An Extensible Materialized View Architecture

NYOUG Meeting
September, 2006

Claudio Fratarcangeli
Adept Technology Inc.
claudiof@computer.org

To appear in SELECT Journal, 4th Quarter 2006

Oracle Materialized Views

- Replicate and transform data from one set of tables to another
- Materialized view is target table in copy/transformation process
- Changes to source(master) tables captured in log tables via triggers on master tables

Oracle Materialized Views

- Information in log tables used to incrementally refresh the materialized views
- Materialized view based on one or more master tables
- Materialized view and master tables can be in different databases
- Transformation logic based upon a query defined against the master tables

Limitations of Oracle Materialized Views

- Support for incremental refresh is limited.
- Many restrictions on the nature of the defining query in order to support incremental refresh.
- Restrictions on the defining query for incremental refresh vary depending upon whether or not the master tables and materialized view are in the same database or in different databases.
- Complex materialized views with joins require use of ROWID based materialized view logs. This makes it impossible to reorganize (i.e. to defragment) a materialized view table without doing a full refresh.

Limitations of Oracle Materialized Views

- Incremental refresh query plans can be very inefficient for complex defining queries.
  - For example, materialized views with remote masters and subqueries are very inefficient when the level of nesting of subqueries is two or more.
- Limited ability to tune refresh queries generated by Oracle
  - Alter optimization parameters
  - Alter stored table and index statistics

Limitations of Oracle Materialized Views

- If the materialized view becomes out of synch with the master because of media failure
  - Full refresh must be performed to resynchronize the materialized view with the master.
  - No capability to automatically and efficiently incrementally resynchronize materialized view with master
- No statistics are recorded indicating number of rows updated, deleted, inserted during a fast refresh.

Workarounds to Oracle Materialized View Limitations

- Create nested intermediate materialized views that can be fast refreshed
  - Requires additional space for the intermediate materialized views.
  - Full and delta refreshes require extra time because multiple materialized views must be refreshed.
  - Added complexity and time to recover from media failure. Need to restore and refresh intermediate materialized views as well as the target materialized view.

Model Refresh/Transformation Architecture

- Modeled after Oracle materialized views
- Suited to efficient movement and transformation of large volumes of data
- Moves all changes that happened since last refresh in bulk
- Overhead at source due to triggers on master tables
  - Reasonable tradeoff for increased refresh speed

Features of Model Architecture Borrows from Materialized Views

- Concept of materialized view is supported
- Changes to master tables are captured via triggers in log tables.
- Fast incremental refresh supported
- Refresh Groups supported
  - Atomic refresh of a group of materialized views

Features of Model Architecture not found in Oracle Materialized Views

- Fast Incremental refresh of arbitrarily complex materialized views
- Choices for transformation logic:
  - A single query against a set of master tables
  - An arbitrarily complex stored procedure.
- Declarative specification of transformation logic for typical transformations
- Incremental resynch of materialized view with master in case of media failure. No full refresh required.
- Materialized view may be a table in a non-Oracle database.

Features of Model Architecture Not found in Materialized Views

- The master table logs store only primary key column(s) of inserted, updated, or deleted rows and a small fixed number of control columns.
  - Non-key column values are not stored
    - Low space consumption in log tables
    - Inserts into log tables are efficient
  - Master table reorganizations do not disrupt replication

Limitations of Model Architecture

- Master table triggers add overhead although only PK column values are stored
- Direct path loads on masters requires application to explicitly insert rows into log tables to support fast refresh.
  - Oracle doesn’t support fast refresh for direct path loads into remote master tables at all.
- All master tables for a single materialized view must reside on the same database

Fast Refresh: Identifying Changed Rows

- Identify rows updated, deleted, inserted in master table(s) since time of last refresh
- Record timestamp of last refresh in log table along with primary key values
- Problem:
  - In trigger we store timestamp of DML operation on master table
  - Instead we really need to store timestamp at time of commit of DML operation

DML Timestamp Versus Commit Timestamp

Example: 2 concurrently executing transactions

- Transaction 1: makes changes to a master table, DEPT
- Transaction 2: Concurrently refreshes a materialized view based on DEPT

- Timestamp of most recent refresh of DEPT is stored in a catalog table
- Assume DEPT was last refreshed at time, T80

DML Timestamp Scenario

Time	Transaction 1	Transaction 2
T100	Update DEPT row. Row inserted into log table with timestamp, T100.	
T103		Records the current timestamp, T103, in a catalog table. We will use this timestamp to identify newly inserted log table rows the next time we do a refresh.
T104		Fetch log table rows with timestamp > T80 (time of last refresh). For each log table row retrieved propagate changes to materialized view table and commit
T105	Commits	

DML Timestamp Versus Commit Timestamp Scenario

- During next refresh we fetch log table rows with timestamp > T103
- T103 is timestamp of most recent refresh
- Log table row inserted at time T100 will not be picked up because T100 < T103
- Log table row inserted at time T100 was also not picked up in prior refresh
- We never replicate update performed by Transaction 1 at T100

Commit Timestamp

- Store commit SCN (System Change Number) instead of DML timestamp in log table row
- Oracle built-in returns commit SCN for current transaction
  - USERENV(‘COMMITSCN’)
  - Can only be invoked once per transaction and stored in a single column in a single row
- Oracle initially stores a place holder in row/column
- Oracle remembers what row/column placeholder was stored
- At commit time Oracle goes back to row/column and sets it to the actual commit SCN of current transaction

Commit Timestamp: Multiple Rows Updated Per Transaction

- Problem:
  - Can only store commit SCN in one log table row
- Solution:
  - Store SCN in a separate table, SCN_HISTORY, along with current tranasaction_id
  - Only one row inserted into SCN_HISTORY per transaction
  - Store transaction_id in log table rows
  - Built-in returns local transaction id:
    - DBMS_TRANSACTION.LOCAL_TRANSACTION_ID

Commit Timestamp: Multiple Rows Updated Per Transaction

```
Table: DEPT_LOG Table
   TRANSACTION_ID VARCHAR2(100) (PK)
   DEPT_NO        NUMBER        (PK)

   Foreign key: TRANSACTION_ID
      references SCN_HISTORY.TRANSACTION_ID

Table: SCN_HISTORY
   TRANSACTION_ID VARCHAR2(100) (PK)
   COMMIT_SCN     NUMBER        (UNIQUE KEY)
```

Commit SCN

DEPT_LOG Table:

TRANSACTION_ID	DEPT_NO
10.40.30	40
10.40.30	53
10.40.30	60

SCN_HISTORY Table:

TRANSACTION_ID	COMMIT_SCN
10.40.30	9900

Commit SCN Scenario

Time	**Transaction 1**	**Transaction 2**
T1	Updates DEPT row Row inserted into DEPT_LOG with transaction_id, 10.40.30 Row inserted into SCN_HISTORY with transaction_id, 10.40.30 and USERENV('COMMIT_SCN')	
T3		Records the current system SCN, 9890, in a catalog table.
T4		Fetch DEPT_LOG rows with COMMIT_SCN > last refresh SCN. Update materialized view table. Commit
T5	Commits. Oracle automatically updates the COMMIT_SCN	

Commit SCN Scenario

- Most recent refresh SCN stored in catalog for DEPT is 9890
- Commit SCN, 9900, is greater than 9890
- Next time we do refresh, log row created at time T1, will be retrieved

Oracle 10G: ORA_ROWSCN

- Pseudo-column, ORA_ROWSCN, available in all tables
- Contains upper bound estimate of the commit SCN for a row in any table.
- Eliminates need to explicitly store transaction_id in log table and to store Commit SCN in SCN_HISTORY table.

Extensible Framework Materialized View Framework

- Standard materialized view refresh functionality is built into the framework
  - Typical types of transformations
  - Concurrency control
  - Keeping track of last refresh SCN in catalog tables
  - Other bookkeeping tasks
- Non-standard functionality is also supported:
  - Custom written refresh procedures can be plugged into the framework to support arbitrarily complex types of transformations

Service Provider Interface (SPI) for Custom Refresh Logic

- SPI is a standard call interface that must be supported by a custom written refresh procedure
- A custom written refresh procedure must support the following parameters:
  - Materialized view name (INPUT)
  - Master database link name (INPUT)
  - Last refresh SCN (INPUT)
  - Number of rows inserted (OUTPUT)
  - Number of rows updated (OUTPUT)
  - Number of rows deleted (OUTPUT)

Custom Refresh Procedure

- Uses master database link name to connect to master database
- Uses last refresh SCN to retrieve master log table rows inserted since last refresh
- Fetches data from master tables based upon rows found in log tables and performs custom refresh logic updating target materialized view
- Updates 3 output parameters with rows processed during refresh
- Framework records row counts in a log and records SCN of current refresh in a catalog table to be used during next refresh cycle.

Meta-Data Maintained by Framework

- Schema.table_name of materialized view.
- Name of custom written refresh procedure (optional)
- Defining query for the materialized view (if standard refresh functionality is used)
- Last refresh SCN for the materialized view.
- List of master tables and database link pointing to master table database.
- Logical foreign keys in the materialized view table that reference primary key columns of master tables.
- For each master table an indication of whether or not the join to the master table in the defining query is an outer or inner join.

Meta Data about Master Tables Maintained by Framework

- Schema.table_name
- List of materialized views and their locations that reference the master table.
- For each referencing materialized view, the last refresh SCN of the materialized view.
- Name of log table.
- Primary key of master table.

Refresh Control Logic

- Start a serializable transaction.
- Retrieve and lock meta-data about the materialized view and its master tables
  - Lock meta data about materialized view in exclusive mode
  - Lock meta data about master table in share mode
- Obtain and save in meta-data repository the current SCN at the master site using
  - SYS.DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER
  - This becomes last refresh SCN to be used during next refresh

Refresh Control Logic (continued)

- Invoke the refresh procedure passing in the last refresh SCN and master database link as arguments
- Record statistics about the refresh including number of rows processed and duration.
- Commit the transaction releasing the locks on the meta-data.

Refresh Logic for a Materialized View Consisting of Joins

- Materialized view is defined as a join query among master tables
- Assumptions:
  - All joins among master tables must be equi-joins.
  - There must exist a column in the materialized view for each join column referenced in an equi-join predicate between master tables.
  - An actual Oracle view exists representing the defining query for the materialized view.

Refresh Logic for a Materialized View Consisting of Joins: Steps

- Identify rows from the log of each master that have been created since the last refresh SCN.
- For each newly created master log row:
  - Identify the primary key column(s) value(s) of the materialized view rows that might need to be updated, deleted, or inserted.
  - Simplifies the task of identifying the rows that need to be deleted from the materialized view table and the rows from the defining view query that must be merged into the materialized view table.
- For each primary key value from prior step, perform the required update, insert, or delete.

Sample Master Tables

- Table: EMP
  - Columns:
    - EMP_ID (PK)
    - NAME
    - DEPT_ID (FK references DEPT.DEPT_ID)
    - COMPANY_SITE_ID (FK references COMPANY_SITE.SITE_ID)
- Table: DEPT
  - Columns:
    - DEPT_ID (PK)
    - NAME
- Table: COMPANY_SITE
  - Columns:
    - SITE_ID (PK)
    - NAME

Master log tables

- Table: EMP_CLOG
  - Columns:
    - TRANSACTION_ID (PK)
    - EMP_ID (PK)
- Table: DEPT_CLOG
  - Columns:
    - TRANSACTION_ID (PK)
    - DEPT_ID (PK)
- Table: COMPANY_SITE_CLOG
  - Columns:
    - TRANSACTION_ID (PK)
    - SITE_ID (PK)

Materialized view table

- Table: EMP_MVIEW
  - Columns:
    - EMP_ID (PK)
    - EMP_NAME
    - DEPT_ID (Logical FK references DEPT.DEPT_ID)
    - DEPT_NAME
    - COMPANY_SITE_ID (Logical FK references COMPANY_SITE.SITE_ID)
    - COMPANY_SITE_NAME

Oracle View Containing Defining Query of Materialized View

View: EMP_MVIEW_VW

```
SELECT e.emp_id, e.name emp_name, e.dept_id,
    d.name dept_name,  e.company_site_id,
    c.name company_site_name
FROM
  emp e
    JOIN
  dept d ON
      (e.dept_id = d.dept_id)
    LEFT OUTER JOIN
  company_site c ON
      (e.company_site_id = c.site_id)
```

Distributed Database Considerations

- Strategy to work around limitations of Oracle’s distributed query optimizer
  - Create views dynamically on master database site
  - Use a local temp table to store primary keys of materialized view rows to update,insert,delete
- Temp Key Table on Local Site:
  - EMP_MVIEW_KEY
    - Columns:
      - EMP_ID number
      - OPERATION_TYPE varchar(2)

Inner Join Master Tables

- Changes to inner join tables can cause inserts, updates, or deletes in materialized view
  - Updates,inserts,deletes of EMP or DEPT can cause updates,deletes, inserts of rows in EMP_MVIEW_VW
- Changes to outer join tables can only cause updates
  - Updates,inserts,deletes of COMPANY_SITE in EMP_MVIEW_VW can only cause rows in EMP_MVIEW_VW to be updated

Oracle View: EMP_MVIEW_KEY_VW

View to retrieve PK’s of materialized view rows to be inserted/updated because of operations on DEPT or EMP

```
SELECT s.commit_scn, v.emp_id
FROM scn s, dept_clog l, emp_mview_vw v
WHERE s.transaction_id = l.transaction_id
  AND l.dept_id = v.dept_id
UNION
SELECT s.commit_scn, v.emp_id
FROM scn s, emp_clog l, emp_mview_vw v
WHERE s.transaction_id = l.transaction_id
  AND l.emp_id = v.emp_id
```

Inner Join Master Tables

- Get PK’s of rows to be inserted or updated in materialized view:

```
INSERT INTO emp_mview_key (emp_id, operation_type)
SELECT DISTINCT emp_id, 'UI'
FROM emp_mview_key_vw@masterdb
WHERE commit_scn > :last_refresh_scn
```

- The operation_type is set to 'UI' indicating update or insert.
- :last_refresh_scn is the last refresh SCN for the EMP_MVIEW materialized view.

Identify Rows to be Deleted from Materialized View

- Cannot find PK of rows to delete on master database because master table rows are gone
- Query materialized view itself to find PK’s of rows to be deleted
- Use PK’s of master tables found in log tables to query materialized view

Identify Rows to be Deleted from Materialized View

```
INSERT INTO emp_mview_key (emp_id, operation_type)
       SELECT emp_id, ‘D’
       FROM emp_mview
       WHERE dept_id in
         (SELECT /*+ no_merge(a) */ *
         FROM
           (SELECT DISTINCT dept_id
           FROM dept_clog@master_db
           WHERE transaction_id IN
             (SELECT transaction_id
             FROM scn@master_db
             WHERE commit_scn > :last_refresh_scn))) a
       UNION
       SELECT emp_id, ‘D’ FROM emp_mview WHERE emp_id IN ….
       MINUS
       SELECT emp_id, ‘D’
       FROM emp_mview_key
```

Identify Rows to be Deleted from Materialized View

- MINUS excludes emp_id’s inserted in prior step which we know to be updates or inserts
- Embed the remote subqueries for each query fragment in an in-line view
- NO_MERGE hint forces the optimizer to execute the entire subquery at the remote site.

Outer Join Master Tables

- Changes to outer join master table rows can only cause rows to be updated
- Query materialized view by PK’s of outer join master tables found in logs
- This is why we require columns corresponding to primary key’s of master tables to exist in our materialized view

Identify Rows to be Updated because of Outer Join Master Tables

```
INSERT INTO emp_mview_key (emp_id, operation_type)
      SELECT emp_id, ‘UI’
      FROM emp_mview
      WHERE company_site_id IN
        (SELECT /*+ no_merge(a) */ *
        FROM
          (SELECT DISTINCT site_id
          FROM company_site_clog@master_db
          WHERE transaction_id IN
            (SELECT transaction_id
            FROM scn@master_db
            WHERE commit_scn > :last_refresh_scn))) a
      MINUS
      SELECT emp_id, ‘UI’
      FROM emp_mview_key
```

Apply Changes to Materialized View Table

- PK’s of all rows to be updated, inserted, deleted and operation type exists in EMP_MVIEW_KEY table
- Create following view on master site for efficiency:

```
CREATE VIEW emp_mview_upsert AS
SELECT *
FROM emp_mview_vw
WHERE emp_id IN
  (SELECT emp_id
  FROM emp_mview_key@mview_db
  WHERE operation_type = 'UI')
```

Merge Updates and Inserts into Materialized View

```
MERGE INTO emp_mview mv
USING emp_mview_upsert@master_db v
ON (mv.emp_id = v.emp_id)
WHEN MATCHED THEN
  SET mv.emp_name = v.emp_name,
          mv.dept_id = v.dept_id,
          mv.dept_name = v.dept_name,
          mv.company_site_id = v.company_site_id,
          mv.company_site_name = v.company_site_name
WHEN NOT MATCHED THEN
  INSERT (mv.emp_id, mv.emp_name, mv.dept_id, v.dept_name,
        mv.company_site_id, mv.company_site_name)
    VALUES
        (v.emp_id, v.emp_name, v.dept_id, v.dept_name,
        v.company_site_id, v.company_site_name)
```

Delete Rows from Materialized View

```
DELETE FROM emp_mview
WHERE emp_id IN
  (SELECT emp_id
  FROM emp_mview_key
  WHERE operation_type = 'D')
```

Resynch After Media Failure on Materialized View Site

- Media failure on materialized view database requires restore of old backup
- Backup was taken prior to last refresh
- Restored backup is out of synch with last refresh SCN stored in catalog on master database
- Normally a full refresh is required to resynch materialized view with masters
- In our case an incremental resynch is possible

Incremental Resynch After Media Failure on Materialized View Site

- During refresh compare last refresh scn’s stored in materialized view database catalog and master database catalog
- If last refresh SCN on materialized view site is earlier than last refresh SCN on master site then
  - Use the earlier last refresh SCN from materialized view site to do refresh
- We reprocess master log records that were processed prior to restore of old backup
- As long as old log records still exist this will work
- Materialized view is brought back in synch with master tables

Preserving Data Consistency for Master Tables During Refresh

- Updates on master tables can occur during refresh
- Consistency of data in master tables can change during refresh
- Would like to logically freeze contents of master tables during refresh
- 2 ways to do this:
  - Serializable transactions
  - Flashback Query

Preserving Data Consistency

Time	User 1: Refresh Transaction	User 2: Update transaction
T1	Read master table DEPT	
T2		Delete a row from DEPT read by User 1 at time, T1. Commit
T3	Read detail table EMP	
T4	Update target materialized view based on DEPT and EMP. Commit	

EMP rows refreshed without related master DEPT row.

Serializable Transaction

- A read-only transaction that also allows updates
- A logical snapshot of database is taken at start of transaction
- All reads occur against the logical snapshot
- Updates performed by other transactions are not seen

Flashback Query

- Create a logical snapshot of database as of a specific point in time
  - Reads occur against the logical snapshot
- Syntax:
  - SELECT * FROM DEPT AS OF SCN 234
  - 234 is the SCN of the last refresh

Preserving Data Consistency of Materialized Views During Refresh

- Multiple related materialized views could be refreshed as a group but in different transactions
- Refresh a set of related materialized views in parallel in different transactions
- Changes to related materialized views could be committed at different times
- Readers could see an inconsistent view of data across different materialized views during refresh

Preserving Data Consistency of Materialized Views During Refresh

- Use flashback query
- Record current SCN of materialized view database in a table at the start of each refresh.
- Readers use the stored SCN to perform flashback queries against materialized views during refresh
  - SELECT * FROM … AS OF SCN 433
  - 433 is SCN just before start of refresh
- Flashback query is only used while a refresh is happening

Alternatives to Materialized Views: Oracle Streams

- Propogate individual low level operations captured in redo log
- Much less efficient than bulk movement of data changes captured in master table logs

Alternatives to Materialized Views: Oracle Data Capture

- Changes to master tables are captured in change tables
- Both before and after images of rows are stored in change tables. This incurs a fair amount of overhead.
- Data in change tables is used to replicate data to target table(s).
- Synchronous Mode:
  - Changes to master tables captured in change tables via triggers
  - Similar to Materialized Views
- Asynchronous Mode:
  - Oracle streams is used to mine redo log and store changes in change tables.