ());
```

brand	size	sum
Bar	ALL	23
Foo	ALL	30
	ALL	2
ALL	ALL	55
ALL	L	17
ALL	M	35
ALL		3

The Cube Operator

- Grouping with all possible combinations?
- G1...Gn ⇒ 2^n criteria with grouping sets

➢abbreviation: the **cube** operator

- cube(G1...Gn) ⇒ grouping sets($2^{\{G1...Gn\}}$)
- Example: cube(A, B) ⇒ grouping sets((A,B), (A), (B), ())
- ( ): grand total

The Cube Operator: Example

- Sales grouped by:
  - brand
  - size
  - brand and size
  - And overall sum (grand total)

```
SELECT brand, size, sum(sales)
  FROM items_sold
 GROUP BY  cube (brand, size )
```

```
 brand | size | sum
-------+------+-----
 Bar   | L    |   5
 Bar   | M    |  15
 Bar   |      |   3
 Bar   |      |  23
 Foo   | L    |  10
 Foo   | M    |  20
 Foo   |      |  30
       | L    |   2
       |      |   2
       |      |  55
       | L    |  17
       | M    |  35
       |      |   3
```

The Rollup Operator

- Often we are not interested in all possible grouping criteria
- But mainly in all aggregates along a (subset of a) dimension hierarchy
- This is, we would like to see a setpwise rollup

➢ **Rollup** operator

- Computes *n* grouping combinations + grand total
- rollup(Family, Department, Product) ⧈
  grouping sets((Family, Department, Product),
  (Family , Department), (Family), ()

The Rollup Operator: Example

- Sum of sales per
  - Brand and size
  - Brand
  - overall (grand total)

```
SELECT brand, size, sum(sales)
  FROM items_sold
 GROUP BY  rollup (brand, size )
```

```
 brand | size | sum
-------+------+-----
 Bar   | L    |   5
 Bar   | M    |  15
 Bar   |      |   3
 Bar   |      |  23
 Foo   | L    |  10
 Foo   | M    |  20
 Foo   |      |  30
       | L    |   2
       |      |   2
       |      |  55
```

Recursive Queries

- the result is computed iteratively
- example: which airports can you reach with at most 3 stops from Zurich?

```
with recursive transverbindung(von, nach, stops, weg) as
  ((select von, nach, 0, von || '-' || nach
     from verbindung
    where von = 'ZRH')
   union all
   (select v.von, v.nach, stops + 1, weg || '-' || v.nach
     from verbindung v join transverbindung t on t.nach = v.von
    where stops <= 2))
select * from transverbindung;
```