



Picking Up Where the SQL Optimizer Leaves Off

Howard Schreier
U.S. Dept. of Commerce

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. © indicates USA registration. Other brand and product names are registered trademarks or Trademarks of their respective companies.

1

Purpose

Structured Query Language (SQL) is an extremely versatile and useful component of Base SAS® software. With SQL, one can sometimes express the solution to a problem in one code statement. Unfortunately, in some cases that code is prohibitively inefficient. This paper is a case study showing ways in which the inefficient code can be modified and tuned to the extent that a solution can be computed.

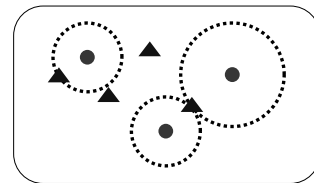
2

Case study, matching spatial data

- First table: locations of fish in a body of water
- Second table: points where water conditions measured
- Task: find closest water-condition point for each fish-location point

3

Diagram (two dimensions)



- Fish data point
- ▲ Water data point

4

SQL code (three dimensions)

```
create table nearest as
select FishID, WaterID,
       sqrt((fish.x-water.x)**2
            +(fish.y-water.y)**2
            +(fish.z-water.z)**2
            ) as distance
from fish, water
group by FishID
having distance=min(distance);
```

5

Scale is the issue

- 650 thousand fish points
- 30 million water points
- Almost 20 **trillion** rows in intermediate result
- Requirements
 - 50 years
 - disk footprint for WORK library: 3 petabytes

6

Preliminary bit of optimization

```
create table nearest as
select FishID, WaterID,
       (fish.x-water.x)**2
       +(fish.y-water.y)**2
       +(fish.z-water.z)**2
       as distance_squared
from fish, water
group by FishID
having distance_squared
       =min(distance_squared);
```

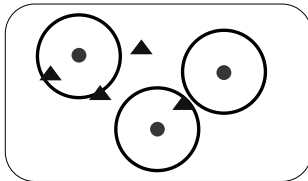
7

Parallel partitioning

- Divide fish table into pieces; solve problem for each separately; combine results
- Makes task no smaller, but more manageable

8

Variant: Upper bound on distance



- Fish data point
- ▲ Water data point

9

Variant: Upper bound on distance

```
%let radius=10;
create table nearest as
select FishID, WaterID,
       (fish.x-water.x)**2+
       (fish.y-water.y)**2+
       (fish.z-water.z)**2
       as distance_squared
from fish, water
```

10

Variant: Upper bound on distance

```
where calculated
distance_squared<&RADIUS**2
group by FishID
having distance_squared=
min(distance_squared);
```

11

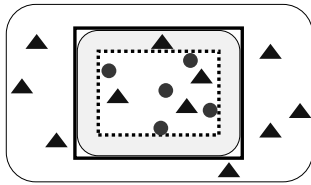
Consequences

- Size of join unchanged
- Smaller intermediate result passed to “back end”
- Wrong answer (but subset of correct answer)

Useful nevertheless

12

Anticipatory subsetting



- Fish data point
- ▲ Water data point

13

Anticipatory subsetting

```
%let radius=10;  
select max(x), max(y), max(z),  
       min(x), min(y), min(z)  
into :maxx, :maxy, :maxz,  
     :minx, :miny, :minz  
from fish;
```

14

Anticipatory subsetting

```
create table watersubset as  
select * from water  
where &MINX-&RADIUS <= x <=  
      &MAXX+&RADIUS  
and &MINY-&RADIUS <= y <=  
     &MAXY+&RADIUS  
and &MINZ-&RADIUS <= z <=  
     &MAXZ+&RADIUS;
```

15

Anticipatory subsetting

```
create table nearest as  
select FishID, WaterID,  
       (fish.x-water.x)**2+  
       (fish.y-water.y)**2+  
       (fish.z-water.z)**2  
       as distance_squared  
from fish, watersubset as water
```

16

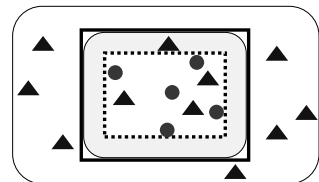
Anticipatory subsetting

```
where calculated  
distance_squared<&RADIUS**2  
group by FishID  
having distance_squared=  
min(distance_squared);
```

17

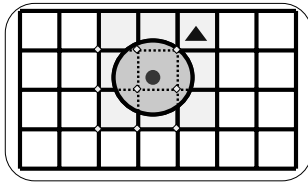
Consequences

- No change in results, since restriction is redundant
- Reduction in size of join (maybe)



18

Redundant equi-join condition



- Fish data point
- ▲ Water data point

19

Redundant equi-join condition

- Coordinates of LL corner of small square with fish point:
 $R * \text{int}(X/R), \quad R * \text{int}(Y/R)$
- Coordinates of LL corners of 9 small squares:
 $R * \text{int}(X/R) \pm R, \quad R * \text{int}(Y/R) \pm R$
- So
 $R * \text{int}(X_w/R) = R * \text{int}(X_f/R) \pm R$
 $R * \text{int}(Y_w/R) = R * \text{int}(Y_f/R) \pm R$
- Dividing through by R:
 $\text{int}(X_w/R) = \text{int}(X_f/R) \pm 1$
 $\text{int}(Y_w/R) = \text{int}(Y_f/R) \pm 1$

20

OFFSETS table implements \pm operator

xoffset	yoffset
-1	-1
-1	0
-1	1
0	-1
0	0
0	1
1	-1
1	0
1	1

21

Redundant equi-join condition

```
%let radius=10;
create table nearest as
select FishID,
       WaterID,
       (fish.x-water.x)**2
      +(fish.y-water.y)**2
      +(fish.z-water.z)**2 as
       distance_squared
from fish, offsets, water
```

22

Redundant equi-join condition

```
where calculated
distance_squared < &RADIUS**2
and      int( fish.x/&RADIUS)
+xoffset = int(water.x/&RADIUS)
and      int( fish.y/&RADIUS)
+yoffset = int(water.y/&RADIUS)
and      int( fish.z/&RADIUS)
+zoffset = int(water.z/&RADIUS)
group by FishID
having distance_squared=
min(distance_squared);
```

23

Consequences

- No change in results, since restriction is redundant (as the circle is within the large square)
- Increases theoretical size of join by factor of 27
- Makes it possible to avoid complete brute-force evaluation

24



The Genuine **BRANNOCK** Device

“Designed in 1927, The Brannock Device® foot-measuring device is a must in all retail footwear stores.”

<http://www.brannock.com/>

25

Building block code

- Based on variant with upper bound
- Incorporates
 - anticipatory subsetting
 - redundant equi-join condition

26

Building block code

```
%let radius=10;
select max(x), max(y), max(z),
       min(x), min(y), min(z)
into :maxx, :maxy, :maxz,
     :minx, :miny, :minz
from fish;
```

27

Building block code

```
create view watersubset as
select * from water
where &MINX- &RADIUS <= x <=
      &MAXX+ &RADIUS
      and &MINY- &RADIUS <= y <=
      &MAXY+ &RADIUS
      and &MINZ- &RADIUS <= z <=
      &MAXZ+ &RADIUS;
```

28

Building block code

```
create table nearest as
select FishID,
       WaterID,
       (fish.x-water.x)**2
      +(fish.y-water.y)**2
      +(fish.z-water.z)**2 as
       distance_squared
from fish,
     offsets,
     watersubset as water
```

29

Building block code

```
where calculated
       distance_squared < &RADIUS**2
       and       int( fish.x/&RADIUS)
       +xoffset = int(water.x/&RADIUS)
       and       int( fish.y/&RADIUS)
       +yoffset = int(water.y/&RADIUS)
       and       int( fish.z/&RADIUS)
       +zoffset = int(water.z/&RADIUS)
group by FishID
having distance_squared=
       min(distance_squared);
```

30

Performance

- Generated test data tables with 10,000 rows in each (fish, water)
- 3:50 using brute force
- 2:50 with anticipatory subsetting
- 0:33 with redundant equi-join condition (only)
- 0:25 with both techniques

31

Review

- Original problem has simple SQL solution
- That solution does not scale well
- Variant problem
 - Returns only a subset of the correct result
 - Does permit performance tuning

32

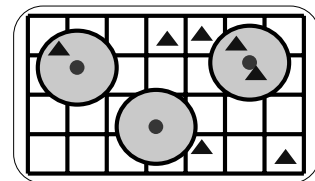
Concentric partitioning

1. Set value for upper bound (&RADIUS)
2. Solve using building block code; result is subset of the correct result
3. Remove fish points successfully matched
4. Increase upper bound (&RADIUS)
5. Repeat (2) through (4) as many times as necessary

The devil is in the details.

33

Concentric partitioning: choosing radii



- Fish data point
- ▲ Water data point

34

Concentric partitioning: choosing radii

- Ideal: one-to-one
- Too small: no match-ups
- Too large: inadequate performance gain over the naive code
- Medium

35

Strategy overview

- Apply parallel partitioning to divide problem into pieces of manageable size
- Use concentric partitioning to solve each piece
- Each concentric partitioning iteration uses the building block SQL code (which incorporates anticipatory subsetting and redundant equi-join condition)

36

Equi-join methods in PROC SQL

- Coordinated sequential processing
- Index-based processing
- Hashing
 - Preferred method for this problem
 - Limited by memory availability

37

Providing memory for hashing

- Partition data table(s) to reduce size of joins, and thus amount of memory needed
- Use (undocumented) option `BUFFERSIZE=` to increase available memory

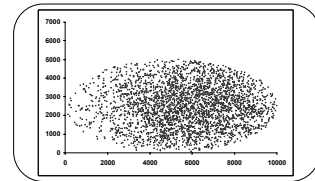
38

Data for testing

- Synthetic
- Full scale
 - 650 thousand fish points
 - 30 million water points
- Realistic (asymmetries, discontinuities, outliers)

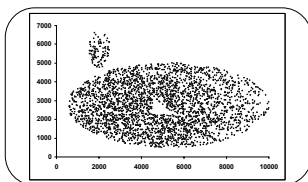
39

Sample fish points (X-Y projection)



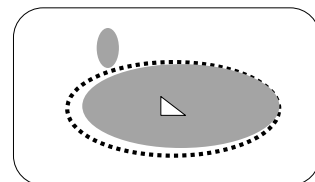
40

Sample water points (X-Y projection)



41

Overlay



42

SAS coding

- Based on building block
- Other code (mostly SQL) for generating data, sampling, partitioning, capturing performance metrics, other housekeeping
- Macro facility, especially for looping
- Validated against results of brute-force solution

43

Decisions

- Number and shape of parallel partitions
- Progression of radii

44

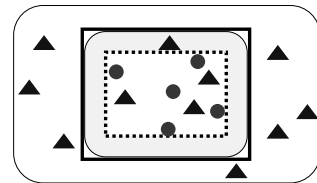
Objectives

- Small joins (to enable hashing)
- Small result sets from joins (to manage disk footprint)
- Avoid excessive overhead
- Limit aggregate number of pairings considered (<< 20 trillion)

45

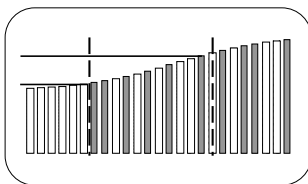
Parallel partitions

- 50
- Based on quantiles of X coordinate



46

Concentric partitions: radii



47

Results

- All 650,000 fish data points matched
- Building block code run 926 times
- Aggregate processing time was approximately 62 hours
- Times for the 50 parallel partitions (each with approximately 13,000 fish data points)
 - Minimum: 13 minutes
 - Maximum: 8 hours
- Aggregate size of joins: approximately 39 trillion
- Estimated 70 billion rows passed to “back end”

48

About the speaker

Howard Schreier

**U.S. Dept. of Commerce
Washington DC 20230**



(202) 482-4180

Howard_Schreier@ita.doc.gov