Data Warehouse Tuning

Without SQL Modification

Agenda

- About Me
- Tuning Objectives
- Data Access Profile
- Data Access Analysis
- Performance Baseline
- Potential Model Changes
- Model Change Testing
- Testing Results

Brandon Hawkes

- Works at: USANA Health Sciences
- Data Warehouse Engineer
- Contributing Author:
  - **Oracle Database 12c PL/SQL Advanced Programming Techniques**
    - **Chapter 10: Optimizing Using PLSQL**
- Co-Author
  - **Accelerate Your Databases with Dell DISOD**
- Contact Me At:
  - Email: brandonhawkes41@gmail.com

Tuning Objectives

- What needs tuning?
  - Single Report?
  - ETL Job?
  - ★ Single Table?
  - ★ Star Schema?
  - ★ Entire Data Warehouse?
- How is the data being used?
- Am I following best practices?
- What are my tuning Options?
- What resources do I have available?

Data Access Profile

- What are the Select Statements being used to access the Data?
- What are the Update Statements being used to Update Data?
- What are the Insert Statements being used to Insert Data?

- How do we gather the Statements?
  - Standard Auditing
  - Fine Grained Auditing for more targeted application

Data Access Profile

- AUDIT (SELECT | UPDATE | INSERT | DELETE) ON <SCHEMA>.<TABLE_NAME> BY ACCESS;
  - Any user that selects from the table places an entry into the SYS.AUD$ table for each statement.
- We audit every statement by every user on the object to be able to understand the workload against the object.
- Can set up scheduled job to import the sum of statements into aggregate table, then delete the rows out of the SYS.AUD$ so as to not take up too much disk space

Data Access Analysis

1. Determine the number of times each statement was issued against the table.
   - SQLTEXT stored as CLOB in SYS.AUD$
   - Need to use hash of CLOB to find distinct SQL Statements
     - (DBMS_CRYPTO.HASH)
2. Gather the SQL statement for analysis of access of data.
3. Create table with SQL Statements for easier access

```
CREATE TABLE <TABLE_NAME$SQL> AS
WITH alpha AS (
  SQLTEXT
, DBMS_CRYPTO.HASH(sqltext, 2) sql_hash
, COUNT(*) OVER (PARTITION BY DBMS_CRYPTO.HASH(sqltext, 2)) dup_sql
, ROW_NUMBER() OVER (PARTITION BY DBMS_CRYPTO.HASH(sqltext, 2) ORDER BY 1) row_num
FROM SYS.AUD$ A
WHERE A.obj$creator = '<SCHEMA_NAME>'
AND A.obj$name = '<TABLE_NAME>'
AND A.action# = 3  /* Use 2 for Insert, 3 for Select, 6 for Update and 7 for Delete */  )
, beta AS (
SELECT
  sql_hash
, ROW_NUMBER() OVER (PARTITION BY DBMS_CRYPTO.HASH(sqltext, 2) ORDER BY 1) row_num
, dup_sql
FROM alpha  )
, charlie AS (
SELECT
  A.sql_hash
, A.dup_sql
, sqltext
FROM beta b
JOIN alpha A
ON A.sql_hash = b.sql_hash
AND b.row_num = A.row_num
AND b.row_num = 1 )
SELECT
  ROWNUM row_num
, c.*
FROM charlie c;
```

Sample Output

ROW_NUM	SQL_HASH	DUP_SQL	SQLTEXT
181	068338848CA5C02565DF71A9947E2B03	3	(HUGECLOB)
182	0A9CAE675F48525154F058B6E6BA25CE	1	(HUGECLOB)
183	2B7590A68B7842DDA2C7350DA8B946E7	6	(HUGECLOB)
184	3212352FC38999A3C3C237344882B421	1	(HUGECLOB)
185	42BA9C4AB0615D4C61DE751B83860CD8	1	(HUGECLOB)
186	5222CFB9054CBB58BD7B7D615A2AB830	4	(HUGECLOB)
187	6EDAB92666C37933422D0342D2AFD2A4	1	(HUGECLOB)
188	004FD8E06D21FD2F793D68D437116F18	3	(HUGECLOB)
189	075CF2A26836CAD0CD94650313A65C41	1	(HUGECLOB)
190	2ED0F98E67252B55EEC27300403597DD	1	(HUGECLOB)
191	3361126DD7388ABD066A3B16E5A2D845	1	(HUGECLOB)
192	390B5BE0CE8D41CA0AD80A4B814CE566	2	(HUGECLOB)
193	41E3E9177518414B9C1FCA95B62AD5CF	3	(HUGECLOB)
194	969790741F25A0DEC2418CB125A364DB	31	(HUGECLOB)
195	D23A51018A0EBF557C0EE5E5A3B3E917	1	(HUGECLOB)
196	F33CB92DD81C3D976E47C4C66D555DFD	3	(HUGECLOB)
197	2991F637043EF42388D87206CFAD1D85	3	(HUGECLOB)

Data Access Analysis

- Most concerned about the most used filtering and joining columns
  - Columns are candidates for Indexing, Partitioning and Pre-Sorting on table
- Need to determine how the columns of the tables are filtered.
  - Otherwise the query will perform a full table scan.
- Can find this from Explain Plans
- Use PL/SQL to automate from gathered SQL statements.
  - Also good way to validate SQL statements run correctly

```
DECLARE
lv_sqlhash varchar2(33);
BEGIN
  FOR i IN (SELECT * FROM <TABLE_NAME$SQL>) LOOP
    BEGIN
      lv_sqlhash := i.sql_hash;
      execute immediate ('explain plan SET STATEMENT_ID = ''' || substr(i.sql_hash,1,30) || ''' for ' || i.sqltext);
    EXCEPTION WHEN others THEN
      dbms_output.put_line(lv_sqlhash);
      CONTINUE;
    END;
  END LOOP;
END;
/
```

Data Access Analysis

- Parse through FILTER_PREDICATES and ACCESS_PREDICATES columns in PLAN_TABLE to find columns that are used to filter the table that you are interested in tuning.
- Count the number of times that a column is filtered in each SQL statement
- Multiply the number of times that a column is used to filter by the number of times that the SQL statement has been used
- Gather Explain Plan Cost as another option.

Sample Output

COLUMN_NAME	SQL_ACCESSES_SUM
ROW_WID	17881
FULL_NAME	15461
LAST_NAME	4854
FST_NAME	3123
CITY	2262
COUNTY	1924
REGION_CODE	1890
ZIPCODE	1720
EMP_HIRE_DT	1290
SUPERVISOR_NAME	1118

Performance Baseline

- Need to baseline current resource usage
- Use STAT_SNAP Package from **Oracle Database 12c PL/SQL Advanced Programming Techniques : Chapter 10**
- Used to gather resource usage from **V$MYSTAT** view
- Can toggle statistics gathered from RTS_TOGGLE table
- Wrap in PL/SQL for automation
- Determine threshold level of queries to test
  - i.e. only gather for queries that have run more than 20 times during the gathering period
  - i.e. only for the top 25% of queries by Explain Plan cost

Performance Baseline Script

```
DECLARE
  PROCEDURE P(x IN varchar) IS BEGIN dbms_output.put_line(x); END;
BEGIN
  P('declare
    lv_int int;');
  P('begin');
  P('stat_snap.start_snap(''Baseline Query'');');
FOR i IN (SELECT * FROM <TABLE_NAME$SQL> WHERE dup_sql > 20) LOOP
  P('stat_snap.start_snap(''Baseline Query #' || i.row_num || ' - Hash: ' || i.sql_hash || ' - SQL Cnt: ' || i.dup_sql ||''');');
  P('execute immediate (''select count(*) from (' || REPLACE(i.sqltext, chr(39), chr(39)||chr(39)) || ')'') into lv_int;');
  P('stat_snap.end_snap(''Baseline Query #' || i.row_num || ' - Hash: ' || i.sql_hash || ' - SQL Cnt: ' || i.dup_sql ||''');');
END LOOP;
  P('stat_snap.end_snap(''Baseline Query'');');
  P('end;
    /');
END;
/
```

Sample Output

SNAP_NAME	STAT_NAME	RESOURCE_AMT_USED	ELAPSED_TIME
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell CUs sent compressed	0	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell CUs sent uncompressed	378195	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell IO uncompressed bytes	51476291362	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell blocks processed by cache layer	1016030	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell blocks processed by data layer	977172	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell blocks processed by index layer	0	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell blocks processed by txn layer	1016030	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell flash cache read hits	17572	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell index scans	0	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell physical IO bytes eligible for predicate offload	16009641984	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell physical IO bytes saved by storage index	0	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell physical IO bytes sent directly to DB node to balance CPU	0	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell physical IO interconnect bytes	12751294352	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell physical IO interconnect bytes returned by smart scan	4528082832	+00 00:00:24.784026
Baseline QUERY #38 - HASH: 14D2D3E0358122DC62B541F95865EA89 - Num Queries: 21	cell smart IO session cache hits	611	+00 00:00:24.784026

Potential Model Changes

- Ways to modify Table Model Structure:
  - Indexing
  - Partitioning
  - Degree of Parallel
  - PCTFREE *
  - Pre-Sorting Data *
  - Compression
- Need to work within licensing
- Effectiveness depends upon the structure of the hardware that your database is running on

Testing Changes

1. Add SQLTEXT_MOD column to <TABLE_NAME$SQL> table as copy of SQL_TEXT column
2. Use Create Table As Select to create copy of table to be tested with modifications
3. Update SQLTEXT_MOD and replace the name of the original table with the new table.
4. Modify Performance Baseline Script to use SQLTEXT_MOD column as the source SQL
5. Repeat steps 2, 3 and 4 for as many modifications as need testing

```
ALTER TABLE <TABLE_NAME$SQL> ADD (sqltext_mod clob);

CREATE TABLE <TABLE_NAME_2> AS
SELECT * FROM <TABLE_NAME>
<PARTITION OPTIONS>
<PARALLEL OPTIONS>
<COMPRESSION OPTIONS>
<PCTFREE OPTIONS>
<ORDER OPTIONS>;

UPDATE <TABLE_NAME$SQL> SET SQLTEXT_MOD = REPLACE(SQLTEXT, '<TABLE_NAME>','<TABLE_NAME_2>');

DECLARE
  PROCEDURE P(x IN varchar) IS BEGIN dbms_output.put_line(x); END;
BEGIN
  P('declare
    lv_int int;');
  P('begin');
  P('stat_snap.start_snap(''Test #1'');');
FOR i IN (SELECT * FROM <TABLE_NAME$SQL> WHERE dup_sql > 20) LOOP
  P('stat_snap.start_snap(''Test Query #' || i.row_num || ' - Hash: ' || i.sql_hash || ' - SQL Cnt: ' || i.dup_sql ||''');');
  P('execute immediate (''select count(*) from (' || REPLACE(i.sqltext_mod, chr(39), chr(39)||chr(39)) || ')'') into lv_int;');
  P('stat_snap.end_snap(''Test Query #' || i.row_num || ' - Hash: ' || i.sql_hash || ' - SQL Cnt: ' || i.dup_sql ||''');');
END LOOP;
  P('stat_snap.end_snap(''Test #1'');');
  P('end;
    /');
END;
/
```

Testing Results

- Can see the difference in individual query resource usage or group query usage.
- Query time is also very important for the end users
- Use STAT_COLLECTION_VW to compare baseline and different tests on resource usage
- Compare results to determine what the optimum table structure is for your bottlenecks
  - i.e. IO, Network, CPU
  - Find information about the different statistics from: http://docs.oracle.com/cd/E11882_01/server.112/e40402/stats002.htm#i375475

QUESTIONS?