

SQL Tuning

Ohio Oracle User's Group
October 2002

Kris T. Mason

- Information Control Corp (ICC) Consultant
 - 10+ years with Oracle
 - Oracle 7x, 8x, 9i
 - 70% Application – 30% Production
- CSCC Adjunct Faculty
 - Oracle Programming
 - Computer Information Systems Concepts

Introduction

- Overview of the SQL Tuning process
- Database Internals
- Schema Objects involved with queries
- SQL Optimizer and Execution Plans
- Application construction

Why Tune?



Philosophy

- Be a Good Neighbor
 - Consume only those resources you need
 - Release resources you are not using
- The more you know ...

Reality Check

- Applications can not treat the database as a 'black-box' and expect optimal performance. They must contribute to efficiency.
- Overall efficiency can be gained by tuning small, frequently executing queries as well as large infrequent ones
- Tuning can't overcome bad design
- Tune SQL before tuning the DB Instance
- Politics, Balance, Judgement

Preparing to tune

- Programming
 - Languages
 - Application segmentation
 - Bind Variables
- Server Configuration
 - Hardware (CPUs, Memory, Disks, Network)
 - Non-Database applications
- Database Configuration
 - Init.ora
 - Gather statistics –
select value, name from v\$sysstat order by name;
 - Analyze Tables & Indexes (DBMS_STATS package)
 - DB Links and Replication
- Know where you are, want to be and how to get there
 - Current performance
 - What is 'acceptable' performance
 - How long it will take

Minimizing

- Logical Reads
- Disk Reads
- Temp Space
- Memory
- Sorts
- Parsing
- Network Traffic
- Application Round Trips
- Response Time (Perceived & Real)

Database Internals

- System Global Area (SGA)
 - Database Block Buffers
 - Block Size usually 8k - 16k
 - Holds portions of Tables and Indexes
 - Minimizes Expensive Disk Reads
 - V\$DB_CACHE_ADVICE
 - Buffer Cache Hit Ratio
 - Library Cache – Dictionary and recent SQL
 - Two SQL statements are identical only if they match byte for byte
 - Programmers and Dynamic Query Rewrite (Cursor Sharing) can force queries to be identical via bind variables
 - Identical Queries accumulate statistics in V\$SQLAREA providing better visibility to statement use

Oracle Features

- Dynamic Query Rewrite (Cursor Sharing)
- Parallel Query
- Optimizer Modes
- Hints
- Alter Session statement
- Table & Index Partitioning
- Statspack

Dynamic Query Rewrite (Cursor Sharing)

- Init.ora
 - QUERY_REWRITE_ENABLED=TRUE
 - QUERY_REWRITE_INTEGRITY=TRUSTED
- Turns literal strings and values into bind variables
 - Bypasses the programmer
- Different queries become identical
 - WHERE STATUS = 1 AND NAME = 'SMITH'
 - WHERE STATUS = 3 AND NAME = 'JONES'
 - WHERE STATUS = :b1 AND NAME = :b2
- Statistics are accumulated under the single query

Parallel Query

- Allows multiple CPUs to work parts of the same query (divide and conquer)
- My experience:
 - Best when 6+ CPUs
 - Can be over done, causing different concurrently executing queries to compete for resources
- Configure PQ Servers $(\#CPUs - 1) * 2$

Optimizer

- Rule / Choose / Cost
- Generates alternative execution plans
- Plans can be seen and understood
 - SET AUTOTRACE ON
 - SET AUTOTRACE TRACEONLY

Primary Schema Objects

- Tables
 - Full Table Scan
 - Random Data Block Read
 - ROWID – Direct Access
 - Index
 - Index Organized Table
- Indexes
 - B*Tree / Bitmap / Index-Only Table / Function Based
 - Ideal when highly selective
 - Index Only Processing avoids Data Block Read
 - Entirely NULL values are not stored in the index
 - ALTER INDEX REBUILD
- Tables & Indexes require Data block reads if not in memory already

Indexes

Selectivity = (# of Distinct Values / # of Rows) * 100

Selectivity = 0% when all the values have the same value

Selectivity = 100% when all the values have different values (PKs)

Indexes lose effectiveness when Selectivity falls below 80%

Indexes gain effectiveness when Selectivity approaches 100%

Index use can be invalidated (Index Invalidation) when the column reference in the WHERE is involved with a function

```
SELECT ... WHERE ID = 5 /* Uses an index */
```

```
SELECT ... WHERE ID + 0 = 5 /* Does not use an index */
```

```
SELECT ... WHERE NAME BETWEEN 'SMITH' AND 'THOMAS' /* Uses an index */
```

```
SELECT ... WHERE UPPER(NAME) BETWEEN 'SMITH' AND 'THOMAS' /* Does not use an index */
```

Secondary Schema Objects

- Views
 - Stored SELECT statement
 - Some Hints do not propagate inside a complex view
- Triggers
 - PL/SQL code
 - Registered against a specific table
 - Fired upon specific DML events

Tuning Process

- Find SQL
 - SQL Analyze & Top SQL
 - Top Sessions
 - SQL*Plus & SQL Worksheet
 - V\$SQLAREA
- Look For
 - High Disk Reads vs. Low Rows Returned
 - High Executions
 - DISTINCT
- Identify Problem
 - Selectivity
 - Access Path
 - Join Method and Order
- Consider Possible Alternatives
 - Hints
 - Indexes
- Document, Measure & Test Alternatives
 - Show your work
- Deploy Solution
 - Get credit for your work

Finding SQL to tune

- **V\$SQLAREA**

```
set linesize 1000
set pagesize 1000
column disk_reads format 999,999,999,999
column executions format 999,999,999,999
column rows_processed format 999,999,999,999
column sql_text format a800 word_wrapped
select executions, disk_reads, rows_processed, sorts, sql_text from v$sqlarea
where disk_reads > 3000 order by disk_reads DESC;
```

Look for high resource consumers

```
DISK_READS
EXECUTIONS
SORTS
```

Limited to 1000 bytes of SQL Statement including white space. Be sure you have the whole statement!
The entire statement exists in v\$sqltext but it is broken into pieces

Helper PL/SQL Functions

```
CREATE OR REPLACE FUNCTION SQUISH
  (P in varchar2)
  return varchar2 as
  l number;
  V varchar2(32000);
Begin
/* Change special chars to blanks */
V :=
  trim(rtrim(translate(P,chr(9)||chr(10)||chr(13),
    ' '))); /* One blank between quotes */
Loop
  l := instr (V, ' '); /* Two blanks between quotes
  */
  Exit when l = 0;
/* V := replace two blanks with one */
  V := replace (V, ' ', ' ');
End loop;
Return V;
End Squish;
```

```
CREATE OR REPLACE FUNCTION
  CAT_SQL_TEXT (P in varchar2) return
  varchar2 as
  V varchar2(32000) := "";
Begin
for C in (select sql_text from v$sqltext where
  address = P order by piece)
Loop
  V := V || C.sql_text;
End loop;
Return V;
End cat_sql_text;
```

```
Grant select on v$sqltext to [function owner];
```

Finding SQL To Tune with Helper Functions

- **V\$SQLTEXT**

```
set linesize 1000
```

```
set pagesize 1000
```

```
column disk_reads format 999,999,999,999
```

```
column executions format 999,999,999,999
```

```
column rows_processed format 999,999,999,999
```

```
column sql_text format a800 word_wrapped
```

```
select address, hash_value, executions, disk_reads, rows_processed, sorts, squish(cat_sql_text(address))  
       sql_text from v$sqlarea where disk_reads > 3000 order by disk_reads DESC;
```

Finding SQL to tune

- **DISTINCT**

EXECUTIONS	DISK_READS	ROWS_PROCESSED	SORTS	SQL_TEXT
81	3,499	5,103	81	SELECT DISTINCT TABLE_B.NAME FROM TABLE_A, TABLE_B WHERE (TABLE_A.ID = TABLE_B.ID)

The Intent of this query is to show the names in Table_B that have any Matching rows in Table_A

Original

- Document the query as originally written

```
SELECT DISTINCT TABLE_B.NAME  
FROM TABLE_A, TABLE_B  
WHERE ( TABLE_A.ID = TABLE_B.ID );
```

Execution Plan

```
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=921 Card=306 Bytes=5814)  
1  0  SORT (UNIQUE) (Cost=921 Card=306 Bytes=5814)  
2  1  HASH JOIN (Cost=33 Card=73605 Bytes=1398495)  
3  2  TABLE ACCESS (FULL) OF 'TABLE_B' (Cost=1 Card=307 Bytes=4298)  
4  2  INDEX (FAST FULL SCAN) OF 'FK_TABLE_A_ID' (NON-UNIQUE) (Cost=14 Card=73605  
Bytes=368025)
```

Original Statistics

Statistics

43	recursive calls
8	db block gets
247	consistent gets
0	physical reads
0	redo size
1368	bytes sent via SQL*Net to client
591	bytes received via SQL*Net from client
6	SQL*Net roundtrips to/from client
5	sorts (memory)
0	sorts (disk)
63	rows processed

Alternative

- Document each version of the query as rewritten

```
SELECT TABLE_B.NAME  
FROM TABLE_B  
WHERE EXISTS (  
SELECT 1 FROM TABLE_A WHERE TABLE_A.ID = TABLE_B.ID );
```

Execution Plan

```
-----  
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=16 Bytes=224)  
1  0  FILTER  
2  1  TABLE ACCESS (FULL) OF 'TABLE_B' (Cost=1 Card=16 Bytes=224)  
3  1  INDEX (RANGE SCAN) OF 'FK_TABLE_A_ID' (NON-UNIQUE) (Cost=4 Card=1169  
    Bytes=5845)
```

Alternative's Statistics

Statistics

```
0 recursive calls
4 db block gets
638 consistent gets
0 physical reads
0 redo size
1409 bytes sent via SQL*Net to client
591 bytes received via SQL*Net from client
6 SQL*Net roundtrips to/from client
2 sorts (memory)
0 sorts (disk)
63 rows processed
```

Documentation

- Original version
- Reformed version
 - Rewrite by hand in a form comfortable to you
 - Make sure it is equivalent to the original
 - Match query's joins to data model
- Each alternative
 - A record of your work
- Recommended solution

Testing

- Full volume production testing
- Test or Personal copy of Database
- DBMS_STATS
 - Import Statistics from Production

What Can Be Done?

- Analyze Tables & Indexes
 - Histograms
- See Oracle's Execution Plan
 - SET AUTOTRACE [ON, TRACEONLY]
 - PLAN table
 - Query Costs (Oradollars)
- Hints – You know more than the Optimizer!
 - Optimizer Goal
 - FIRST_ROWS
 - ALL_ROWS
 - Access Path
 - FULL
 - INDEX
 - Query Transformation
 - ORDERED
 - STAR
 - Join Operation
 - USE_NL
 - USE_MERGE
 - USE_HASH
 - DRIVING_SITE
 - LEADING

Execution Plans

- A roadmap for executing the query
 - Top to bottom
 - Inside to outside
- Execution Techniques
 - Full Table Scan – Not necessarily a bad thing
 - Unique Index Scan
 - Index Range Scan
 - Concatenation
 - Sort
 - Aggregation
 - Order By
 - Group By
 - Nested Loops
 - Merge Join
 - Hash Join
 - IN-List Iterator

Hints

- `SELECT /*+ hint goes here */ ...`
- Features:
 - Syntax errors are not reported and not used
 - Specify the table alias not the table name
 - Put preceding and trailing blank
 - You can use multiple hints
 - Hints don't have to be used by the optimizer

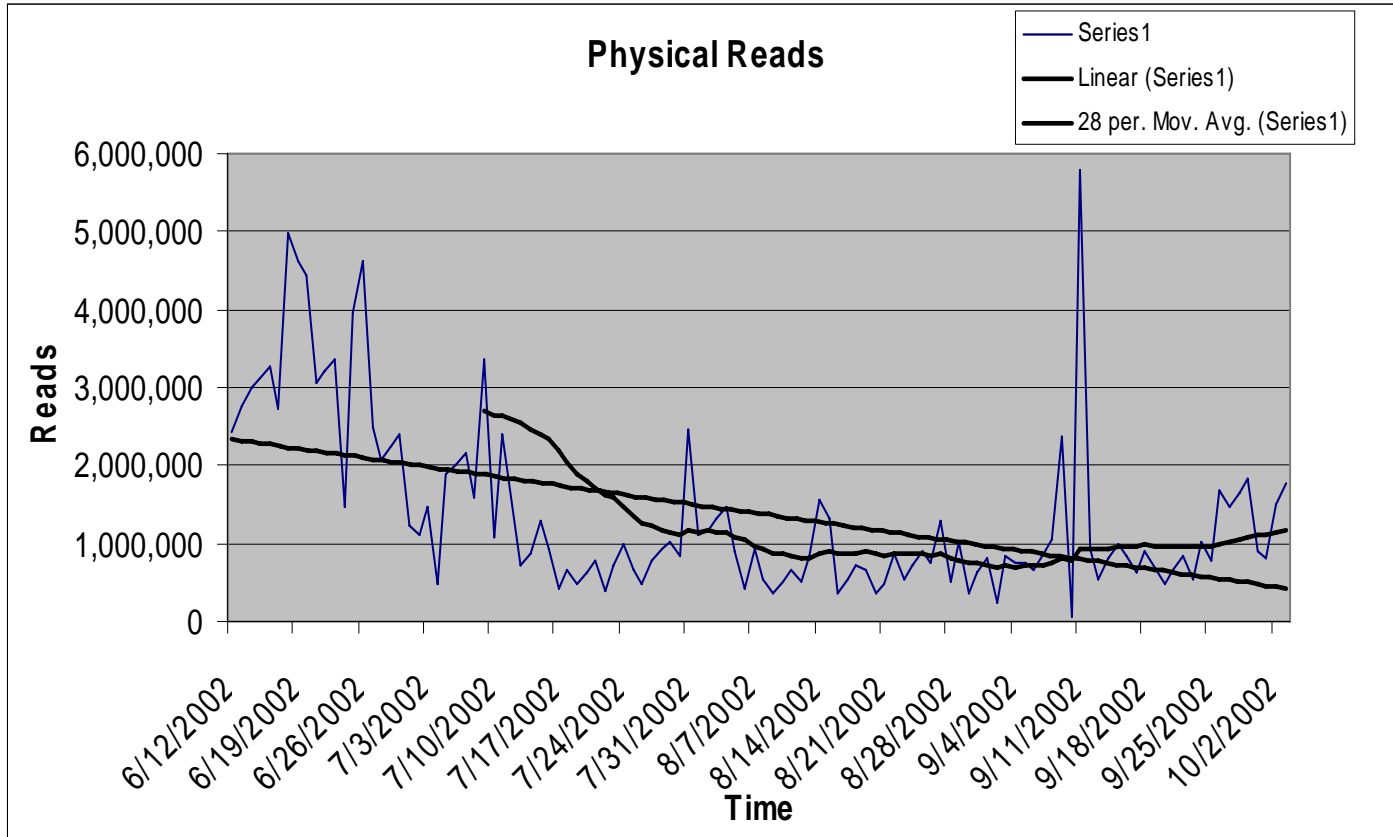
Other Techniques to Optimize Performance

- ALTER SESSION
 - SET SORT_AREA_SIZE=1000000
 - SET ROLLBACK_SEGMENT=RBS_BIG
- Minimized Chained Rows
- How full are the data blocks?
 - Random Deletes reduce efficiency of table scans
- Use Plan Stability to insure optimizer does not change the plan over time

Distributed Queries

- Queries across multiple instances
- Via a DB_Link
- Create a view of the remote tables
- Minimize 'round-trips' to the remote DB
- Pull data to the local DB via a snapshot
- Wire the local & remote servers together

Effects of tuning



Applications

- Start with optimized queries so when code is cloned it has a better chance of working well
- Review 3rd party produced code such as PowerBuilder
- Write code for the proper platform (application segmentation)
- PL/SQL is not 'cool' but it is fast and effective
- Use Bind Variables whenever possible
- Avoid Dynamic SQL if possible

References

- Oracle's Technet
 - Database Performance Tuning Guide & Reference
http://otn.oracle.com/docs/products/oracle9i/doc_library/release3/toc.htm



www.iccohoio.com

- KTMason@iccohoio.com

Summary

SQL Tuning is not necessarily hard,
it just takes some practice
and the willingness to try