Things You Should Know About 11g CBO

Dave Anderson
Founder and Oracle DBA, SkillBuilders
dave@skillbuilders.com

SkillBuilders.com/SQLTuning

Some Things I Really Hope You Want to Know About CBO!

Agenda

- Feedback-Based Optimization
- Direct Reads for Full Scans
- New Join Techniques
- Parallel Query Enhancements
- Null Aware Anti Joins
- DBMS_STATS Extended Stats
- SQL Plan Management

Feedback-Based Optimization

The bridge between the SQL
Engine and the CBO.

What's the Problem?

- Two issues have been particularly troublesome

- Inaccurate cardinality
  - Number of rows passed out of plan step
  - How to get better estimates ?

- Bind variables
  - Value not known until runtime
  - Value can change
    - Sometimes warrants new plan

Solutions Weren't "Perfect"

- Bind Variable Peeking
  - Plan based on 1st value used
- Dynamic Sampling
  - Cost, not automatic
- Hints
  - Too rigid, Things change
- Stored Outlines (deprecated)
  - Too fixed
- SQL Profiles
  - Provide adjustments for parameters, card, object statistics
  - Need to manually run "Automatic Tuning Advisor"
- Plans change - Bad Surprises?

11g Feedback-Based Optimization

- Optimizer can automatically get *runtime* data from SQL Engine
  - Precise cardinalities
  - Execution statistics
- Provides two new features
  1. Cardinality Feedback
  2. Adaptive Cursor Sharing

Cardinality Mis-Estimates

- Several reasons for bad cardinality estimates
  - Complex and / or multiple predicates
    - 11g Extended Statistics can help (more later)
  - Functions
    - 11g Extended Stats again...
  - Data skew
    - Histograms
  - Missing / old stats
  - Non-pushable / merge-able Views

card_demo1.sql

- One wrong estimate can snowball
  - Wrong access path, wrong join method

Cardinality Feedback

- "Suspicious" queries monitored
  - Cardinalities compared after 1st execution
  - If different, cursor marked for re-optimization
  - Injects hint into query to supply correct cardinality
- Feeds back just once
  - Monitoring disabled after 1st execution
  - Not intended to solve volatile data environments
  - Not intended to evolve plan over life of cursor
- Single-table cardinality feedback supported
  - Not join cardinality (yet?)
    - Card value returned would be for the join order, technique used
    - Too many variables
- Can disable
  - alter session set "_optimizer_use_feedback" = false;

card_demo2.sql

To Bind or Not to Bind....

- Literals
  - Prevent cursor reuse
  - Cause more parsing
  - Shared pool latch
  - Decrease scalability
  - Use more memory
- Bind variables
  - Increase cursor reuse
  - But can cause sub-optimal plans

9i / 10g Bind Variable Peeking

- CBO “peeks” at bind variable values on hard parse
  - *One-time only*
- Provides ability to determine selectivity
  - Better than guessing
- But repeated query executions use same plan
  - What if subsequent values warrant different plans?
  - Use literals in this case

11g Adaptive Cursor Sharing

- Queries with bind variables
  - Equality predicates on columns with histograms
  - Range predicates
- Marked “bind-sensitive”
  - Bind Profile created
    - Selectivity range stored for each bind variable
  - Query monitored with rowsource profiling
  - Sample rate decreases
  - Eventually turned off if no changes seen

*ACS_demo.
sql*

11g Adaptive Cursor Sharing

- Made “bind-aware” if feedback dictates
  - Only if significant difference in amount of data
    - rowsource profiling
  - Cursor not shared if new values are outside range
  - Reoptimizes
  - Will merge cursors if plan same, save cursor cache space
- Oracle now recommending cursor_sharing=force....
  - See blog by Oracle's Maria Colgan on August 2, 2010
  - I recommend intensive testing first

Full Table Scans Have Changed.

Yes, for better or worse.

Direct Reads for Serial Full Scans

- Serial full scan of "large" tables can be done with direct reads
  - Runtime feature, not CBO (but I thought I'd sneak it in here)
  - Plan will not reveal
- Pros
  - Reduced latches
  - Reduce flooding of BC
- Cons
  - "Selfish" reads
  - Segment level checkpoint required
  - Delayed block cleanouts
  - Use of DP can be unpredictable

Undocumented except Metalink Note 793845.1

"Adaptive Direct Reads"?

Direct Reads for Serial Full Scans

- What's large?
  - 5 * _small_table_threshold

```
Dave > select ksppstvl, ksppstvl*5
  2   from x$ksppi x, x$ksppcv y
  3  where (x.indx = y.indx)
  4    and ksppinm='_small_table_threshold';

KSPPSTVL                       KSPPSTVL*5
------------------------------ ----------
674                                  3370
```

- Other factors
  - BC size
  - Number of cached blocks (1/2?)
  - Number of dirty blocks (1/4?)
  - ASSM, others?

DPR_setup.sql
DPR_demo1.sql
DPR_demo2.sql

Direct Reads for Serial Full Scans

- Clumsy to disable
  - alter session set "_small_table_threshold" = 999999999;
  - alter session set events '10949 trace name context forever, level 1';
  - Create tables in KEEP cache?
    - But KEEP meant for small tables ...
- Compression
  - Can change reads to BC
  - (11g offers Advanced Compression)

Changes to Nested Loops Joins

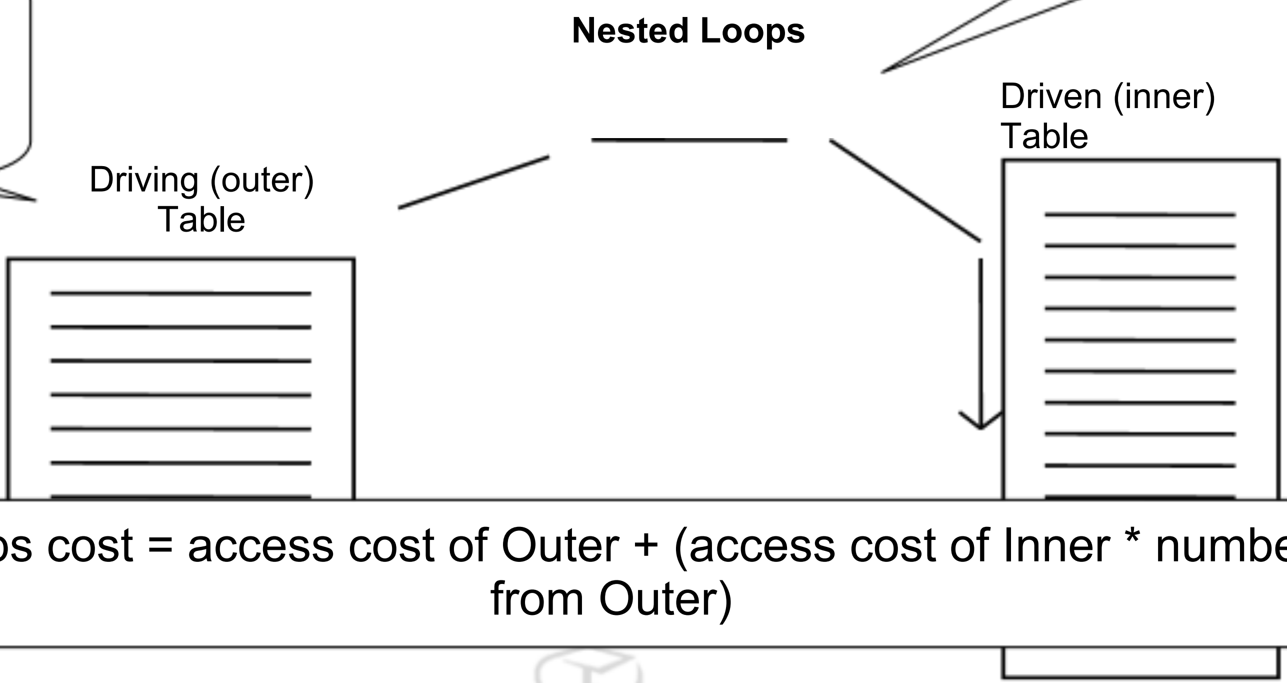
Another new technique for nested loops joins.

Traditional Nested Loops Join

- Common join technique for small-ish sets
- Starts returning rows quickly
  - No pre-processing of data

Repeated probes done via index

CBO will usually choose smaller table to drive join



Nested Loops cost = access cost of Outer + (access cost of Inner * number of rows from Outer)

Traditional Nested Loops Join

```
SQL> show parameter optimizer_features_enable

NAME TYPE VALUE
------------------------------------ ----------- --------
optimizer_features_enable string 11.2.0.1

SQL> explain plan for
2 select /*example1*/ small.object_id
3 from small, big
4 where small.object_id=big.object_id
5 and small.object_type='JAVA RESOURCE';


---------------------------------------------------------------------
| Id | Operation | Name | Rows | Bytes | Cost (
---------------------------------------------------------------------
| 0 | SELECT STATEMENT | | 3072 | 58368 | 17
| 1 | NESTED LOOPS | | 3072 | 58368 | 17
|* 2 | TABLE ACCESS FULL| SMALL | 6 | 84 | 5
|* 3 | INDEX RANGE SCAN | BIG_OBJECT_ID | 512 | 2560 | 2
---------------------------------------------------------------------
```

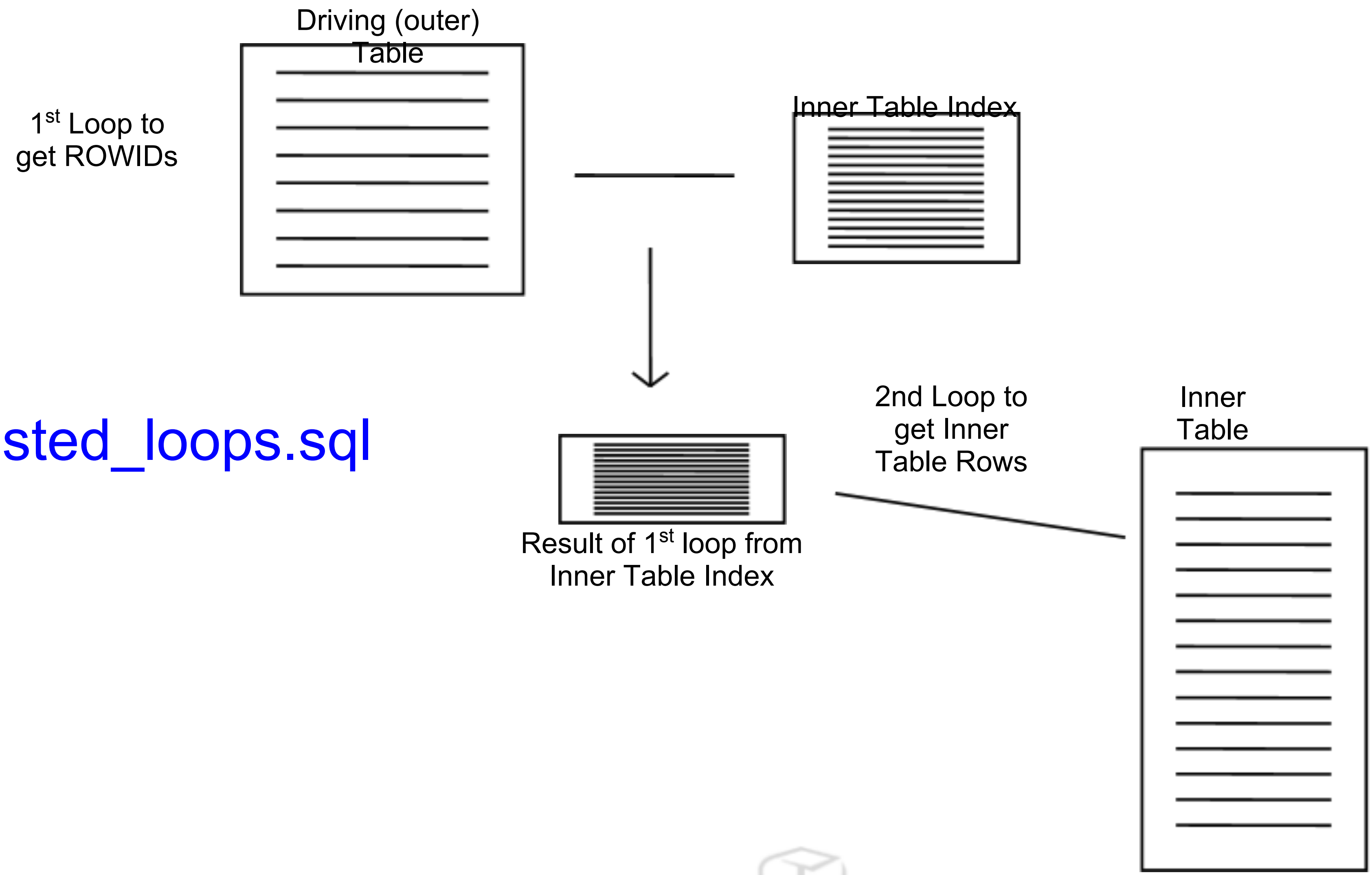
9i/10g Nested Loop

SQL> alter session set optimizer_features_enable = '10.2.0.4';

```
SQL> explain plan for
2 select small.object_id, small.object_type
3 from small, big
4 where small.object_id=big.object_id
5 and small.object_type='JAVA RESOURCE'
6 and big.status = 'VALID' ;

--------------------------------------------------------------------------
| Id | Operation | Name | Rows | Bytes | Cost
--------------------------------------------------------------------------
| 0  | SELECT STATEMENT | | 3072 | 76800 | 106
|* 1 | TABLE ACCESS BY INDEX ROWID| BIG | 512 | 5632 | 18
|  2 | NESTED LOOPS | | 3072 | 76800 | 106
|* 3 | TABLE ACCESS FULL | SMALL | 6 | 84 | 5
|* 4 | INDEX RANGE SCAN | BIG_OBJECT_ID | 512 | | 2
--------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
1 - filter("BIG"."STATUS"='VALID')
3 - filter("SMALL"."OBJECT_TYPE"='JAVA RESOURCE')
4 - access("SMALL"."OBJECT_ID"="BIG"."OBJECT_ID")
```

11g Nested Loops Join


Driving (outer) Table
1st Loop to get ROWIDs
Inner Table Index
2nd Loop to get Inner Table Rows
Inner Table
Result of 1st loop from Inner Table Index

nested_loops.sql

Parallel Query

11g Auto-DOP…

- Oracle decides
  - When to enable parallel execution
    - estimated elapsed time > parallel_min_time_threshold
  - DOP
    - Set based on current workload
- Also enables
  - In-Memory PX
    - Has option to read into buffer cache
  - Statement queuing
    - PQ's wait in FIFO queue for enough PX servers

…11g Auto-DOP

- Enable with
  - Parameters parallel_degree_policy=AUTO
    - Default is MANUAL (disabled)
  - Hint

```
select /*+ parallel(auto) */ order_date, sum(sale_amount)
from order_history
group by order_date
```

- Limit with
  - Parameter parallel_degree_limit
- On another note...
  - PQ now has auto dynamic sampling

DBMS_STATS

Extended Stats and other enhancements.

11g Extended Statistics

- Two common issues needed to be addressed
- Correlated columns, esp with skew
  - city / state
  - country / state
- Functions
  - LOWER(lastname)
- CBO has hard time estimating cardinality

ext_stats1.sql
ext_stats2.sql

```
dbms_stats.create_extended_stats(
    ownname=>user,
    tabname => 'CUSTOMER_HISTORY',
    extension => '(zip, work_zip)' )
```

Other Stuff

- New Sampling Algorithm
- Concurrent (parallel) Statistics Job
- Incremental Partition Stats
- Copy Partition Statistics
- Lock Stats at Partition Level
- GATHER_PREFERENCES Granularity
- Pending Statistics
- Compare to Pending Statistics

Null-Aware Anti-Joins

Quick Review: NOT IN Subquery: Anti-Joins

- Return rows from with no match in NOT IN subquery
- 3 kinds
  - Hash anti-join
  - Nested loops anti-join
  - Merge anti-join
- Can be very fast method of handling NOT IN
- Affected by _ALWAYS_ANTI_JOIN parameter

Nested Loops Anti-Join

```
select /* example1 */ small.object_id, small.owner
from small
where small.object_id not in
(select big.object_id from big)
and small.object_type='JAVA RESOURCE'

call count cpu elapsed disk query current
------- ------ -------- ---------- ---------- ---------- ----------
Parse 1 0.00 0.00 0 0 0
Execute 1 0.00 0.00 0 0 0
Fetch 1 0.00 0.09 21 27 0
------- ------ -------- ---------- ---------- ---------- ----------
total 3 0.00 0.10 21 27 0

Rows Row Source Operation
------- ---------------------------------------------------
0 NESTED LOOPS ANTI (cr=27 pr=21 pw=0 time=0 us cost=17 size=25
6 TABLE ACCESS FULL SMALL (cr=13 pr=11 pw=0 time=45 us cost=5
6 INDEX RANGE SCAN BIG_OBJECT_ID (cr=14 pr=10 pw=0 time=0 us c
```

Anti-Join Hints

```
select /* example3 */ small.object_id, small.owner
from small
where small.object_id not in
(select /*+ nl_aj */ big.object_id from big)

call count cpu elapsed disk query
------- ------ -------- ---------- ---------- ----------
Parse 1 0.00 0.00 0 0
Execute 1 0.00 0.00 0 0
Fetch 5 0.18 0.17 0 3149
------- ------ -------- ---------- ---------- ----------
total 7 0.18 0.17 0 3149

Rows Row Source Operation
------- ---------------------------------------------------
50 NESTED LOOPS ANTI
1000 TABLE ACCESS FULL SMALL
486400 INDEX RANGE SCAN BIG_OBJECT_ID (object id 33423)
```

Code hint in subquery

Problem: NOT IN with Nullable Subquery

```
SQL> alter session set optimizer_features_enable='10.2.0.4';

SQL> explain plan for
2 select customer.*
3 from customer
4 where id not in (select customer_id from orders);
```

Nullable column

```
Plan hash value: 3433771971

---------------------------------------------------------------------------------
| Id | Operation | Name           | Rows | Bytes | Cost (%CPU)| Time      |
---------------------------------------------------------------------------------
|  0  | SELECT STATEMENT |          | 265K | 36M   | 287M (2)   | 546:01:28 |
|* 1  | FILTER                   |     |       |            |
|  2  | TABLE ACCESS FULL| CUSTOMER | 265K | 36M    | 1381 (2)   | 00:00:10  |
|* 3  | TABLE ACCESS FULL| ORDERS   |     2 | 10     | 1128 (2)   | 00:00:08  |
---------------------------------------------------------------------------------
```

Ouch!

```
Predicate Information (identified by operation id):
---------------------------------------------------
1 - filter( NOT EXISTS (SELECT 0 FROM "ORDERS" "ORDERS" WHERE LNNVL("CUSTOMER_ID"<>:B1)))
3 - filter(LNNVL("CUSTOMER_ID"<>:B1))
```

Solution: 11g Null-Aware Antijoins

```
alter session set optimizer_features_enable='11.2.0.1';

[cut]

Plan hash value: 1984967365

-----------------------------------------------------------
| Id | Operation | Name | Rows | Bytes |TempSpc
-----------------------------------------------------------
| 0  | SELECT STATEMENT | | 2658 | 384K|
|* 1 | HASH JOIN ANTI NA | | 2658 | 384K| 39M
| 2  | TABLE ACCESS FULL| CUSTOMER | 265K| 36M|
| 3 | TABLE ACCESS FULL| ORDERS | 2658K| 12M|
-----------------------------------------------------------

Predicate Information (identified by operation id):
-----------------------------------------------


1 - access("ID"="CUSTOMER_ID")
```

```
-------------------------
| Cost (%CPU)| Time |
-------------------------
| 16737 (3)| 00:01:55 |
| 16737 (3)| 00:01:55 |
| 1380 (2)| 00:00:10 |
| 11125 (2)| 00:01:17 |
-------------------------
```

Plan Management

Introduction: SQL Plan Managment

- "...to guarantee any plan changes that do occur lead to better performance..."
- Plans change
  - Updated stats
  - Parameter changes
  - Database upgrade
  - Data changes
- Goals
  - avoid performance regression
  - enable gains when possible

RMAN issue

SPM Basics

- Statement plan(s) managed
- SYSAUX holds repository of plans
  - Plan baseline - set of accepted plans for a statement
  - Non-Accepted plans
    - Need to evolve into accepted plans, if better
    - Privilege ADMINISTER SQL MANAGEMENT OBJECT

How is SPM Used?

- When statement runs
  - CBO does normal hard parse (if not in LC)
  - Looks for match in plan baseline
  - Match
    - Uses matching plan
  - No Match
    - Uses lowest cost plan in baseline
    - Adds new plan as non-accepted
    - Need to "evolve" the plan
      - Manually
      - Automatic
        - Weekly Tuning Advisor task
        - Consider "fixed" baseline

Lastly....

Questions....

One More Thing

All You Need is Love

John Lennon 1940 - 1980

Credits

- Jonathan Lewis
  - http://jonathanlewis.wordpress.com/
- Christian Antognini
  - http://antognini.ch
- Optimizer Developer Team
  - Allison W Lee and Mohamed Zait
  - http://blogs.oracle.com/optimizer/
- Alex Fatkulin, Pythian
  - http://afatkulin.blogspot.com/
- Charles Hooper
    - http://hoopercharles.wordpress.com/
- Tanel Poder
  - http://blog.tanelpoder.com/
- Arup Nanda
  - http://arup.blogspot.com/
- Richard Foote
  - http://richardfoote.wordpress.com
- Dion Cho
  - http://dioncho.wordpress.com/

Thanks Again for Attending

Dave Anderson
SkillBuilders.com/SQLTuning (slides posted here)
dave@SkillBuilders.com
1.401.783.6172