Sybase Adaptive Server Advanced SQL Programming and Optimization

Adaptive Server Join Processing
Chapter Seven

Soaring Eagle Consulting

Important Notes

- Sybase Adaptive Server is a trademark of Sybase Inc.
- This presentation is copyrighted.
- This presentation is not for re-sale
- This presentation shall not be used or modified without express written consent of Soaring Eagle Consulting, Inc.

Soaring Eagle Consulting

About Soaring Eagle

Since 1997, Soaring Eagle Consulting has been helping enterprise clients improve their Application Performance Management, (APM) at the database tier, arguably the most volatile and critical component of distributed application architecture. Our clients range in size from fledgling startups through Fortune 100 companies and leading financial institutions.

Soaring Eagle has been a leader in software development, architecture, performance and tuning databases, while promoting mentoring and training all over the world for over a decade. Many of our employees, and partners have written books, speak at seminars about leading edge technologies. We have expertise in all business tiers, financial; health, manufacturing, government agencies and many ecommerce businesses.

Consulting

- Performance & Tuning
- Application Performance Management
- Emergency Triage
- Performance audits
- Staff Augmentation
- Project management
- Database architecture
- Scalability assessment and planning

Training

- Onsite/Web based
  - Microsoft
  - Sybase
  - Oracle
  - APM
  - Six Sigma

Managed Services

- Remote Database Management
- Performance management
- Emergency db Service
- Proactive mitigation
- Problem notification
- Problem resolution

Software

- Application Performance Management
- Database performance accelerator
- Database performance management
- Application Development

rpm™ remote performance monitoring

OPNET Application and Network Performance

CALL 911 EMERGENCY db

EXPERT Assist

CONFIO SOFTWARE

OverSight db

Soaring Eagle Consulting

Adaptive Serve Join Processing

Topics

* Understand Join Optimization
* Consider Special Topics In Multi-Table Optimization
* Join Order
* Indexes and Joins
* Overriding the Optimizer
* Breaking Up Large Queries
* Self Joins
* Outer Joins

Soaring Eagle Consulting

Join Optimization

* The optimizer must evaluate how much work is required based on
  * Possible join orders
  * Index selection for joins
  * Reformatting the query (dynamically creating an index)
* When evaluating join order, the server already knows the best index for each SARG, any or clauses, and join clauses
* The server evaluates all table sequences and join indexes, estimates cost (in I/O), and chooses the fastest query plan

Optimizer Limitations

* The optimal plan involves picking the best indexes and join orders
* As the number of tables increases, the number of permutations that the optimizer must evaluate increases factorially
* Sometimes, it is necessary to fool the optimizer or to break the query into parts (see below)
* There are some permutations so unlikely to help that the optimizer will not bother to evaluate them

Nested Iteration

The server executes the join by constructing a set of nested loops; this is called "Nested Iterations"

```
select *
from authors a, titleauthor ta
where a.au_id = ta.au_id
```

Slide 1 0f 3

SOARING EAGLE CONSULTING

Nested Iteration

PSEUDOCODE:

```
while not eof(authors) *
        read next row(authors) *
        while not eof(titleauthor) *
            read next row(titleauthor)
            if authors.au_id = titleauthor.au_id
               return this row
        end loop (titleauthor)
end loop (authors)
* {also physical aspect}
```

Slide 2 0f 3

Nested Iteration

OR

```
while not eof(titleauthor)
        read next row(titleauthor)
        while not eof(authors)
                read next row(authors)
                if authors.au_id = titleauthor.au_id
                     return this row
        end loop (titleauthor)
end loop (authors)
```

Slide 3 0f 3

Cost of Nested Iteration

* The general approach to a nested iteration is to choose the processing order, then to walk the outer table, matching it with each corresponding row in associated inner tables
* The choice of outer query vs. inner directly affects performance

Cost of Nested Iteration (Continued)

	Table 1	Table 2
Rows	1000	100,000
Bytes/Row	100	300
Pages	53	16,1667

If Table 1 is outer

```
53 + (1,000 * 16,667) =  16,667,053 I/Os
```

If Table 2 is outer

```
16,667 + (100000* 53) = 5,316,667 I/Os
```

Soaring Eagle Consulting

Join Order

* The server considers (within limitations) all possible join orders when looking for an optimal path
* Usually, the best path will be one that is supported with an existing index

```
select *
from authors a, titleauthor ta, titles t
where a.au_id = ta.au_id
and ta.title_id = t.title_id
```

Cost of Nested Iteration: Notes

* Performance is directly related to the number of times the inner query will be done
* You will often find that *with no indexes* the larger table should be the outer table
* When evaluating join order, the server looks to see if the table will fit in cache; If so, all I/O, except the first pass, is assumed to be logical

Join Order Continued

Here are the join orders the server will consider

authors → titleauthor → titles

titles → titleauthor → authors

Optimizer Tips

- If you provide more information about paths that may be supported by indexes, the server may find a better path
  - Explicitly list all join options in the query
  - When in doubt, add more join clauses

```
select *
from a,b,c
where a.key = b.key
and b.key = c.key
```

Should be written

```
select *
from a,b,c
where a.key = b.key
and b.key = c.key
and a.key = c.key
```

Soaring Eagle Consulting

Optimizer Tips (Continued)

Also,

```
select *
from a,b
where a.key = b.key
and a.key = 'ABC'
```

Should be written

```
select *
from a,b
where a.key = b.key
and a.key = 'ABC'
and b.key = 'ABC'
```

Nested Iteration Costs

* The cost of nested iteration is the number of pages in the outer table plus the number of rows in the outer table times the number of pages in the inner table

Table Statistics

	Rows	Rows/Page	Pages
titles	15000	15	1000
titleauthor	25000	50	500

```
select *
from titles t, titleauthor ta
where t.title_id = ta.title_id
and royalty_per > 50
```

Sample Cost Calculation

Assumptions

```
titles is outer table
titleauthor is inner table
No indexes
```

Optimizer Assumptions

```
logical read costs 2 ms
physical read costs 18 ms
```

Calculation

```
Outer Loop: 1000 pps in titles
Inner Loop (15000 rows in titles * 500 pps in titleauthor)
```

Sample Cost Calculations Continued

Assumptions

```
Ref'l Integrity (all titleauthors have titles)
> 50% of rows in titleauthor have royalty_per > 50
```

	I/O	Appr. Time
Titles first		
No index	7,502,500	10 hours
CI on ta.title_id	46,000	230 seconds
NCI ta.title_id	23,500	101 seconds
Titleauthor First		
No index	12,500,500	17 hours
CI on ta.title_id	13,000	65 seconds
NCI title_id	25,500	125 seconds
CI t.title_id CI ta.royaltyper	12,750	64 seconds

Based on 2ms / 18 ms

Soaring Eagle Consulting

Sample Cost Calculation Continued

1000 physical reads from titles	18ms * 1000	18000ms
1000 logical reads from titles	2ms * 1000	2000ms
500 physical reads from ***titleauthor***	18ms * 500	9000ms
15000 * 500 logical reads	2ms * 7,500,000	15,000,000ms
TOTAL		15,029,000ms

TA-T

500 pp * 18ms	900
500 1p * 2ms	1000
1000 pp * 18ms	18000

25,000 * 1000 1p =
25,000,000 * 2 ms 50,000,000

Soaring Eagle Consulting

Reformatting

Worst Case Join Cost

```
pages in outer + (rows in outer * pages in inner)
```

* Occasionally, it may be cheaper to index the inner table and use the index than to repeatedly scan the table
  (This is called reformatting.)
* (This is the cost of a merge join)

Cost

Pages in inner + Pages in outer + Rows in outer + (Pages in inner * $\log_2$(Pages in inner))

Reformatting Example

Outer:	300 rows, 200 pages
Inner	100 Pages
No index **Cost to Reformat**	200 + 300 * 100 = 30200 100 + 100 *$\log_2(100)$ + 200+300= 1300

Soaring Eagle Consulting

Query Plan: Reformatting

(No indexes)

SHOWPLAN

```
select * from pt_sample, pt_tx
where pt_sample.id = pt_tx.id
```

```
STEP 1
    The type of query is SELECT.
     7 operator(s) under root

   |ROOT:EMIT Operator
   |
   |   |SEQUENCER Operator has 2 children.
   |   |
   |   |   |STORE Operator
   |   |   |  Worktable1 created, in allpages locking mode, for REFORMATTING.
   |   |   |  Creating clustered index.
   |   |   |
   |   |   |   |INSERT Operator
   |   |   |   |  The update mode is direct.
   |   |   |   |
   |   |   |   |   |SCAN Operator
   |   |   |   |   |  FROM TABLE
   |   |   |   |   |  pt_sample
   |   |   |   |   |  Table Scan.
   |   |   |   |   |  Forward Scan.
   |   |   |   |   |  Positioning at start of table.
   |   |   |   |   |  Using I/O Size 2 Kbytes for data pages.
   |   |   |   |   |  With LRU Buffer Replacement Strategy for data pages.
   |   |   |   |
   |   |   |   |  TO TABLE
   |   |   |   |  Worktable1.
   |   |
   |
```

Query Plan: Reformatting

(No indexes)

SHOWPLAN

```
select * from pt_sample, pt_tx
where pt_sample.id = pt_tx.id
```

```
|     |     |     |   Worktable1.
          |     |
          |     |     |NESTED LOOP JOIN Operator (Join Type: Inner Join)
          |     |     |
          |     |     |     |SCAN Operator
          |     |     |     |   FROM TABLE
          |     |     |     |   pt_tx
          |     |     |     |   Table Scan.
          |     |     |     |   Forward Scan.
          |     |     |     |   Positioning at start of table.
          |     |     |     |   Using I/O Size 2 Kbytes for data pages.
          |     |     |     |   With LRU Buffer Replacement Strategy for data pages.
          |     |     |
          |     |     |     |SCAN Operator
          |     |     |     |   FROM TABLE
          |     |     |     |   Worktable1.
          |     |     |     |   Using Clustered Index.
          |     |     |     |   Forward Scan.
          |     |     |     |   Positioning by key.
          |     |     |     |   Using I/O Size 2 Kbytes for data pages.
          |     |     |     |   With LRU Buffer Replacement Strategy for data pages.
```

Example: OR with Joins

```
select title
from titles t, titleauthor ta, salesdetail sd
where t.title_id = ta.title_id
or t.title_id = sd.title_id
or ta.title_id = sd.title_id
```

Results

* A list of all titles that have either an author specified or have been sold
* If either of the two tables used in the join is empty, Adaptive Server will return no rows (why?)
  * Because the server cannot properly optimize it; so it creates a Cartesian product of the three tables

Soaring Eagle Consulting

Overriding the Optimizer

- Question
  - When might you out guess the optimizer?

- Answer
  - When you know something it doesn't.
  - Indexes did not exist at optimization time
  - Statistics are out of date
  - Device performance is different from assumption
  - Information is in cache already

Soaring Eagle Consulting

Forcing Join Order

* As we have seen, order of joins is as important as index selection for joins. If the server optimizes incorrectly, it is possible to force a specific join order with the FORCEPLAN option

```
set forceplan on

select * from titles,  publishers
where titles.pub_id = publishers.pub_id

set forceplan off
```

Forcing Join Order Continued

* FORCEPLAN tells the server to join in the order presented in the from clause
* In this example, titles will be the outer table and publishers will be the inner table, regardless of the access method the optimizer would normally pick
* This may cause problems as the data distribution changes. The optimizer would have noticed as soon as statistics were updated
* Always turn FORCEPLAN off after running the query for which you've turned it on
* Prior to putting a FORCEPLAN into production, try
  * set table count *int_value*

Soaring Eagle Consulting

Other Things We Can Force

* select *select_list*
* [from *table_name*
  * [(index {*index_name* | *table_name*}
  * [parallel [*degree_of_parallelism*]]
  * [prefetch *size*] [lru | mru])]}
  * [holdlock | noholdlock] [shared] [REPEAT AS NEEDED]
  * [where *search_conditions*]
  * [group by [all] *aggregate_free_expression*...]
  * [having *search_conditions*]
  * [order by *column_name* | *select_list*_number | *expression*} [asc | desc]
  * [at isolation {read uncommitted | read committed | serializable}]

Lab 7.1 Forcing Join Order

Identify a query from a prior lab with which you are familiar. Force the join order in the opposite direction, compare I/O.

Device Performance Examined

* The optimizer assumes that the ratio of physical to logical read performance is 18 to 2. What is the effect of this assumption on a typical join?

Query Cost Estimate

* PC is Physical Cost of page retrieval
* LC is Logical Cost of page retrieval

Consider this example

TABLE	TABLE A	TABLE B
Size	10MB (5000 pgs)	2 MB (1000 pgs)
Rows	1M	250K
NCI levels/size	4 levels, 500 (pgs)	3 levels, 100 (pgs)

Device Performance Examined Continued

A->B join order, using nonclustered index (NCI) on B

* Assume that table B and NCI fit in cache

```
COST = (5,000 pgs * PC) + (1,100 pgs * PC)
            + 1,000,000 rows * 4 levels * LC
      = 6,100 * PC + 4,000,000 * LC
```

B->A join order, using nonclustered index on A

Query Cost Estimate

* Assume that NCI of Table A (but not table) fit in cache

```
COST   = (1000 pgs * PC) + (500 pgs * PC) + 250,000 rows
       * (4 index levels * LC + 4 data pages * PC)
       = 1,001,500 * PC + 1M * LC
```

Breaking Up Large Queries

This section examines how to break up large queries

* Defining Large Queries
* When to Break Up a Large Query
* How to Break Up Large Queries

Defining Large Queries

* A large multi-table query is a query consisting of many tables with sufficient rows in the tables to make a join expensive
* The critical issue in multi-table queries is join order
* The server may not be able to find an efficient join order with all of the tables in the query
* May require breaking the query into multiple parts, using one or more temporary tables

Note: This is a strategy that the optimizer will not try

Soaring Eagle Consulting

When to Break Up a Large Query

Here is a candidate query for breaking into components:

```
select sum(t1.amount)
from pt_tx_CIid t1, pt_tx_NCamount t2,
      pt_sample_CIidNCk s
where t1.id = t2.id
and s.id = t1.id
and t2.id = s.id
and s.key2 between 157573487 and 257573487
```

Soaring Eagle Consulting

A Sample of a Large Query

SHOWPLAN OUTPUT

```
 STEP 1
     The type of query is SELECT.

5 operator(s) under root

    |ROOT:EMIT Operator
    |
    |   |SCALAR AGGREGATE Operator
    |   |  Evaluate Ungrouped SUM OR AVERAGE AGGREGATE.
    |   |
    |   |   |N-ARY NESTED LOOP JOIN Operator has 3 children.
    |   |   |
    |   |   |   |SCAN Operator
    |   |   |   |  FROM TABLE
    |   |   |   |  pt_sample_CIidNCk
    |   |   |   |  s
    |   |   |   |  Index : NCK
    |   |   |   |  Forward Scan.
    |   |   |   |  Positioning by key.
    |   |   |   |  Keys are:
    |   |   |   |    key2 ASC
    |   |   |   |  Using I/O Size 2 Kbytes for index leaf pages.
    |   |   |   |  With LRU Buffer Replacement Strategy for index leaf pages.
    |   |   |   |  Using I/O Size 2 Kbytes for data pages.
    |   |   |   |  With LRU Buffer Replacement Strategy for data pages.
    |   |   |
```

A Sample of a Large Query

SHOWPLAN OUTPUT

```
 |   |   |
          |   |   |   |SCAN Operator
          |   |   |   |  FROM TABLE
          |   |   |   |  pt_tx_NCamount
          |   |   |   |  t2
          |   |   |   |  Table Scan.
          |   |   |   |  Forward Scan.
          |   |   |   |  Positioning at start of table.
          |   |   |   |  Using I/O Size 2 Kbytes for data pages.
          |   |   |   |  With LRU Buffer Replacement Strategy for data pages.
          |   |   |
          |   |   |   |SCAN Operator
          |   |   |   |  FROM TABLE
          |   |   |   |  pt_tx_CIid
          |   |   |   |  t1
          |   |   |   |  Using Clustered Index.
          |   |   |   |  Index : ci
          |   |   |   |  Forward Scan.
          |   |   |   |  Positioning by key.
          |   |   |   |  Keys are:
          |   |   |   |    id ASC
          |   |   |   |  Using I/O Size 2 Kbytes for data pages.
          |   |   |   |  With LRU Buffer Replacement Strategy for data pages.
```

A Sample of A Large Query Continued

```
  STEP 3
        The type of query is SELECT.
Server Message:  Number  1562, Severity  10
The sort for Worktable1 is done in Serial
```

- Results (keep it around to check your answer at the end!)

```
-----------------------------
390998878041299392.15
```

A Sample of A Large Query Continued

STATISTICS IO

```
Table: pt_tx_CIid  scan count 21,  logical reads:
(regular=3822 apf=0 total=3822),  physical reads:
(regular=0 apf=0 total=0),  apf IOs used=0
Table: pt_tx_NCamount  scan count 1,  logical
reads: (regular=182 apf=0 total=182),  physical
reads: (regular=0 apf=0 total=0),  apf IOs used=0
Table: pt_sample_CIidNCk  scan count 1,  logical
reads: (regular=23 apf=0 total=23),  physical
reads: (regular=0 apf=0 total=0),  apf IOs used=0
Table: Worktable1   scan count 34,  logical
reads: (regular=13414 apf=0 total=13414),
physical reads: (regular=0 apf=0 total=0),  apf
IOs used=0
Total writes for this command: 129
```

When to Break Up a Query

* To allow the server to make use of all candidate indexes
* To allow a table scan to be used on an outer table (not an inner table)

Soaring Eagle Consulting

How to Break Up Large Queries

* Separate the query into components that each have an effective strategy
* Try several approaches to see which gives the lowest total STATISTICS IO values
* Use one or many temporary tables to store and re-join values, especially when you have a highly selective index

Soaring Eagle Consulting

How to Break Up Large Queries Continued

- At the first step, reduce the number of rows in play

```
select s.id, t1.amount
into #temp
from pt_tx_NCamount t1, pt_sample_CIidNCk s
where s.id = t1.id
and s.key2 between 157573487 and 257573487
```

```
select sum(t.amount)
from pt_tx_CIid t2, #temp t
where t.id = t2.id
```

```
drop table #temp
```

Statistics Output: 2-Step Query

```
Table: pt_tx_CIamountNCamount  scan count 1,  logical
  reads: 88,  physical reads: 0
Table: pt_sample_CIidNCk  scan count 7282,  logical
  reads: 22152,  physical reads: 0
Table: #temp_______________00024884F  scan count 0,
  logical reads: 554,  physical reads: 0


Table: pt_tx_CIid  scan count 550,  logical reads:
  1109,  physical reads: 0
Table: #temp_______________00024884F  scan count 1,
  logical reads: 5,  physical reads: 0
```

Soaring Eagle Consulting

Statistics Output: 2-Step Query Continued

Note:

Remember that temporary table modifications are mostly logical writes (unless you run out of cache).

Showplan Output :2-Step Query

SHOWPLAN OUTPUT

```
 STEP 1
     The type of query is CREATE TABLE.

 STEP 2
     The type of query is INSERT.

4 operator(s) under root

     |ROOT:EMIT Operator
     |
     |    |INSERT Operator
     |    |  The update mode is direct.
     |    |
     |    |    |NESTED LOOP JOIN Operator (Join Type: Inner Join)
     |    |    |
     |    |    |    |SCAN Operator
     |    |    |    |  FROM TABLE
     |    |    |    |  pt_sample_CIidNCk
     |    |    |    |  s
     |    |    |    |  Index : NCK
     |    |    |    |  Forward Scan.
     |    |    |    |  Positioning by key.
     |    |    |    |  Keys are:
     |    |    |    |    key2 ASC
     |    |    |    |  Using I/O Size 2 Kbytes for index leaf pages.
     |    |    |    |  With LRU Buffer Replacement Strategy for index leaf pages.
```

Showplan Output :2-Step Query

SHOWPLAN OUTPUT

```
 |    |    |    |  Using I/O Size 2 Kbytes for data pages.
        |    |    |    |  With LRU Buffer Replacement Strategy for data pages.
        |    |    |
        |    |    |    |SCAN Operator
        |    |    |    |  FROM TABLE
        |    |    |    |  pt_tx_NCamount
        |    |    |    |  t1
        |    |    |    |  Table Scan.
        |    |    |    |  Forward Scan.
        |    |    |    |  Positioning at start of table.
        |    |    |    |  Using I/O Size 2 Kbytes for data pages.
        |    |    |    |  With LRU Buffer Replacement Strategy for data pages.
        |    |
        |    |  TO TABLE
        |    |  #temp
        |    |  Using I/O Size 2 Kbytes for data pages.
```

Soaring Eagle Consulting

Showplan Output :2-Step Query

SHOWPLAN OUTPUT

```
QUERY PLAN FOR STATEMENT 1 (at line 1).


    STEP 1
        The type of query is SELECT.

   4 operator(s) under root

       |ROOT:EMIT Operator
       |
       |   |SCALAR AGGREGATE Operator
       |   |  Evaluate Ungrouped SUM OR AVERAGE AGGREGATE.
       |   |
       |   |   |NESTED LOOP JOIN Operator (Join Type: Inner Join)
       |   |   |
       |   |   |   |SCAN Operator
       |   |   |   |  FROM TABLE
       |   |   |   |  #temp
       |   |   |   |  t
       |   |   |   |  Table Scan.
       |   |   |   |  Forward Scan.
       |   |   |   |  Positioning at start of table.
       |   |   |   |  Using I/O Size 2 Kbytes for data pages.
       |   |   |   |  With LRU Buffer Replacement Strategy for data pages.
```

Showplan Output :2-Step Query

SHOWPLAN OUTPUT

```
 |    |    |
           |    |    |    |SCAN Operator
           |    |    |    |  FROM TABLE
           |    |    |    |  pt_tx_CIid
           |    |    |    |  t2
           |    |    |    |  Using Clustered Index.
           |    |    |    |  Index : ci
           |    |    |    |  Forward Scan.
           |    |    |    |  Positioning by key.
           |    |    |    |  Keys are:
           |    |    |    |    id ASC
           |    |    |    |  Using I/O Size 2 Kbytes for data pages.
           |    |    |    |  With LRU Buffer Replacement Strategy for data pages.
```

Soaring Eagle Consulting

Breaking Up Large Queries Summary

Recommendations

* Understand outer/inner processing with multi-table joins
* Look for situations where the optimizer does not use all available indexing resources
* Look for situations where nested scans of large tables are occurring
* Break up queries by building temporary result sets
* Try to encourage the optimizer to make use of many indexes
* Use the first step to reduce the number of rows being processed
* Be careful of I/O contention in tempdb

Soaring Eagle Consulting

Self-Joins

* A table can be joined to itself to find inter-related information
* There are two general types of self-joins
  * Parent/Child relationships in one table
  * Matching values within a single column
* Table aliases are required with all self-joins to provide two pointers to a single table

```
/* display authors living in the same city */
select a1.city, a1.au_lname, a1.au_fname, a2.au_lname,
a2.au_fname
from authors a1, authors a2
where a1.city = a2.city
and a1.au_id > a2.au_id
```

Self-Joins (1): Parent/Child

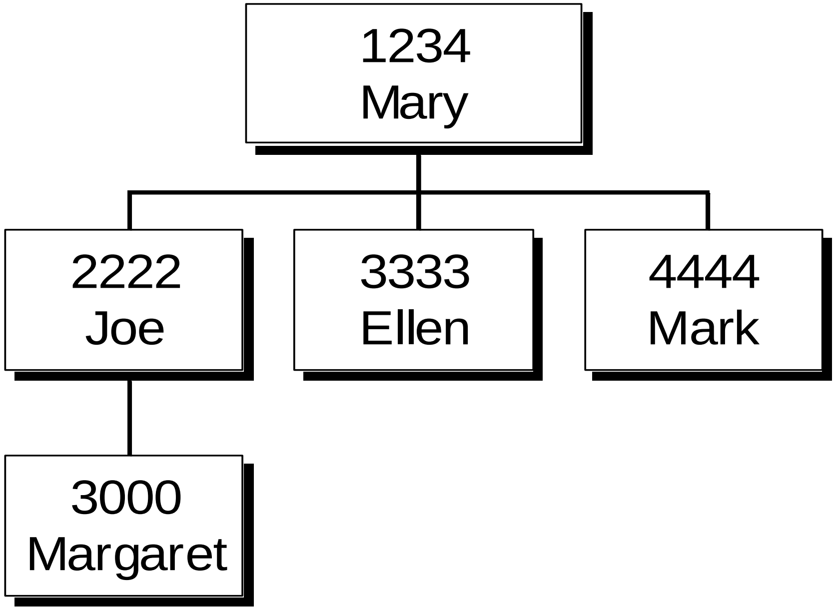
* Use a self-join when a single table contains parent and child values
* Parent/Child relationships exist when a row in a table stores an identifier for another row in the table

Examples of Parent/Child Relationships

- **Manager/Employee**
- **Kit/Part**
- **Parent/Child**

Manager/Employee Relationship

* Consider this organization chart



Soaring Eagle Consulting

Manager/Employee Relationship Continued

- To store each employee and each relationship within one table, you need three columns
- You need to write a self-join to display names of employees and managers on a single line

Id	name	mgr_id
1234	Mary	(NULL)
2222 3333	Joe Ellen	1234 1234
4444 3000	Mark Margaret	1234 2222

Recipe: Self-Join (1), Parent/Child

* Write the from clause first, identifying two copies of the table. Use meaningful table aliases to help you keep the two copies straight in your mind (you'll use these aliases in the next step)

```
from employees emp, employees mgr
```

* Now decide which columns you need from each copy of the table. Use column aliases to identify duplicate columns in the select list

```
select emp.id, emp.name, mgr.name "manager"
```

SOARING EAGLE CONSULTING

Recipe: Self-Join (1), Parent/Child

* Finally, write a where clause to join the tables. If you used meaningful aliases, it shouldn't be too hard to get the relationship right

```
where emp.mgr_id = mgr.id
```

Recipe: Self-Join (1), Parent/Child

* Test the select statement to make sure you got all of the relationships right. (It's fairly common to get the relationship backwards, which would give you all managers for the employees instead of all employees for each manager)

```
select emp.id, emp.name, mgr.name "manager"
from employees emp, employees mgr
where emp.mgr_id = mgr.id

id          name         manager
----------- ------------ ------------
       2222 Joe          Mary
       3333 Ellen        Mary
       4444 Mark         Mary
       3000 Margaret     Joe
```

Self-Joins (2): Matching Data

```
select a1.au_lname, a1.au_fname,
       a2.au_lname, a2.au_fname, a1.city
from authors a1, authors a2
where a1.city = a2.city
```

authors a1

	_lname	au_fname	city
172-32-1176	White	Johnson	Menlo Park
213-46-8915	Green	Marjorie	Oakland
...	...	...	...
893-72-1158	McBadden	Heather	Vacaville
899-46-2035	Ringer	Anne	Salt Lake City
998-72-3567	Ringer	Albert	Salt Lake City

a1.city = a2.city

authors a2

	_lname	au_fname	city
172-32-1176	White	Johnson	Menlo Park
213-46-8915	Green	Marjorie	Oakland
...	...	...	...
893-72-1158	McBadden	Heather	Vacaville
899-46-2035	Ringer	Anne	Salt Lake City
998-72-3567	Ringer	Albert	Salt Lake City

Remove these duplicates using a primary (unique) key:
and a1.au_id <> a2.au_id

a1.au_lname	a1.au_fname	a2.au_lname	a2.au_fname	a1.city
Ringer	Anne	Ringer	Anne	Salt Lake City
Ringer	Anne	Ringer	Albert	Salt Lake City
Ringer	Albert	Ringer	Anne	Salt Lake City
Ringer	Albert	Ringer	Albert	Salt Lake City

- Use a self-join to identify rows sharing column values
- In this example, search for authors from the same city

Recipe: Self-Join (2), Matching Data

1. Identify the columns you need in the result, identify which copy of the table each column will come from, and include column names qualified by an alias in the select list

```
/* display authors living
  in the same city*/
  select a1.au_id,
  a2.au_id, a1.au_lname,
  a1.au_fname,a2.au_lname,
  a2.au_fname
```

2. Include the table name twice, with different aliases (remember, aliases are required here)

```
from authors a1, authors a2
```

3. Join the tables on the common field, which usually is not a unique key

```
where a1.city = a2.city
```

Soaring Eagle Consulting

Recipe: Self-Join (2), Matching Data

4. Add a condition to prevent a row from joining with itself (usually with the unique key)

```
and a1.au_id <> a2.au_id
```

5. If you are displaying data from both copies of the table in the select list, add a condition to suppress duplicate rows

```
and a1.au_id > a2.au_id
```

You should be able to combine steps 4 and 5 in a single condition

Self-Joins: Notes

* Self-joins can be combined with all other SQL syntax
* Sometimes the matching data type of self-join can be avoided by using count(*) with GROUP BY and HAVING

```
/* show all cities
       where there is more than one author */
select city, count(*)
from authors
group by city
having count(*) > 1
```

Soaring Eagle Consulting

Self-Joins: Notes Continued

* If you need to know which authors live in cities with other authors, you can list them like this:

```
/* show names of authors living in cities
      with many authors */
select au_lname, au_fname, city
from authors
where city in
      (select city
      from authors
      group by city
      having count(*) > 1)
```

Note: The count (*) column does not appear in the select list of the subquery!

Soaring Eagle Consulting

Using Temporary Tables to Eliminate Self-Joins

- You can use a temporary table to eliminate a self-join by using the aggregate method above to create a lookup table of duplicate keys, then refer to the lookup table in the next step

```
select city
into #t
from authors
group by city
having count(*) > 1

select au_lname, au_fname, a.city
from authors a, #t
where a.city = #t.city

drop table #t
```

Using Temporary Tables to Eliminate Self-Joins

* This method could be more efficient (depending on table characteristics) because joins are often more efficient than subqueries

Outer Joins

* Outer joins permit the display of all values from one table, whether or not there is related information in the second table
* To write an outer join, use the *= or =* operator

```
/* quarterly sales of all titles */
select t.title, sum(qty) total_qty
from titles t, salesdetail sd
where t.title_id *= sd.title_id
and stor_id = "5023"
group by t.title
order by sum(qty) desc
```

Outer Joins Continued

- The asterisk (*) forces all values for a table to be displayed
- Double outer joins (*=*) are not supported

```
title          total_qty
--------       -----------
MC3021         19850
BU2075         17700
TC4203         11750
...
TC3218         85
MC3026         (NULL)
PC9999         (NULL)
PS2106         (NULL)
```

In this example, total sales for titles not sold to store "5023" are displayed as null

Outer Joins
with Parent-Child Self Joins

- Consider the output from the outer join example above

Query

```
select emp.id, emp.name, mgr.name "manager"
from employees emp, employees mgr
where emp.mgr_id = mgr.id
```

Result

```
id          name         manager
----------- ------------ ------------
       2222 Joe          Mary
       3333 Ellen        Mary
       4444 Mark         Mary
       3000 Margaret     Joe
```

Soaring Eagle Consulting

Continued next page

Outer Joins with Parent-Child Self Joins

- Question
- Where is the value for Mary?

* An outer join finds Mary's row, which does not join to any row on the Manager ID value

```
select emp.id, emp.name, mgr.name "manager"
from employees emp, employees mgr
where emp.mgr_id *= mgr.id
```

```
id          name         manager
----------- ------------ ------------
       1234 Mary         NULL
       2222 Joe          Mary
       3333 Ellen        Mary
       4444 Mark         Mary
       3000 Margaret     Joe
```

SOARING EAGLE CONSULTING

Simulating Double-Outer Joins

* Adaptive Server does not support a double outer join (*=*), but you can simulate the double outer join with the UNION statement

Query

```
select emp.id, emp.name, mgr.name "manager"
from employees emp, employees mgr
where emp.mgr_id *= mgr.id
UNION
select emp.id, emp.name, mgr.name "manager"
from employees emp, employees mgr
where emp.mgr_id =* mgr.id
```

Soaring Eagle Consulting

Simulating Double-Outer Joins Continued

- Adaptive Server does not support a double outer join (*=*), but you can simulate the double outer join with the UNION statement

Result

Id1	name	manager
1234	Mary	(NULL)
(NULL) (NULL)	(NULL) (NULL)	Mark Ellen
(NULL) 2222 4444	(NULL) Joe Mark	Margaret Mary Mary
3333 3000	Ellen Margaret	Mary Joe

Soaring Eagle Consulting

Problems with Outer Joins

Outer joins limit possible multi-table joins

* This query seeks to display sales of titles to stores in California, listing all titles (even those with no sales)

```
select title, sum(qty)
from titles t, salesdetail sd, stores s
where t.title_id *= sd.title_id
and sd.stor_id = s.stor_id
and s.state = "CA"
group by title
```

Soaring Eagle Consulting

Problems with Outer Joins

* The query is illegal and the Adaptive Server returns error message 303:

* To solve this problem, we need to understand the optimization plan for an outer join

```
The table 'salesdetail' is an inner member of an outer-
join clause. This is not allowed if the table also
participates in a regular join clause.
```

Outer Join Optimization

```
select title, sum(qty)
from titles t, salesdetail sd
where t.title_id *= sd.title_id
group by title
```

Soaring Eagle Consulting

Outer Join Optimization: Showplan

SHOWPLAN OUTPUT

```
 STEP 1
     The type of query is SELECT.

5 operator(s) under root

     |ROOT:EMIT Operator
     |
     |    |HASH VECTOR AGGREGATE Operator
     |    |  GROUP BY
     |    |  Evaluate Grouped SUM OR AVERAGE AGGREGATE.
     |    | Using Worktable3 for internal storage.
     |    |  Key Count: 1
     |    |
     |    |    |MERGE JOIN Operator (Join Type: Left Outer Join)
     |    |    | Using Worktable2 for internal storage.
     |    |    |  Key Count: 1
     |    |    |  Key Ordering: ASC
     |    |    |
     |    |    |    |SCAN Operator
     |    |    |    |  FROM TABLE
     |    |    |    |  titles
     |    |    |    |  t
     |    |    |    |
```

Soaring Eagle Consulting

Outer Join Optimization: Showplan

SHOWPLAN OUTPUT

```
|    |    |
        |    |    |    |SCAN Operator
        |    |    |    |  FROM TABLE
        |    |    |    |  titles
        |    |    |    |  t
        |    |    |    |  Table Scan.
        |    |    |    |  Forward Scan.
        |    |    |    |  Positioning at start of table.
        |    |    |    |  Using I/O Size 2 Kbytes for data pages.
        |    |    |    |  With LRU Buffer Replacement Strategy for data pages.
        |    |    |
        |    |    |    |SORT Operator
        |    |    |    | Using Worktable1 for internal storage.
        |    |    |    |
        |    |    |    |    |SCAN Operator
        |    |    |    |    |  FROM TABLE
        |    |    |    |    |  salesdetail
        |    |    |    |    |  sd
        |    |    |    |    |  Table Scan.
        |    |    |    |    |  Forward Scan.
        |    |    |    |    |  Positioning at start of table.
        |    |    |    |    |  Using I/O Size 2 Kbytes for data pages.
        |    |    |    |    |  With LRU Buffer Replacement Strategy for data pages.
```

Outer Join Optimization Continued

* The outer join forces Adaptive Server to use a join order where the outside table (with the *) is processed before the inside table
* In this query, the optimizer chooses a more expensive plan because of the constraints of using an outer joins

```
select s.id, count(*)
from pt_tx_CIid t, pt_sample_CIcompany s
where t.id *= s.id
group by s.id
```

Soaring Eagle Consulting

Outer Join Optimization Showplan

SHOWPLAN OUTPUT

```
QUERY PLAN FOR STATEMENT 1 (at line 1).


    STEP 1
        The type of query is SELECT.

  5 operator(s) under root

        |ROOT:EMIT Operator
        |
        |    |HASH VECTOR AGGREGATE Operator
        |    |  GROUP BY
        |    |  Evaluate Grouped COUNT AGGREGATE.
        |    | Using Worktable3 for internal storage.
        |    |  Key Count: 1
        |    |
```

Outer Join Optimization Showplan

```
 |    |
        |    |    |MERGE JOIN Operator (Join Type: Left Outer Join)
        |    |    | Using Worktable2 for internal storage.
        |    |    |  Key Count: 1
        |    |    |  Key Ordering: ASC
        |    |    |
        |    |    |    |SCAN Operator
        |    |    |    |  FROM TABLE
        |    |    |    |  pt_tx_CIid
        |    |    |    |  t
        |    |    |    |  Table Scan.
        |    |    |    |  Forward Scan.
        |    |    |    |  Positioning at start of table.
        |    |    |    |  Using I/O Size 2 Kbytes for data pages.
        |    |    |    |  With LRU Buffer Replacement Strategy for data
   pages.
```

SHOWPLAN OUTPUT

Outer Join Optimization Showplan

```
|    |    |
        |    |    |    |SORT Operator
        |    |    |    | Using Worktable1 for internal storage.
        |    |    |    |
        |    |    |    |    |SCAN Operator
        |    |    |    |    |   FROM TABLE
        |    |    |    |    |   pt_sample_CICompany
        |    |    |    |    |   s
        |    |    |    |    |   Table Scan.
        |    |    |    |    |   Forward Scan.
        |    |    |    |    |   Positioning at start of table.
        |    |    |    |    |   Using I/O Size 2 Kbytes for data pages.
        |    |    |    |    |   With LRU Buffer Replacement Strategy for
    data pages.
```

Soaring Eagle Consulting

Outer Join Optimization Continued

```
STEP 2
  The type of query is SELECT (into Worktable1).
  GROUP BY
  Evaluate Grouped COUNT AGGREGATE.


  FROM TABLE
      pt_tx_CIid
      t
  Nested iteration.
  Table Scan.
  Ascending scan.
  Positioning at start of table.
  Using I/O Size 2 Kbytes.
  With LRU Buffer Replacement Strategy.
```

SHOWPLAN OUTPUT

Outer Join Optimization

```
QUERY PLAN FOR STATEMENT 1 (at line 1).


    STEP 1
        The type of query is SELECT.

    5 operator(s) under root

       |ROOT:EMIT Operator
       |
       |   |HASH VECTOR AGGREGATE Operator
       |   |  GROUP BY
       |   |  Evaluate Grouped COUNT AGGREGATE.
       |   | Using Worktable3 for internal storage.
       |   |  Key Count: 1
       |   |
       |   |   |MERGE JOIN Operator (Join Type: Left Outer Join)
       |   |   | Using Worktable2 for internal storage.
       |   |   |  Key Count: 1
       |   |   |  Key Ordering: ASC
       |   |   |
```

Soaring Eagle Consulting

Outer Join Optimization

```
|    |    |    |SORT Operator
       |    |    |    | Using Worktable1 for internal storage.
       |    |    |    |
       |    |    |    |    |SCAN Operator
       |    |    |    |    |   FROM TABLE
       |    |    |    |    |   pt_sample_CICompany
       |    |    |    |    |   s
       |    |    |    |    |   Table Scan.
       |    |    |    |    |   Forward Scan.
       |    |    |    |    |   Positioning at start of table.
       |    |    |    |    |   Using I/O Size 2 Kbytes for data pages.
       |    |    |    |    |   With LRU Buffer Replacement Strategy
for data pages.
```

Note: Even though the server has a clustered index on id in one of the tables, the server chooses to reformat the query

Outer Join Forces a Join Order

```
*********************************************************************
*********
          BEGIN: Search Space Traversal for OptBlock1
*********************************************************************
*********

Scan plans selected for this optblock:
```

Show_search_engine

```
OptBlock1 Eqc{1} -> Pops added:

        ( PopIndScan ci pt_tx_CIid t ) cost: 6269 T(L122,P121,C30000)
O(L122,P121,C30000) props: [{1}]


OptBlock1 Eqc{2} -> Pops added:

        ( PopIndScan CICompany pt_sample_CICompany s ) cost: 7498
T(L224,P222,C15000) O(L224,P222,C15000) props: [{}]

        ( PopSort ( PopIndScan CICompany pt_sample_CICompany s ) )
cost: 13150.6 T(L246,P256,C62585.96) O(L22,P34,C47585.96) props:
[{1}]
```

Soaring Eagle Consulting

Outer Join Forces a Join Order

Show_search_engine

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
        BEGIN: APPLYING EXHAUSTIVE SEARCH STRATEGY TO OBTAIN BEST
PLAN
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~

======================================================================
       BEGIN: Left deep tree evaluation (tree #1)
======================================================================

Tree shape: (2(1)(1))


xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    BEGIN: Complete join order evaluation (perm #1)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Slide 1 0f 8

Soaring Eagle Consulting

Outer Join Forces a Join Order

Show_search_engine

```
Permutation Order: Gt1 |X| Gt2


Join plans selected for this permutation:

OptBlock1 Eqc{1,2} -> Pops added for the join Eqc{1} - Eqc{2}:

        ( PopMergeOuterJoin ( PopIndScan ci pt_tx_CIid t ) ( PopSort ( PopIndScan CICompany pt_sample_CICompany s ) ) ) cost: 21419.6 T(L368,P377,C112586) O(L0,P0,C20000) props: [{}]

        ( PopSort ( PopMergeOuterJoin ( PopIndScan ci pt_tx_CIid t ) ( PopSort ( PopIndScan CICompany pt_sample_CICompany s ) ) ) ) cost: 27072.19 T(L390,P411,C160171.9) O(L22,P34,C47585.96) props: [{5}]
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    DONE: Complete join order evaluation (perm #1)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Outer Join Forces a Join Order

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    BEGIN: Complete join order evaluation (perm #2)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Permutation Order: Gt2 |X| Gt1

!! This complete join order has been found to be:

Invalid by a heuristic rule

!! Rejecting it !!


xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    DONE: Complete join order evaluation (perm #2)
Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Slide 1 0f 8

SOARING EAGLE CONSULTING

Outer Join Forces a Join Order

```
=====================================================================
      DONE: Left deep tree evaluation (tree #1)
=====================================================================


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
        DONE: APPLYING EXHAUSTIVE SEARCH STRATEGY TO OBTAIN BEST PLAN
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~

------------------------------------------
Search Engine Statistics (Summary)
------------------------------------------

Total number of tree shapes considered:1
Number of major tree shapes generated:1
Number of tree shapes generated by flipping major tree shapes:0

Number of valid complete plans evaluated:1
Total number of complete plans evaluated:2

------------------------------------------
```

Show_search_engine

Soaring Eagle Consulting

Outer Join Forces a Join Order

Show_search_engine

```
The best plan found in OptBlock1 :

( PopMergeJoin cost: 21419.6 props: [{}] ( PopIndScan cost: 6269
T(L122,P121,C30000) O(L122,P121,C30000) props: [{1}] Gti1( ci ) Gtt1(
pt_tx_CIid t ) ) cost: 6269 T(L122,P121,C30000) O(L122,P121,C30000)
props: [{1}]
( PopSort cost: 13150.6 T(L246,P256,C62585.96) O(L22,P34,C47585.96)
props: [{1}] ( PopIndScan cost: 7498 T(L224,P222,C15000)
O(L224,P222,C15000) props: [{}] Gti2( CICompany ) Gtt2(
pt_sample_CICompany s ) ) cost: 7498 T(L224,P222,C15000)
O(L224,P222,C15000) props: [{}]
) cost: 13150.6 T(L246,P256,C62585.96) O(L22,P34,C47585.96) props:
[{1}]
) cost: 21419.6 props: [{}]

( PopSort cost: 27072.19 props: [{5}] ( PopMergeJoin cost: 21419.6
props: [{}] ( PopIndScan cost: 6269 T(L122,P121,C30000)
O(L122,P121,C30000) props: [{1}] Gti1( ci ) Gtt1( pt_tx_CIid t ) )
cost: 6269 T(L122,P121,C30000) O(L122,P121,C30000) props: [{1}]
( PopSort cost: 13150.6 T(L246,P256,C62585.96) O(L22,P34,C47585.96)
props: [{1}] ( PopIndScan cost: 7498 T(L224,P222,C15000)
O(L224,P222,C15000) props: [{}] Gti2( CICompany ) Gtt2(
pt_sample_CICompany s ) ) cost: 7498 T(L224,P222,C15000)
O(L224,P222,C15000) props: [{}]
) cost: 13150.6 T(L246,P256,C62585.96) O(L22,P34,C47585.96) props:
[{1}]
) cost: 21419.6 props: [{}]
) cost: 27072.19 props: [{5}]
```

Slide 1 0f 8

Outer Join Forces a Join Order

Show_search_engine

```
**********************************************
*********
          DONE: Search Space Traversal for OptBlock1
**********************************************************************
*********

Checking for Deep PushDown the Pop tree:
( PopMergeOuterJoin ( PopIndScan ci pt_tx_CIid t ) ( PopSort (
PopIndScan CICompany pt_sample_CICompany s ) ) ) cost: 21419.6 props:
[{}]
done.
Checking for Deep PushDown the Pop tree:
( PopSort ( PopMergeOuterJoin ( PopIndScan ci pt_tx_CIid t ) (
PopSort ( PopIndScan CICompany pt_sample_CICompany s ) ) ) ) cost:
27072.19 props: [{5}]
done.

**********************************************************************
*********
          BEGIN: Search Space Traversal for OptBlock0
**********************************************************************
*********

OptBlock0 Eqc{0} -> Pops added:
```

Slide 1 0f 8

Outer Join Forces a Join Order

Show_search_engine

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~
         BEGIN: APPLYING GREEDY SEARCH STRATEGY TO OBTAIN INITIAL PLAN
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~
OptBlock0 Eqc{0} -> Pops added:



** Costing set up for RowLimit optimization **

Scan plans selected for this optblock:


OptBlock0 Eqc{0} -> Pops added:

        ( PopGroupHashing ( PopMergeOuterJoin ( PopIndScan ci pt_tx_CIid
t ) ( PopSort ( PopIndScan CICompany pt_sample_CICompany s ) ) ) ) cost:
21929.6 T(L373,P377,C117586) O(L5,P0,C5000) props: [{}]



~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~
         DONE: APPLYING GREEDY SEARCH STRATEGY TO OBTAIN INITIAL PLAN
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~
```

Slide 1 0f 8

Outer Join Forces a Join Order

Show_search_engine

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~
        BEGIN: APPLYING EXHAUSTIVE SEARCH STRATEGY TO OBTAIN BEST PLAN
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
        DONE: APPLYING EXHAUSTIVE SEARCH STRATEGY TO OBTAIN BEST PLAN
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~
The best plan found in OptBlock0 :

( PopGroupHashing cost: 21929.6 T(L373,P377,C117586) O(L5,P0,C5000)
props: [{}] ( PopMergeOuterJoin cost: 21419.6 T(L368,P377,C112586)
O(L0,P0,C20000) props: [{}] ( PopIndScan cost: 6269 T(L122,P121,C30000)
O(L122,P121,C30000) props: [{1}] Gti1( ci ) Gtt1( pt_tx_CIid t ) ) cost:
6269 T(L122,P121,C30000) O(L122,P121,C30000) props: [{1}]
( PopSort cost: 13150.6 T(L246,P256,C62585.96) O(L22,P34,C47585.96)
props: [{1}] ( PopIndScan cost: 7498 T(L224,P222,C15000)
O(L224,P222,C15000) props: [{}] Gti2( CICompany ) Gtt2(
pt_sample_CICompany s ) ) cost: 7498 T(L224,P222,C15000)
O(L224,P222,C15000) props: [{}]
) cost: 13150.6 T(L246,P256,C62585.96) O(L22,P34,C47585.96) props: [{1}]
) cost: 21419.6 T(L368,P377,C112586) O(L0,P0,C20000) props: [{}]
) cost: 21929.6 T(L373,P377,C117586) O(L5,P0,C5000) props: [{}]

************************************************************************
*****
          DONE: Search Space Traversal for OptBlock0
```

Slide 1 of 8

Handling a Multi-Table Join

- To handle a multi-table join, you need to take two steps

In the first step, resolve the inner query to a worktable

```
select title_id, sum(qty) “sum_qty”
into #worktable
from salesdetail sd, stores s
where sd.stor_id = s.stor_id
and s.state = "CA"
group by title_id
```

In the second step, perform the outer join

```
select title, sum_qty
from titles t, #worktable w
where t.title_id *= w.title_id
order by title

drop table #worktable
```

Soaring Eagle Consulting

Merge Join

* Is a variation of join that takes advantage of sorted data and performs an interleaved scan of the data.
* By using data sorted on the join clause, the joins processing will encounter and generate tuples in the same order
* It is said that Merge joins can produce results almost 500x faster than via traditional Nested loops

```
select s.id, count(*)
from pt_tx_CIid t, pt_sample_CICompany s
where t.id = s.id
group by s.id
```

Soaring Eagle Consulting

Merge Join

```
QUERY PLAN FOR STATEMENT 1 (at line 1).


    STEP 1
        The type of query is SELECT.

  5 operator(s) under root

       |ROOT:EMIT Operator
       |
       |   |GROUP SORTED Operator
       |   |  Evaluate Grouped COUNT AGGREGATE.
       |   |
       |   |   |MERGE JOIN Operator (Join Type: Inner Join)
       |   |   | Using Worktable2 for internal storage.
       |   |   |  Key Count: 1
       |   |   |  Key Ordering: ASC
       |   |   |
       |   |   |   |SORT Operator
       |   |   |   | Using Worktable1 for internal storage.
       |   |   |   |
       |   |   |
```

Merge Join

```
|   |   |   |
       |   |   |   |   |SCAN Operator
       |   |   |   |   |  FROM TABLE
       |   |   |   |   |  pt_sample_CICompany
       |   |   |   |   |  s
       |   |   |   |   |  Table Scan.
       |   |   |   |   |  Forward Scan.
       |   |   |   |   |  Positioning at start of table.
       |   |   |   |   |  Using I/O Size 2 Kbytes for data pages.
       |   |   |   |   |  With LRU Buffer Replacement Strategy for
 data pages.
       |   |   |
       |   |   |   |SCAN Operator
       |   |   |   |  FROM TABLE
       |   |   |   |  pt_tx_CIid
       |   |   |   |  t
       |   |   |   |  Table Scan.
       |   |   |   |  Forward Scan.
       |   |   |   |  Positioning at start of table.
       |   |   |   |  Using I/O Size 2 Kbytes for data pages.
```

- ```
         |   |   |   |  With LRU Buffer Replacement Strategy for
  data pages.
  ```
  ```

Hash Join

* Is a useful join algorithm for dealing with joining tables without usable indexes
* A hash join will scan each table to generate a hash table of rows that meet the join criteria. Then hash tables are joined to produce the results.
* This means that the server will ultimately scan all the tables in the join before producing any results.
* Hash joins can be affected by the amount of available memory in the server
* Hash joins can be performed left deep, right deep or bushy

Hash Join

```
STEP 1
    The type of query is SELECT.
            3 operator(s) under root

   |ROOT:EMIT Operator
   |
   |   |HASH JOIN Operator (Join Type: Inner Join)
   |   | Using Worktable1 for internal storage.
   |   |  Key Count: 1
   |   |
   |   |   |SCAN Operator
   |   |   |  FROM TABLE
   |   |   |  pt_sample
   |   |   |  s
   |   |   |  Table Scan.
   |   |   |  Forward Scan.
   |   |   |  Positioning at start of table.
   |   |   |  Using I/O Size 2 Kbytes for data pages.
   |   |   |  With LRU Buffer Replacement Strategy for data pages  |   |
   |   |   |
```

Hash Join

```
| | |SCAN Operator
   | | | FROM TABLE
   | | | pt_tx
   | | | t
   | | | Table Scan.
   | | | Forward Scan.
   | | | Positioning at start of table.
   | | | Using I/O Size 2 Kbytes for data pages.
   | | | With LRU Buffer Replacement Strategy for data pages.
```

SOARING EAGLE CONSULTING

Optimization Criteria

* The various join/optimization techniques can be enabled/disabled at the session level using a set command
  * Joins
    * hash_join, merge_join, nl_join
    * bushy_search_space – allow bushy-tree-shaped query plans vs. nli
  * Union/Union All
    * append_union_all, merge_union_all,
    * hash_union_distinct , merge_union_distinct
  * Reformatting
    * store_index , multi_table_store_ind

Soaring Eagle Consulting

Optimization Criteria Continued

* Distinct() and Group By
  * opportunistic_distinct_view – flexibility for enforcing distinctness
  * distinct_hashing, distinct_sorted, distinct_sorting
  * group_hashing, group-sorted
* Parallelism
  * parallel_query, index_intersection

Soaring Eagle Consulting

ASE 15.0 & Star Schema

* Star-Schema Optimization
  * Prior to ASE 15, not a lot of choice
    * Multiple composite indices on combinations of most often used dimensions
    * Query plan forcing to access dimensions first
      * Cartesian across dimensions
  * Competitive implementations
    * Some use cartesian of dimensions as above
  * Part of the problem is space consumed by indices
    * Not to mention index maintenance, update statistics, etc.
* ASE 15.0 supports star-schema
  * Recognizes dimensions/fact-table
  * Uses index intersection/index union capability to avoid multiply defined composite indices.

Summary

- The Adaptive Server Optimizer typically to finds the most efficient way to perform multi-table queries
- Override the optimizer only when you know something the optimizer doesn't
- Break up large queries when the optimizer cannot find a good optimization plan
- Be aware of the likely cost of performing outer joins

Soaring Eagle Consulting

Summary – addendum

Opt goal	Nested Loop Join	Merge Join	Hash Join	N-ary NLJ	Parallel Query	Bushy Join	Star Schema	Eager Aggregation
ASE 12.5	✓				✓			
allrows_oltp	✓			✓				
allrows_mix	✓	✓		✓	✓			
allrows_dss	✓	✓	✓	✓	✓	✓	✓	✓

Lab 7.2: Join Processing

1. Break up the below query into two steps. (There are two ways – which, if either, is worth doing?)
2. Identify the costs of each.

```
select count(*)
from pt_tx t, pt_sample_CICompany s, pt_tx_CIid t2
where t.id = s.id
and t2.id = t.id
and t2.id = s.id
and s.company between 'bk' and 'br'
and t2.amount between $1000 and $5000
```

Thank You!

Let's take a few questions

Jeff Garbus

jeff@soaringeagle.biz

813. 641.3434

Contact Ray Rannala to register for your Free *Express Tune-up!*

ray@soaringeagle.biz – 941-981-3913

Soaring Eagle Consulting