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
<table>
<thead>
<tr>
<th>ROW_NUM</th>
<th>SQL_HASH</th>
<th>DUP_SQL</th>
<th>SQLTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>181</td>
<td>068338848CA5C02565DF71A9947E2803</td>
<td>3</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>182</td>
<td>0A9CAE675F48525154F05886E6BA25CE</td>
<td>1</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>183</td>
<td>2B7590A687B842DDA2C7350D48B946E7</td>
<td>6</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>184</td>
<td>3212352FC38999A3C3C2373448828421</td>
<td>1</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>185</td>
<td>42BA9C4A80615D4C61DE751B83860CD8</td>
<td>1</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>186</td>
<td>5222CFB9054CB858BD7BD615A2BAB30</td>
<td>4</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>187</td>
<td>6EDAB92666C37933422D0342D2AFD2A4</td>
<td>1</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>188</td>
<td>00FD806ED21FD2F793D68D437116F18</td>
<td>3</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>189</td>
<td>075CF2A26836CAD0CD94650313A55C41</td>
<td>1</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>190</td>
<td>2ED0F9867252B55EEC27300403597DD</td>
<td>1</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>191</td>
<td>361126DD7388ABD066A3816E5A2D845</td>
<td>1</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>192</td>
<td>390B5BE0CE8D41CA0ADB0A48814CE566</td>
<td>2</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>193</td>
<td>413E917751B4189C1FCA95B62AD5SCF</td>
<td>3</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>194</td>
<td>96979074F25ADEC2418CB125A364DB</td>
<td>31</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>195</td>
<td>D23A510180EF557C0E5E5A3B3E917</td>
<td>1</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>196</td>
<td>F33CB92DD81C3D976E47C4C66D555DFD</td>
<td>3</td>
<td>(HUGELOB)</td>
</tr>
<tr>
<td>197</td>
<td>2991F637043EF42388D87206CFAD1D85</td>
<td>3</td>
<td>(HUGELOB)</td>
</tr>
</tbody>
</table>
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(35);
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.
<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>SQL_ACCESSSES_SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW_WID</td>
<td>17881</td>
</tr>
<tr>
<td>FULL_NAME</td>
<td>15461</td>
</tr>
<tr>
<td>LAST_NAME</td>
<td>4854</td>
</tr>
<tr>
<td>FST_NAME</td>
<td>3123</td>
</tr>
<tr>
<td>CITY</td>
<td>2262</td>
</tr>
<tr>
<td>COUNTY</td>
<td>1924</td>
</tr>
<tr>
<td>REGION_CODE</td>
<td>1890</td>
</tr>
<tr>
<td>ZIPCODE</td>
<td>1720</td>
</tr>
<tr>
<td>EMP_HIRE_DT</td>
<td>1290</td>
</tr>
<tr>
<td>SUPERVISOR_NAME_NAME</td>
<td>1118</td>
</tr>
</tbody>
</table>
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$MYSTAT view
- Can toggle statistics gathered from RTS_TOGGLE table
- Wrap in PL/SQL for automation
- Determine threshold level of queries to test
  - i.e. only gather for queries that have run more than 20 times during the gathering period
  - i.e. only for the top 25% of queries by Explain Plan cost
DECLARE
PROCEDURE F(x IN varchar) IS BEGIN dbms_output.put_line(x); END;
BEGIN
  F('declare
     lv_int int;');
  F('begin');
  F('stat_snap.start_snap( ''Baseline Query'' );');
  FOR i IN (SELECT * FROM <TABLE_NAME$SQL> WHERE dup_sql > 20) LOOP
    F('stat_snap.start_snap( ''Baseline Query #'' || i.row_num || ' - Hash: '' || i.sql_hash || ' - SQL Cnt: ' || i.dup_sql || ''');');
    F('execute immediate (''select count(*) from (' || REPLACE(l.sqltext, chr(39), chr(39)||chr(39)) || '')'') into lv_int;');
    F('stat_snap.end_snap( ''Baseline Query #'' || i.row_num || ' - Hash: ' || i.sql_hash || ' - SQL Cnt: ' || i.dup_sql || ''');');
  END LOOP;
  F('stat_snap.end_snap( ''Baseline Query'' );');
  F('end;');
END;
<table>
<thead>
<tr>
<th>SNAP_NAME</th>
<th>STAT_NAME</th>
<th>RESOURCE_AMT_USED</th>
<th>ELAPSED_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell CUs sent compressed</td>
<td>0</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell CUs sent uncompressed</td>
<td>387195</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell IO uncompressed bytes</td>
<td>51476291362</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell blocks processed by cache layer</td>
<td>1016030</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell blocks processed by data layer</td>
<td>977172</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell blocks processed by index layer</td>
<td>0</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell blocks processed by txn layer</td>
<td>1016030</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell flash cache read hits</td>
<td>17572</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell index scans</td>
<td>0</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell physical IO bytes eligible for predicate offload</td>
<td>16009641984</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell physical IO bytes saved by storage index</td>
<td>0</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell physical IO bytes sent directly to DB node to balance CPU</td>
<td>0</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell physical IO interconnect bytes</td>
<td>12751294352</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell physical IO interconnect bytes returned by smart scan</td>
<td>4528082832</td>
<td>+00 00:00:24.784026</td>
</tr>
<tr>
<td>Baseline QUERY #38 - HASH: 14D2D3E035B122DC62B541F95865E8A9 - Num Queries: 21</td>
<td>cell smart IO session cache hits</td>
<td>611</td>
<td>+00 00:00:24.784026</td>
</tr>
</tbody>
</table>
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 = 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?