Three R’s of Data Warehouse Fault Tolerance

Stewart Bryson, Managing Director, Rittman Mead America
Oracle Open World 2010, San Francisco, September 2010
Who am I?
• Stewart Bryson, Managing Director, Rittman Mead America
• Oracle BI/DW Architect and Delivery Specialist
• Oracle Database and BI Stack since 1996
• Regular Community Speaker
• Writer for Rittman Mead Blog: http://www.rittmanmead.com/blog
• Developer of Transcend Framework
• Email: stewart.bryson@rittmanmead.com
• Twitter: @stewartbryson
Who is Rittman Mead?
• Oracle BI&DW Experts
• Consulting, Training and Support
• Global Company, Offices in the US, UK and India
• Clients in the UK, USA, Europe, Middle-East
• Voted UKOUG BI Partner of the Year 2008
• 13 Sessions at Oracle Open World 2010
• Twitter : @stewartbryson
Adding Default Rows with OBIEE

May 3rd, 2010 by Stewart Bryson

When looking at a typical dimensional model, there are cases when the application of one or more dimension tables may not make sense for particular fact rows. We handle these situations with special rows in the dimension table that specify this lack of relationship, sometimes known generically as "default" rows. The three basic cases of default rows are: the "placeholder" row, the "not applicable" row, and the "unknown" row.

The placeholder row is typically seen in cases with late-arriving dimension records. Perhaps a latency issue exists between two source systems, and we aren’t yet aware which dimension row we should apply. So we make a temporary association with a placeholder row until the appropriate dimension row is loaded. Or perhaps we have data quality issues in the source system, and we’ll never know which dimension record should be applied, but we want to load the fact rows anyway. This is usually referred to as the "unknown" dimension row. Finally, a particular dimension may not have a logical application for this row. For instance, an order may not have a particular promotion associated with it. This is typically referred to as the "not applicable" row.
- Longest-running Oracle (BI) Blog
- OBIEE, OWB, BI, data warehousing, Hyperion, etc.
- Step-by-step demonstrations
- Test cases
- Product evaluations
- Authors include
  - Mark Rittman (ACE Director)
  - Venkat Janakiraman (ACE)
  - Stewart Bryson
  - Borkur Steingrimsson
  - Peter Scott
Rittman Mead Oracle BI Training Days 2010

- Running in London, Atlanta and Bangalore, Fall 2010
- Three days of intense OBIEE 11gR1 hands-on training
- Led by the Rittman Mead 11g Beta Testing team
What will this session answer?
‣ What do I mean by “fault tolerance”?

‣ How is fault tolerance different for BI/DW systems?

‣ Are there standard features of the Oracle Database that can help with fault tolerance?

‣ What do I mean by “Resuming”, “Restarting” and “Restoring”?
How are BI/DW Systems Different?
• Millions or billions of rows
• Constraints are often disabled
  ‣ No referential integrity protecting errant ETL runs
  ‣ Long query processing to find issues
• DDL operations included in ETL code
• Large memory sizes, parallelism
What is “Fault Tolerance” for BI/DW?
• Ability to recover from different kinds of errors
  ‣ Hardware and software
  ‣ General System issues: space, network latency
  ‣ Human mistakes

• Elimination of aftermath scenarios
  ‣ Data correction scenarios
  ‣ Long rerun processes
What is available to us with the database?
The first thing we should do is exploit the features in the database we already own

Threat a BI/DW environment as a production system

Simple errors should never cause downtime

Complex errors should cause minimal downtime

Understand that ETL processes will sometimes break and plan for that
Resuming
• Error is detected and processing is suspended
• Once the error is corrected, processing continues automatically
• Oracle Database offers Resumable Space Allocation
  ▶ Available since 9i
  ▶ Functionality for all space-related errors: target datafiles, tempfiles, UNDO space
• Supported for SELECT, DML and DDL
• Available in Oracle Utilities
  ▶ Datapump (automatic)
  ▶ SQL-Loader
  ▶ Export/Import
Enabling Resumable Space Allocation

```
SQL> alter system
    2   set resumable_timeout=3600;
System altered.
Elapsed: 00:00:00.08
SQL>
```
Enabling Resumable Space Allocation

- Enabled at the system Level
  - RESUMABLE_TIMEOUT parameter
Enabling Resumable Space Allocation

- Enabled at the system Level
  - RESUMABLE_TIMEOUT parameter
- Enabled at the session Level
  - ENABLE RESUMABLE
  - TIMEOUT
  - NAME

```
SQL> alter session
2    enable resumable
3    timeout 7200
4    name 'Sales Fact Load';

Session altered.
Elapsed: 00:00:00.00
SQL> |
```
Enabling Resumable Space Allocation

• Enabled at the system Level
  ‣ RESUMABLE_TIMEOUT parameter

• Enabled at the session Level
  ‣ ENABLE RESUMABLE
  ‣ TIMEOUT
  ‣ NAME

• Enabled with command-line parameters for Oracle Utilities
  ‣ RESUMABLE
  ‣ RESUMABLE_NAME
  ‣ RESUMABLE_TIMEOUT
Monitoring Resumable Sessions

```
SQL> select name, start_time, suspend_time, status from dba_resumable;

NAME            | START_TIME       | SUSPEND_TIME       | STATUS        
-----------------|------------------|--------------------|---------------
SALES fact load2 | 02/06/10 10:33:33| 02/06/10 10:33:33  | SUSPENDED     
SALES fact load  | 02/06/10 10:29:03| 02/06/10 10:29:03  | SUSPENDED     

2 rows selected.

SQL> select name, error_msg from dba_resumable;

NAME            | ERROR_MSG                                                   
-----------------|-------------------------------------------------------------
SALES fact load2 | ORA-01653: unable to extend table TARGET.SALES by 8 in tablespace TARGET 
SALES fact load  | ORA-01653: unable to extend table TARGET.SALES by 8 in tablespace TARGET 

2 rows selected.
```

T: (888) 631 1410 or +44 (0) 8446 697 995   E: info@rittmanmead.com   W: www.rittmanmead.com
Monitoring Resumable Sessions

- **Invalid Objects by Schema**
  - Name: Owner's Invalid Object Count
  - Message: 208 object(s) are invalid in the PUBLIC schema.
  - Date: Feb 6, 2010 12:51:07 AM

- **Invalid Objects by Schema**
  - Name: Owner's Invalid Object Count
  - Message: 12 object(s) are invalid in the OWBSYS_AUDIT schema.
  - Date: Feb 6, 2010 12:51:07 AM

- **User Audit**
  - Name: Audited User
  - Message: User SYS logged on from localhost.localdomain.
  - Date: Feb 6, 2010 10:16:54 AM

- **Session Suspended**
  - Name: Session Suspended by Tablespace Limitation
  - Message: Operation on resumable session SALES fact load session id 42 suspended because of errors in tablespace TARGET. Error message is ORA-01653: unable to extend table TARGET.SALES by 8 in tablespace TARGET.
  - Date: Feb 6, 2010 10:29:03 AM

- **Waits by Wait Class**
  - Name: Database Time Spent Waiting (%)
  - Message: Metrics "Database Time Spent Waiting (%)" is at 98.96692 for event class "Configuration".
  - Date: Feb 6, 2010 10:32:26 AM

- **Tablespaces Full**
  - Name: Tablespace Space Used (%)
  - Message: Tablespace TARGET is 100 percent full.
  - Date: Feb 6, 2010 10:37:42 AM
Monitoring Resumable Sessions

```
Sat Feb 06 18:35:42 2010
statement in resumable session 'SALES fact load' was suspended due to
 ORA-01653: unable to extend table TARGET.SALES by 8 in tablespace TARGET
Sat Feb 06 18:40:27 2010
statement in resumable session 'SALES fact load2' was suspended due to
 ORA-01653: unable to extend table TARGET.SALES by 8 in tablespace TARGET
Sat Feb 06 18:49:19 2010
alter database datfile '/oracle/oradata/bidw1/target01.dbf' autoextend on next 10M maxsize 1000M
Completed: alter database datfile '/oracle/oradata/bidw1/target01.dbf' autoextend on next 10M maxsize 1000M
Sat Feb 06 18:49:20 2010
statement in resumable session 'SALES fact load' was resumed
Sat Feb 06 18:49:21 2010
statement in resumable session 'SALES fact load2' was resumed
Sat Feb 06 18:49:39 2010
Thread 1 advanced to log sequence 109 (LGWR switch)
 Current log# 1 sec# 109 mem# 0: /oracle/oradata/bidw1/redo01.log
Sat Feb 06 18:50:02 2010
Thread 1 cannot allocate new log, sequence 110
Private strand flush not complete
Current log# 1 sec# 110 mem# 0: /oracle/oradata/bidw1/redo01.log, 46634
```

T: (888) 631 1410 or +44 (0) 8446 697 995  E: info@rittmanmead.com  W: www.rittmanmead.com
Monitoring Resumable Sessions

```
SQL> SELECT event,
        2       SUM(time_waited) time_waited,
        3       SUM(total_waits) total_waits,
        4       AVG(average_wait) average_wait
        5       FROM gv$session_event
        6       WHERE lower(event) LIKE 'suspend%'
        7       GROUP BY event
        8       ORDER BY time_waited ASC
        9 /

EVENT                                               | TIME_WAITED | TOTAL_WAITS | AVERAGE_WAIT
----------------------------------------------------|-------------|-------------|---------------
statement suspended, wait error to be cleared       | 305373      | 1377        | 221.78        

1 row selected.

SQL>
```
AFTER SUSPEND Trigger
AFTER SUSPEND Trigger

- System-wide trigger fires when operations suspended
- Can use the DBMS_RESUMABLE package
  - SPACE_ERROR_INFO function
  - SET_TIMEOUT
- Seems unnecessary
  - BI/DW environments should be production-ready
  - Should already have monitoring and alerts
  - AUTOEXTEND handles adding space
Demonstration

Resumable Space Allocation
Restarting
• Complex errors occur during ETL processing
  › Hardware and software issues
  › Unforeseen data issues
  › Human error
  › Resumable operations are useless
• Planning means eliminating aftermath scenarios
• Investigate the current landscape for errors
• Easy errors can sometimes be corrected with a simple DML statement
• **What about the rest?**
Planning For Complex Errors
Planning For Complex Errors

- Need a framework that makes it easy to identify major milestones in the ETL process
- Need to catch unexpected errors
- Not the same as “code-controlled restartability”
Planning For Complex Errors

• Need a framework that makes it easy to identify major milestones in the ETL process
• Need to catch unexpected errors
• Not the same as “code-controlled restartability”
Code-Controlled Restartability
Code-Controlled Restartability

- Coding for exceptions is a best practice and should always be done, but we can’t catch everything
Oracle Flashback Technology
• Revert whole database, or just portions of it
• Flashback Table or Flashback Database
• Point-in-time is the System Change Number (SCN)
• Can create restore points, which are “named SCN’s”
• Alternatively, log SCN’s with messages in the log table
Flashback Table

• Completely an UNDO operation
• Identify a particular SCN to flashback to
• Can only flashback to SCN’s still in the UNDO
• This is not “really” new functionality
• Doesn’t require operations involvement
  ‣ grant flashback on sales_fact to ETL;
Demonstration

Flashback Table
Flashback Database
Flashback Database Scenario: Aftermath

• ODS and staging-area lookup components
  ‣ Deleting from history tables
  ‣ Correcting MDM scenarios

• Star Schemas
  ‣ SCD Type 2 or Hybrid Type 1/Type 2
  ‣ Deleting or updating fact tables
  ‣ Recalculating aggregates

• Difficult to quantify exactly what went wrong
• Need a complete do-over
• Alternatives
  ‣ Execution ID
  ‣ Audit dimension
Flashback Database

• Not an UNDO operation like Flashback Table
• Uses Flashback Logs
  › Contains prior versions of changed blocks
  › Physical not logical
• Redo Logs (online or archived)
  › Contain logically applied SQL statements
  › Logical not physical
• Uses a combination of Flashback and Redo Logs
  › Find the data block from the Flashback Log just prior to the SCN
  › Use the REDO Logs to roll forward to the appropriate SCN
• Requires operations involvement
Preparing for Flashback Database

• Set the DB_RECOVERY_FILE_DEST_SIZE parameter
  ‣ SQL> alter system set
    db_recovery_file_dest_size=3G;

• Set the DB_RECOVER_FILE_DEST parameter
  ‣ SQL> alter system set db_recovery_file_dest='/u01/app/oracle/flash_recovery_area';

• Set the DB_FLASHBACK_RETENTION_TARGET parameter (in minutes)
  ‣ SQL> alter system set
    db_flashback_retention_target=2880;

• Turn on Flashback Database
  ‣ SQL> alter database flashback on;
Demonstration
Flashback Database
Should a BI/DW Database in in ARCHIVELOG Mode?

- Explanations for why BI/DW databases are usually in NOARCHIVELOG Mode
  - REDO generation is only important for transactional systems
  - Generation of REDO causes performance overhead
  - Putting a database in NOARCHIVELOG Mode eliminates REDO generation
Database in ARCHIVELOG Mode with a NOLOGGING Table

SQL> ALTER TABLE target.sales
  2  nologging;
Table altered.

SQL> SET autotrace on statistics
SQL> INSERT INTO target.sales
  2  SELECT * FROM sh.sales;
918843 rows created.

Statistics
-------------------------------------------------------
15401 recursive calls
33766 db block gets
13759 consistent gets
6534 physical reads
38250540 redo size
738 bytes sent via SQL*Net to client
942 bytes received via SQL*Net from client
4 SQL*Net roundtrips to/from client
123 sorts (memory)
0 sorts (disk)
918843 rows processed

SQL>

SQL> ALTER TABLE target.sales
  2  nologging;
Table altered.

SQL> SET autotrace on statistics
SQL> INSERT /*+ APPEND */ INTO target.sales
  2  SELECT * FROM sh.sales;
918843 rows created.

Statistics
-------------------------------------------------------
1 recursive calls
4637 db block gets
1718 consistent gets
1704 physical reads
8028 redo size
730 bytes sent via SQL*Net to client
956 bytes received via SQL*Net from client
4 SQL*Net roundtrips to/from client
1 sorts (memory)
0 sorts (disk)
918843 rows processed

SQL>
Database in ARCHIVELOG Mode with a NOLOGGING Table

SQL> ALTER TABLE target.sales 2  nologging;
Table altered.

SQL> SET autotrace on statistics
SQL> INSERT INTO target.sales 2  SELECT * FROM sh.sales;
918843 rows created.

Statistics
-------------------------------------------------------
 15401 recursive calls
 33766 db block gets
 13759 consistent gets
 6534 physical reads
38250540 redo size
 738 bytes sent via SQL*Net to client
 942 bytes received via SQL*Net from client
 4 SQL*Net roundtrips to/from client
 123 sorts (memory)
 0 sorts (disk)
918843 rows processed

SQL>

SQL> ALTER TABLE target.sales 2  nologging;
Table altered.

SQL> SET autotrace on statistics
SQL> INSERT /*+ APPEND */ INTO target.sales 2  SELECT * FROM sh.sales;
918843 rows created.

Statistics
-------------------------------------------------------
 1 recursive calls
 4637 db block gets
 1718 consistent gets
 1704 physical reads
8028 redo size
 730 bytes sent via SQL*Net to client
 956 bytes received via SQL*Net from client
 4 SQL*Net roundtrips to/from client
 1 sorts (memory)
 0 sorts (disk)
918843 rows processed

SQL>
Database in NOARCHIVELOG Mode

```
SQL> set autotrace on stat
SQL> insert into target.sales
    2  select * from sh.sales;
918843 rows created.

Statistics
-------------------------------------------------------
 9  recursive calls
33251  db block gets
10533  consistent gets
879  physical reads
38282140  redo size
1550  bytes sent via SQL*Net to client
1141  bytes received via SQL*Net from client
6  SQL*Net roundtrips to/from client
1  sorts (memory)
0  sorts (disk)
918843  rows processed

SQL>
```

```
SQL> set autotrace on stat
SQL> insert /*+ APPEND */ into target.sales
    2  select * from sh.sales;
918843 rows created.

Statistics
-------------------------------------------------------
0  recursive calls
4638  db block gets
1718  consistent gets
62  physical reads
7984  redo size
734  bytes sent via SQL*Net to client
956  bytes received via SQL*Net from client
4  SQL*Net roundtrips to/from client
1  sorts (memory)
0  sorts (disk)
918843  rows processed

SQL>
```
### Database in NOARCHIVELOG Mode

```sql
SQL> set autotrace on stat
SQL> insert into target.sales
  2  select * from sh.sales;
918843 rows created.

Statistics
-------------------------------------------------------
  9  recursive calls
 33251  db block gets
10533  consistent gets
  879  physical reads
38282140  redo size
 1550  bytes sent via SQL*Net to client
 1141  bytes received via SQL*Net from client
   6  SQL*Net roundtrips to/from client
   1  sorts (memory)
   0  sorts (disk)
918843  rows processed
SQL>
```

```sql
SQL> set autotrace on stat
SQL> insert /*+ APPEND */ into target.sales
  2  select * from sh.sales;
918843 rows created.

Statistics
-------------------------------------------------------
  0  recursive calls
 4638  db block gets
1718  consistent gets
   62  physical reads
  7984  redo size
  734  bytes sent via SQL*Net to client
  956  bytes received via SQL*Net from client
   4  SQL*Net roundtrips to/from client
   1  sorts (memory)
   0  sorts (disk)
918843  rows processed
SQL>
```
Real Meaning of NOARCHIVELOG
• All REDO is not suppressed
  ▸ REDO is still needed for crash recovery
  ▸ Oracle wouldn’t be able to open after a simple server crash
• NOARCHIVELOG Mode only means we are foregoing the ability to do Media Recovery
• Direct-path writes bypass the buffer cache
  ▸ Blocks inserted directly into the appropriate datafiles
  ▸ INSERT /*+ APPEND */ statements
  ▸ SQL-Loader direct-path mode
• These statements are not needed for crash recovery
  ▸ They already exist inside the datafile as they are committed
  ▸ No chance of losing them during server crashes
• Unless direct-path operations are occurring, then REDO is being generated regardless of ARCHIVELOG mode
Restoring (Actually, Backing Up)

SQL> alter database archivelog;

Database altered.

SQL> archive log list;

Database log mode          Archive Mode
Automatic archival          Enabled
Archive destination        USE_DB_RECOVERY_FILE_DEST
Oldest online log sequence 29
Next log sequence to archive 31
Current log sequence       31
SQL>
Restoring (Actually, Backing Up)

- Put the database in ARCHIVELOG Mode
  - Enables Online Backups
  - Enables Flashback Database

```sql
SQL> alter database archivelog;
Database altered.

SQL> archive log list;
Database log mode          Archive Mode
Automatic archival         Enabled
Archive destination        USE_DB_RECOVERY_FILE_DEST
Oldest online log sequence 29
Next log sequence to archive 31
Current log sequence       31
SQL>
```
Restoring (Actually, Backing Up)

• Put the database in ARCHIVELOG Mode
  ‣ Enables Online Backups
  ‣ Enables Flashback Database
• Create a Block Change Tracking file
  ‣ Increases backup performance
Restoring (Actually, Backing Up)

• Put the database in ARCHIVELOG Mode
  ‣ Enables Online Backups
  ‣ Enables Flashback Database
• Create a Block Change Tracking file
  ‣ Increases backup performance
• Modify tables to be NOLOGGING

```
SQL> alter table sales_fact nologging;
Table altered.

SQL> alter table customers nologging;
Table altered.
```

---

T: (888) 631 1410 or +44 (0) 8446 697 995  E: info@rittmanmead.com  W: www.rittmanmead.com

Thursday, September 23, 2010
Restoring (Actually, Backing Up)

- Put the database in ARCHIVELOG Mode
  - Enables Online Backups
  - Enables Flashback Database
- Create a Block Change Tracking file
  - Increases backup performance
- Modify tables to be NOLOGGING
- Perform INSERT /*+ APPEND */ statements as part of the ETL
Restoring (Actually, Backing Up)

- Put the database in ARCHIVELOG Mode
  - Enables Online Backups
  - Enables Flashback Database
- Create a Block Change Tracking file
  - Increases backup performance
- Modify tables to be NOLOGGING
- Perform INSERT /*+ APPEND */ statements as part of the ETL
- Perform an incremental backup upon completion of the ETL process
  - Incremental backups affect just the changed blocks
  - Can be run throughout the day
  - Effective with mini-batch scenarios
Conclusions

• All ETL mappings and flows, plus maintenance processes, should use Resumable Space Allocation
• Flashback provides “data management restartability”
• Still need “code-controlled restartability”
• Flashback Table doesn’t require operations involvement, while Flashback Database does
• Keep your database in ARCHIVELOG Mode
• Use ONLINE Backups for the BI/DW environment
• Execute incremental backups at the completion of the ETL processing
Three R’s of Data Warehouse Fault Tolerance

Stewart Bryson, Managing Director, Rittman Mead America
Oracle Open World 2010, San Francisco, September 2010