Extrema Functions in SQL

By

Joe Celko

copyright 2003

MIN() and MAX()

- Standard SQL aggregate functions
- Work on all SQL datatypes
  - All SQL datatypes are ordered
  - This includes temporal data

MIN() and MAX() - 2

- **First: compute the expression in the parameter to build a working set**
  - **expression can be a formula, column or function call, but not a subquery expression**
- **Second: drop all the NULLs from that set**
- **Third: return the appropriate value**
- **Fourth: if the working set is empty or all NULLs, then return NULL**

MIN() and MAX() - 3

- MAX()
  - For numeric, greatest numeric value
  - For temporal, latest date or time, even if it is in the future
  - For character, highest value as defined by collation sequence

MIN() and MAX() - 4

- MIN()
  - For numeric, least numeric value
  - For temporal, earliest date or time, even if it is in the future
  - For character, lowest value as defined by collation sequence

MIN() and MAX() - 5

- MIN(x) = MAX(x) implied one element in the working set of x's
- MAX(DISTINCT <expression>) and MIN(DISTINCT <expression>) exist but are redundant -- think about it
- Aggregate functions cannot be nested
  - They are set properties, so there is only one of them MAX(MAX(x)) is at best redundant

Computing Extremas

- **Brute force table scan -- The worst way to do it**
- **Use an index tree -- search down the tree**
- **Use a hash index -- depends on the hashing algorithm used; Parallelism helps**
  - **Get the lowest (highest) value from each hash bucket**
  - **Pick the lowest (highest) from that result set**
- **Go to the Statistics table and read the extrema values directly**
  - **DB2 and large databases use them to determine the statistical distribution for cost algorithms.**

GREATEST() and LEAST()

- Oracle extensions, not standard
- GREATEST(a,b,c, ..) returns the highest value in the list
- LEAST(a,b,c, ..) returns the lowest value in the list

GREATEST() and LEAST()

- Can be done in Full SQL-92, but it is ugly
- Use subquery expression and table constructor
- (SELECT MAX(x)
  FROM VALUES (a), (b), (c),.. AS X(x))
- (SELECT MIN(x)
  FROM VALUES (a), (b), (c),.. AS X(x))

Top(n) Values

- One version exists in Microsoft ACCESS
- Assumes an ordering on a set of values
- Best done procedurally with Partition routine from QuickSort

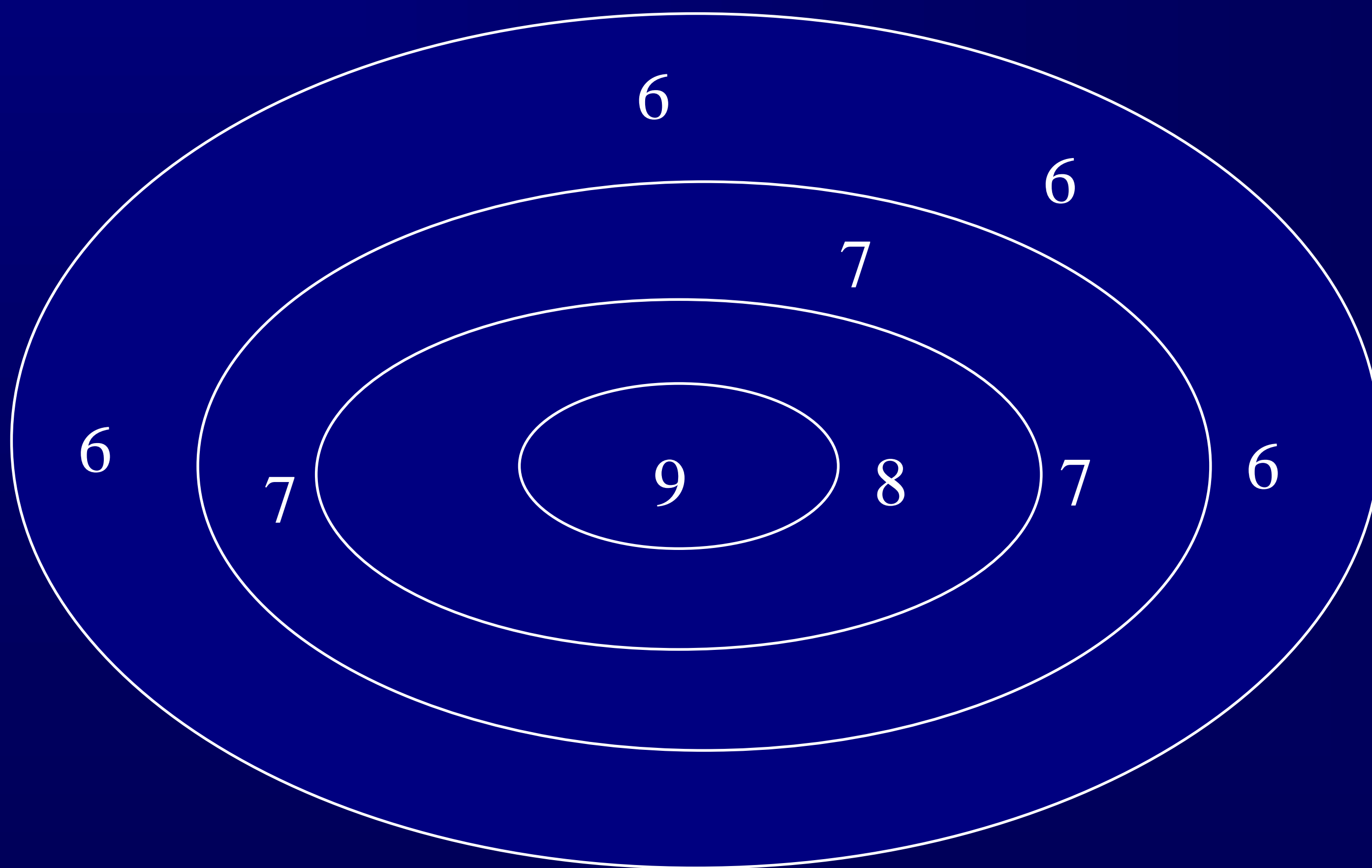
Top (n) Values -2

- Procedural approach:
  - Sort the file in descending order
  - return the top (n) of them with a loop
- Problems: the spec is bad
  - How do you handle ties?
  - How do you handle less than (n) values?

Top (n) Values - 3

- **Subset approach:**
  - **Decide if ties count or not; this is the pure set model versus SQL's multi-set model**
  - **Find the subset with (n) or fewer distinct values that are greater than some value (x).**
  - **Remember "distinct values" is not the same thing as "distinct elements"**
  - **Use self-join - one for subset elements, one for subset boundary**

Nested Sets



Nested Sets - 2

- SELECT DISTINCT salary
  FROM Employees AS E1 -- elements
  WHERE :n >= (SELECT COUNT(*) -- n is parameter
  FROM Employees AS E2 -- boundary
  WHERE E1.salary >= E2.salary);
- Use > or >= , depending on where you put the boundary in relation to the elements.
- Use SELECT or SELECT DISTINCT, depending on how you want to count elements

Nested Sets - 3

- An equivalent version can also be done with a GROUP BY clause, but you need to use a COUNT(DISTINCT) operator to handle duplicates.
- SELECT E1.salary

```
SELECT E1.salary
   FROM Personnel AS E1, Personnel AS E2
 WHERE E1.salary <= E2.salary
  GROUP BY E1.salary  -- boundary value
HAVING COUNT(DISTINCT E2.salary) <= :n;
```

Rankings - 1

- Rankings are how a value is positioned in a set.
- SQL has to do this with nested sets instead of sequential ordering.
- This is useful in scalar subqueries where it substitutes for row numbers

Rankings -2

- **Rankings allow for ties. This means that ranking numbers are always consecutive**
- **SELECT company, amount,**

```
(SELECT COUNT(DISTINCT amount)
    FROM Clients AS C1
  WHERE C1.amount >= C0.amount) AS rank
FROM Clients AS C0;
```

- **Used in Statistics**

Standings -1

- Standings do not allow for ties. This means that numbers can gap
- SELECT company, amount,

```
(SELECT COUNT(amount)
    FROM Clients AS C1
  WHERE C1.amount >= C0.amount) AS standing
FROM Clients AS C0;
```

- Used in some school systems

Complement of Extrema

- **If you need to list them**
  - **x [NOT] IN (<extema subquery>)**
  - **[NOT] EXISTS (<extema subquery>)**
- **If you need to summarize them into an “everyone else”, “the field” etc. group**
  - **UNION query with SELECT ...GROUP BY**

ALL predicate versus Extremas

- **It is counter-intuitive at first that these two predicates are not the same in SQL**
  - **x >= (SELECT MAX(y) FROM Table1)**
  - **x >= ALL (SELECT y FROM Table1)**
- **Extrema functions drop NULLs before returning the greatest or least values**
- **The ALL predicate does not drop NULLs, so you can get them in the results.**

Group Characteristics via Extremas

- You can use the aggregate functions and the HAVING clause to determine certain characteristics of the groups formed by the GROUP BY clause.

```
SELECT col1, col2
  FROM Foobar
GROUP BY col1, col2
HAVING ..
```

- You can determine the following properties of the groups with these HAVING clauses:

Group Characteristics- 2

- HAVING COUNT (DISTINCT col_x) = COUNT (col_x) col_x has all distinct values
- HAVING COUNT(*) = COUNT(col_x); -- There are no NULL in the column
- HAVING MIN(col_x - <const>) = -MAX(col_x - <const>) -- col_x deviates above and below const by the same amount
- HAVING MIN(col_x) * MAX(col_x) < 0 -- MAX is positive and MIN is negative
- HAVING MIN(col_x) * MAX(col_x) = 0 -- either one or both of MIN or MAX is zero

Group Characteristics- 3

- **HAVING MIN(col_x) * MAX(col_x) > 0 -- col_x is either all positive or all negative**
- **HAVING MIN(col_x) = -MAX(col_x) -- col_x deviates above and below zero by the same amount**
- **HAVING MIN(SIGN(col_x)) = MAX(SIGN(col_x)) -- col_x is all positive, all negative or all zero**
- **HAVING MIN(ABS(col_x)) = 0; -- col_x has at least one zero**
- **HAVING MIN(ABS(col_x)) = MIN(col_x) -- col_x >= 0 (although the where clause can handle this, too)**

Group Characteristics- 4

- **HAVING MIN(col_x) = -MAX(col_x) -- col_x deviates above and below zero by the same amount**
- **HAVING MIN(col_x) * MAX(col_x) = 0 -- either one or both of MIN or MAX is zero**
- **HAVING MIN(col_x) < MAX(col_x) -- col_x has more than one value (may be faster than count (*) > 1)**
- **HAVING MIN(col_x) = MAX(col_x) -- col_x has one value or NULLs**
- **HAVING MIN(SIGN(ABS(col_x)) = 0 -- col_x has at least one zero**

Group Characteristics- 5

- **The HAVING clause does not need to be used with a GROUP BY clause**
- **By itself, HAVING treats the entire table as if it were one group based on a “phantom column”**
- **For example, to find any gaps in a table with a sequentially numbered column :**

```
(SELECT 'x'
  FROM Foobar
 HAVING COUNT(*) = MAX(seq_nbr)) = 'x'
```

- **Notice the use of a constant in the SELECT clause - column name or * will not work**

Questions & Answers

?