Scaling To Infinity: Partitioning Data Warehouses on Oracle Database

Thursday 18-October 2012
Tim Gorman
www.EvDBT.com
Speaker Qualifications

- Co-author...
  2. “Essential Oracle8i Data Warehousing”, 2000 John Wiley & Sons
  3. “Oracle Insights: Tales of the Oak Table”, 2004 Apress
  4. “Basic Oracle SQL” 2009 Apress
  5. “Expert Oracle Practices: Database Administration with the Oak Table”, 2010 Apress

- 28 years in IT...
  - “C” programmer, sys admin, network admin (1984-1990)
  - Consultant and technical consulting manager at Oracle (1990-1998)
  - Independent consultant (http://www.EvDBT.com) since 1998
  - Rocky Mountain Oracle Users Group (http://www.RMOUG.org) since 1992
  - Oak Table network (http://www.OakTable.net) since 2002
  - Oracle ACE since 2007, Oracle ACE Director since 2012
Agenda

• The virtuous cycle and the death spiral
• Basic 5-step EXCHANGE PARTITION load technique
• 7-step EXCHANGE PARTITION technique for “dribble effect”
• Performing MERGE/up-sert logic using EXCHANGE PARTITION
Data warehousing reality

• We have to recognize how features for large data volumes and optimal queries work together
  – Partitioning
  – Direct-path loading
  – Compression
  – Star transformation
  – Bitmap indexes
  – Bitmap-join indexes
  – READ ONLY tablespaces
  – Information lifecycle management

• Because it really isn’t documented anywhere
The Virtuous Cycle

- Non-volatile time-variant data implies...
  - Data warehouses are INSERT only
- Insert-only data warehouses implies...
  - Tables and indexes range-partitioned by a DATE column
- Tables range-partitioned by DATE enables...
  - Data loading using EXCHANGE PARTITION load technique
  - Partitions organized into time-variant tablespaces
  - Incremental statistics gathering and summarization
- Data loading using EXCHANGE PARTITION enables...
  - Direct-path (a.k.a. append) inserts
  - Data purging using DROP/TRUNCATE PARTITION instead of DELETE
  - Bitmap indexes and bitmap-join indexes
  - Elimination of ETL “load window” and 24x7 availability for queries
The Virtuous Cycle

• Direct-path (a.k.a. *append*) inserts *enable*...
  – Load more data, faster, more efficiently
  – Optional NOLOGGING on inserts
  – Basic table compression (9i) or HCC (11gR2) for Oracle storage
  – Eliminates contention in Oracle Buffer Cache during data loading

• Optional NOLOGGING inserts *enable*...
  – Option to generate less redo during data loads
  – Optimization of backups

• Table compression enables...
  – Less space consumed for tables and indexes
  – Fewer I/O operations during queries

• Partitions organized into time-variant tablespaces *enable*...
  – READ ONLY tablespaces for older, less-volatile data
The Virtuous Cycle

• READ ONLY tablespaces for older less-volatile data enables...
  – Tiered storage
  – Backup efficiencies
• Data purging using DROP/TRUNCATE PARTITION enables...
  – Faster more efficient data purging than using DELETE statements
• Bitmap indexes enable...
  – Star transformations
• Star transformations enable...
  – Optimal query-execution plan for dimensional data models
  – Bitmap-join indexes
• Bitmap-join indexes enable...
  – Further optimization of star transformations
The Death Spiral

- ETL using “conventional-path” INSERT, UPDATE, and DELETE operations
- Conventional-path operations work well in transaction environments
  - High-volume data loads in bulk are problematic
  - High parallelism causes contention in Shared Pool, Buffer Cache
    - Mixing of queries and loads simultaneously on table and indexes
    - Periodic rebuilds/reorgs of tables if deletions occur
    - Full redo and undo generation for all inserts, updates, and deletes
  - Bitmap indexes and bitmap-join indexes
    - Modifying bitmap indexes is slow, SLOW, SLOW
    - Unavoidable locking issues in during parallel operations
The Death Spiral

• ETL dominates the workload in the database
  – Queries will consist mainly of “dumps” or extracts to downstream systems
  – Query performance worsens as tables/indexes grow larger
  – Stats gathering takes longer, smaller samples worsen query performance
  – Contention between queries and ETL become evident
  – Uptime impacted as bitmap indexes must be dropped/rebuilt

• Backups consume more and more time and resources
  – Entire database must be backed up regularly
  – Data cannot be “right-sized” to storage options according to IOPS, so storage becomes non-uniform and patchwork, newer less-expensive storage is integrated amongst older high-quality storage, failure points proliferate
Basic 5-step technique

• The basic technique of bulk-loading new data into a temporary-user “scratch” table, which is then indexed, analyzed, and finally “published” using the EXCHANGE PARTITION operation
  – This should be the default load technique for all large tables in a data warehouse

• Assumptions for this example:
  – A “type 2” time-variant composite-partitioned fact table named TXN
    • Range partitioned on DATE column TXN_DATE
    • Hash sub-partitioned on NUMBER column ACCT_KEY
  – 25-Feb 2014 data to be loaded into “scratch” table named TXN_SCRATCH
  – Ultimately data to be published into partition P20140225 on TXN
Basic 5-step technique

1. Create Scratch Table
2. Bulk Loads
3. Table & Col Stats
4. Index Creates
5. Exchange Partition

Range-hash composite-partitioned TXN

Hash-partitioned TXN_SCRATCH

<table>
<thead>
<tr>
<th>Date</th>
<th>22-Feb 2014</th>
<th>23-Feb 2014</th>
<th>24-Feb 2014</th>
<th>(empty)</th>
<th>25-Feb 2014</th>
</tr>
</thead>
</table>
Basic 5-step technique

1. Create ScratchTable
2. Bulk Loads
3. Table & Col Stats
4. Index Creates
5. Exchange Partition
Basic 5-step technique

1. Create temporary table TXN_SCRATCH as a hash-partitioned table
2. Perform parallel, append load of data into TXN_SCRATCH
3. Gather CBO statistics on table TXN_SCRATCH
   • Only table and columns stats
4. Create indexes on TXN_SCRATCH matching local indexes on TXN
5. alter table TXN
   exchange partition P20140225 with table TXN_SCRATCH
   including indexes without validation update global indexes;
Basic 5-step technique

• It is a good idea to encapsulate this logic inside PL/SQL packaged- or stored-procedures:

```sql
SQL> exec exchpart.prepare('TXN','TXN_SCRATCH','25-FEB-2014');
SQL> alter session enable parallel dml;
SQL> insert /*+ append parallel(n, 16) */ into txn_scratch n
   3 select /*+ full(x) parallel(x, 16) */ *
   4 from   ext_stage x
   5 where x.load_date >= '25-FEB-2014'
   6 and    x.load_date < '26-FEB-2014';
SQL> commit;
SQL> exec exchpart.finish('TXN','TXN_SCRATCH');
```

• DDL for EXCHPART package posted at [http://www.EvDBT.com/tools.htm#exchpart](http://www.EvDBT.com/tools.htm#exchpart)
The “dribble effect”

• In real-life, data loading is often much *messier*...
  – Due to range partition key column not matching load cycles...

**Example:** data to be loaded on 25-Feb is ~1,000,000 rows:

• 950,000 rows for 25-Feb
• 45,000 rows for 24-Feb
• 4,000 rows for 23-Feb
• 700 rows for 22-Feb
• 200 rows for 21-Feb
• 90 rows for 20-Feb
• ...and a dozen rows left over from 07-Jan...
The “dribble effect”

Use EXCHANGE PARTITION technique when >= N rows; otherwise, conventional INSERT.

for d in (select trunc(txn_dt) dt, count(*) cnt from EXT_STAGE group by trunc(txn_dt)) loop
    if d.cnt >= 100 then
        exchpart.prepare('TXN','TXN_P'||to_char(d.dt,'YYYYMMDD'), d.dt);
        insert /*+ append parallel(n,16) */ into TXN_P20140224 n
        select /*+ parallel(x,16) */ * from EXT_STAGE x
        where x.txn_dt >= d.dt and x.txn_dt < d.dt + 1;
        exchpart.finish('TXN','TXN_P'||to_char(d.dt,'YYYYMMDD'));
        exchpart.drop_indexes('TXN_P'||to_char(d.dt,'YYYYMMDD'));
        insert /*+ append parallel(n,16) */ into TXN_P20140224 n
        select /*+ parallel(x,16) */ * from EXT_STAGE x
        where x.txn_dt >= d.dt and x.txn_dt < d.dt + 1;
    else
        insert into TXN
        select * from ext_stage
        where txn_dt >= d.dt and txn_dt < d.dt + 1;
    end if;
end loop;
7-step technique

1. Create ScratchTable
2. Bulk Loads
3. Table & Col Stats
4. Index Creates
5. Exchange Partition

Range-hash composite-partitioned TXN

Hash-partitioned TXN_P20140224

<table>
<thead>
<tr>
<th>22-Feb 2014</th>
<th>23-Feb 2014</th>
<th>24-Feb 2014</th>
<th>(empty)</th>
<th>24-Feb 2014</th>
</tr>
</thead>
</table>

22-Feb 2014
23-Feb 2014
24-Feb 2014
(empt)
7-step technique

1. Create ScratchTable
2. Bulk Loads
3. Table & Col Stats
4. Index Creates
5. Exchange Partition

Composite-partitioned table TXN

Hash-partitioned TXN_P20140224


(950,000 rows)
7-step technique

Composite-partitioned table TXN

22-Feb 2014
23-Feb 2014
24-Feb 2014
25-Feb 2014

Hash-partitioned TXN_P20140224

24-Feb 2014

6. Drop Indexes
7. Bulk load
7 step technique

1. Create temporary table TXN_P20140224 as a hash-partitioned table
2. Perform parallel, append load of data into TXN_P20140224
3. Gather CBO statistics on table TXN_P20140224
   • Only table and columns stats
4. Create indexes on TXN_P20140224 matching local indexes on TXN
5. alter table TXN
   exchange partition P20140224 with table TXN_P20120224
   including indexes without validation update global indexes;
6. Drop indexes on TXN_P20120224
7. Perform parallel, append load of data into TXN_P20120224
8. ...and...
...OK, more than 7 steps...

• Need to determine how long to retain date-stamped “scratch” tables
  – EXCHPART.PREPARE procedure first checks if the proposed “scratch” table exists
    • If not, then creates it from base partition
    • Otherwise, just use what exists
  – Need to drop “scratch” tables after \( N \) load cycles
MERGE / Up-sert logic

• Slowly-changing dimension tables
  – Change often enough to require time-variant image of data
    • Should be loaded similar to fact tables using basic 5-step or advanced 7-step EXCHANGE PARTITION loads
  – Also require current point-in-time image of data
    • MERGE or update-else-insert (a.k.a. up-sert) logic
      – If row exists, then update, else insert
MERGE / Up-sert or...

- So we could either do it this way...

```sql
merge into curr_acct_dim
using (select * from acct_dim
    where eff_dt >= '25-FEB-2014'
    and eff_dt < '26-FEB-2014')
when matched then update set ...
when not matched then insert ...;
```
...or EXCHANGE PARTITION

1. Create temporary table ACCT_SCRATCH as a hash-partitioned table
2. Perform parallel, append load of data into ACCT_SCRATCH
   - Nested in-line SELECT statements doing UNION, ranking, and filtering
3. Gather CBO statistics on table ACCT_SCRATCH
4. Create indexes on ACCT_SCRATCH matching local indexes on CURR_ACCT_DIM
5. alter table CURR_ACCT_DIM
   - exchange partition PDUMMY with table ACCT_SCRATCH including indexes without validation;
**CURR_ACCT_DIM**
- Range-hash composite-partitioned
- Range partition key column = PK column
- Single range partition named PDUMMY
- B*Tree index on PK (local)
- Bitmap indexes (local) on attributes

**ACCT_SCRATCH**
- Hash partitioned
- Hash partition key column same as CURR_ACCT_DIM
- Indexes created to match local indexes on CURR_ACCT_DIM

**Merge / Up-sert**

Exchange

Partition
**Merge / Up-sert**

```
INSERT /*+ append parallel(t,8) */ INTO ACCT_SCRATCH t
SELECT ... *(list of columns) ...
FROM (SELECT ... *(list of columns) ...
      ROW_NUMBER() over (PARTITION BY acct_key
                        ORDER BY eff_dt desc) rn
      FROM CURR_ACCT_DIM
      UNION ALL
      SELECT ... *(list of columns) ...
      FROM ACCT_DIM partition(P20140225))
WHERE RN = 1;
```

1. Inner-most query pulls newly-loaded data from ACCT_DIM, unioned with existing data from type-1 CURR_ACCT_DIM
2. Middle query ranks rows within each ACCT_KEY value, sorted by EFF_DT in descending order
3. Outer-most query selects only the latest row for each ACCT_KEY and passes to INSERT
4. INSERT APPEND (direct-path) and parallel, can compress rows, if desired
Merge / Up-sert

• Assume that...
  – CURR_ACCT_DIM has 15m rows total
  – 1m new rows just loaded into 25-Feb partition of ACCT_DIM
    • 100k (0.1m) rows are new accounts, 900k (0.9m) rows changes to existing accounts

• Then, what will happen is...
  – Inner-most query in SELECT fetches 15m rows from CURR_ACCT_DIM unioned with
    1m rows from 25-Feb partition of ACCT_DIM, returning **16m rows** in total
  – Middle query in SELECT ranks rows within each ACCT_KEY by EFF_DT in descending
    order, returning **16m rows**
  – Outer-most query in SELECT filters to most-recent row for each ACCT_KEY,
    returning **15.1m rows**
  – Inserts **15.1m rows** into ACCT_SCRATCH
Summary

1. During load cycles, load time-variant type-2 tables...
   ● Either using basic 5-step EXCHANGE PARTITION load technique when load cycles match granularity of range partitions...
   ● Or using 7-step EXCHANGE PARTITION load technique for “dribble effect” when load cycles do not match granularity of range partitions

2. ...then, merge newly-loaded data from time-variant tables into point-in-time type-1 tables
   ● Using EXCHANGE PARTITION load technique to accomplish merge / up-sert logic
Thank You!

Tim’s contact info:
- Web: http://www.EvDBT.com
- Email: Tim@EvDBT.com

- “Scaling to Infinity” paper by Tim Gorman
- “Supercharging Star Transformations” by Jeff Maresh
- “Managing the Data Lifecycle” by Jeff Maresh

- “exchpart.sql” package