Handling Human Errors

Human errors and logical data corruptions they cause are one of the most often faced types of failures in any database environment. Handling them is not always a trivial task as required procedure depends on many factors like: current database configuration, scope of data corruption, type of destructive action or time passed after the mistake was made.

This document describes tools useful for handling human errors, classifies human errors and proposes recovery procedures that can be used to fix logical data corruption caused by them.

Tools

Log Miner

Log Miner is a piece of functionality of Oracle database that can be extremely useful for handling human errors. It can be especially helpful for determining the exact point in time when the error has occurred and therefore it can improve recovery accuracy.

Oracle Log Miner analyzes one or more redo and archived redo logs and provides a SQL interface to query their contents. These contents can be used not only to determine the exact point in time when someone made a mistake but also in some cases to undo unwanted changes.

Log Miner is available through the `DBMS_LOGMNR` PL/SQL package existing in the SYS schema.

The main problem about the Log Miner is that it can be useful for human errors handling only if supplemental logging (at least on minimum level) has been enabled on the affected database prior to the error. The supplemental logging is not enabled by default in a newly created database but it is required by Streams so all the databases taking advantage of the Streams technology have the supplemental logging configured. To verify if a given database uses supplemental logging one can query the `v$database` view:

```
sqlsys_DB
SQL> SELECT SUPPLEMENTAL_LOG_DATA_MIN FROM V$DATABASE;
```

If the result is 'YES' or 'IMPLICIT' then supplemental logging is enabled and Log Miner can be used.

Flashback Query/Table

Flashback Query allows querying table's contents as it was in the specified moment in the past. To produce this 'past-image' of the table Flashback Query relies on undo data. Therefore the point in time up to which one can go back with Flashback Query is strictly related to undo tablespace size and undo retention. Usually this time window does not exceed several hours. Flashback queries are very similar to normal queries. The only difference is an extra clause specifying time or SCN. E.g:

```
SELECT * FROM schema.table_name AS OF TIMESTAMP TO_TIMESTAMP('2007-12-01 09:40:00','YYYY-MM-DD HH
SELECT * FROM schema.table_name AS OF SCN 7515652;
```

Flashback Table feature takes advantage of Flashback Query functionality to moving back in time a whole table preserving dependencies, e.g:

```
FLASHBACK TABLE schema.table_name TO TIMESTAMP TO_TIMESTAMP('2007-12-01 09:40:00','YYYY-MM-DD HH:
FLASHBACK TABLE schema.table_name TO SCN 751652;
```

Unfortunately Flashback Table works only for tables that has row movement enabled. Flashing a table back to a point in time before row movement has been enabled is not possible.

Undrop table

Allows 'rolling back' effects of DROP TABLE statements. FLASHBACK TABLE schema_name.table_name TO BEFORE DROP; This feature works only if recyclebin was enabled before the drop statement was executed.

Flash backups and point in time recovery

In case any of the tools described above is not applicable one can use on-disk backups stored in the Flash Recovery Area and point-in-time recovery. On-disk backup checkpoint lags behind the database by 1 or 2 days depending on the database so if human error happened during this window they can be used for the recovery. Recovery procedure is described here (Recovery scenario 3:Point in time recovery using flash backups section)

Automatic recovery script

When the human error happened too long ago to use flash backups, on tape backups are the last resort. As long as the error happened during the period covered by on-tape backup retention policy (usually 31 days of recovery window) automatic recovery script described here here can be used to restore and recover the database to a desired point in time. Once the database is recovered the Data Pump tool can be used to copy over logically corrupted piece of data to the affected database.

Procedures

Faulty DML statements

This procedure describes steps necessary to recover from a logical corruption caused by a fault DML statement such as: INSERT, UPDATE, DELETE or MERGE.

Required information:

- Type of statement executed
- Affected database, schema and table(s)
- Name of the user who executed the fault DML
- Error time
- Whether the schema can be locked

Procedure:

1. If possible lock the affected schema (together with associated reader and writer schemas) and kill all the sessions connecting to it. Having a quiescent schema helps during recovery as it ensures that further changes are not being introduced by users and eliminates locking problems.

```
sqlsys_DB
SQL> alter user ... account lock; -- repeat for reader/writer schemas
SQL> select inst_id, sid, serial# from gv$session where username like '...%';
SQL> -- for each selected row execute the following kill session statement:
SQL> alter system kill session 'sid,serial#' immediate;
```

2. Check if the affected database uses supplemental logging and if yes, use Log Miner to determine the SCN of the unwanted DML statement:

```
sqlsys_DB    # It is much more convenient to use benthic or other GUI instead
SQL> SELECT SUPPLEMENTAL_LOG_DATA_MIN FROM V$DATABASE; -- If YES or IMPLICIT then Log Mine
SQL> -- Start log mining, choose the start time which is before error time
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SQL> EXECUTE SYS.DBMS_LOGMNR.START_LOGMNR(STARTTIME=>'26-Jan-2008 15:30:00', ENDTIME => sy
SQL> -- Query the data modifying appropriately the WHERE clause:
SQL> select SCN, TIMESTAMP, SEG_OWNER, SEG_NAME, TABLE_NAME, USERNAME, OPERATION, SQL_REDO
SQL> -- Finish log mining:
SQL> EXECUTE SYS.DBMS_LOGMNR.end_LOGMNR;
```

3. Find out if the damaged table depends on other tables or if other tables depend on it. Determine if those other tables need recovery.
4. Recover table:
    1. If the affected table (and related tables if relevant) was not modified after the human error and if Log Miner could be used one can take advantage of UNDO_SQL generated by Log Miner to fix the problem.
    2. If the affected table (and related tables if relevant) has row movement enabled, try to use flashback table feature:

    ```
    SQL> FLASHBACK TABLE schema.table_name TO SCN ...;   # use SCN retrieved with Log Mi
    or
    SQL> FLASHBACK TABLE schema.table_name TO TIMESTAMP TO_TIMESTAMP('...','YYYY-MM-DD H
    ```

    3. Otherwise try try to use flashback query:

    ```
    SQL> CREATE TABLE temp_name_date AS SELECT * FROM schema.table_name AS OF SCN ...; #
    or
    SQL> CREATE TABLE temp_name_date AS SELECT * FROM schema.table_name AS OF TIMESTAMP
    SQL> -- overwrite data from original corrupted table(s) with recovered data
    ```

    4. If none of the above works try recovery using flash backup + Data Pump as described here. Use SCN determined with Log Miner if relevant.
    5. If flash backup can't be used, use point-in-time recovery using on-tape backups and test recovery script as described here. Use time determined with Log Miner if relevant. Import recovered data with Data Pump.
5. Cleanup:
    1. drop any temporary objects that have been created
    2. unlock schema(s)

Truncate table

If a table or set of tables has been truncated then the only recovery option is point in time recovery either using on-disk (flash) or on-tape backup.

Required information:

- Affected database, schema and table(s)
- Name of the user who executed the statement
- Error time
- Whether the schema can be locked

Procedure:

1. If possible lock the affected schema (together with associated reader and writer schemas) and kill all the sessions connecting to it. Having a quiescent schema helps during recovery as it ensures that further changes are not being introduced by users and eliminates locking problems.

```
sqlsys_DB
SQL> alter user ... account lock; -- repeat for reader/writer schemas
SQL> select inst_id, sid, serial# from gv$session where username like '...%';
SQL> -- for each selected row execute the following kill session statement:
SQL> alter system kill session 'sid,serial#' immediate;
```

2. Check if the affected database uses supplemental logging and if yes, use Log Miner to determine the SCN of the TRUNCATE statement:

```
sqlsys_DB    # It is much more convenient to use benthic or other GUI instead
SQL> SELECT SUPPLEMENTAL_LOG_DATA_MIN FROM V$DATABASE; -- If YES or IMPLICIT then Log Mine
SQL> -- Start log mining, choose the start time which is before error time
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SQL> EXECUTE SYS.DBMS_LOGMNR.START_LOGMNR(STARTTIME=>'26-Jan-2008 15:30:00', ENDTIME => sy
SQL> -- Query the data modifying appropriately the WHERE clause:
SQL> select SCN, TIMESTAMP, SEG_OWNER, SEG_NAME, TABLE_NAME, USERNAME, OPERATION, SQL_REDO
SQL> -- Look for row containing truncate statement in the SQL_REDO column
SQL> -- Finish log mining:
SQL> EXECUTE SYS.DBMS_LOGMNR.end_LOGMNR;
```

3. If possible use on-disk backup, point-in-time recovery and Data Pump to restore truncated data. Point-in-time recovery using flash backup is described here. Use SCN determined with Log Miner if relevant.
4. If for some reason on-disk backup cannot be used use on-tape one, automatic recovery script and Data Pump. Automatic recovery script details can be found here. Use time determined with Log Miner if relevant.
5. Cleanup:
    1. unlock the schema(s)

Drop table

Required information:

- Affected database, schema and table(s)
- Name of the user who executed the statement
- Error time

Procedure:

1. Check if recyclebin is in use. If yes, check if the dropped table is still in there. In case the table is still being kept in the recyclebin un-drop it:

```
sqlsys_DB
SQL> -- to check if recycle bin is in use and the table is still in there:
SQL> select owner, object_name, original_name, droptime, can_undrop from dba_recyclebin wh
SQL> -- if the query returns any rows and if the the CAN_UNDROP column contains 'YES', the
SQL> flashback table schema_name.table_name to before drop;
```

2. If recyclebin can't be used then if possible determine the failure SCN using Log Miner.

```
sqlsys_DB    # It is much more convenient to use benthic or other GUI instead
SQL> SELECT SUPPLEMENTAL_LOG_DATA_MIN FROM V$DATABASE; -- If YES or IMPLICIT then Log Mine
SQL> -- Start log mining, choose the start time which is before error time
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SQL> EXECUTE SYS.DBMS_LOGMNR.START_LOGMNR(STARTTIME=>'26-Jan-2008 15:30:00', ENDTIME => sy
SQL> -- Query the data modifying appropriately the WHERE clause:
SQL> select SCN, TIMESTAMP, SEG_OWNER, SEG_NAME, TABLE_NAME, USERNAME, OPERATION, SQL_REDO
SQL> -- Look for row containing drop table statement in the SQL_REDO column
SQL> -- Finish log mining:
SQL> EXECUTE SYS.DBMS_LOGMNR.end_LOGMNR;
```

3. Find out if the damaged table depends on other tables or if other tables depend on it. Determine if those other tables need recovery.
4. If possible use on-disk backup, point-in-time recovery and Data Pump to restore dropped table and tables related to it. Point-in-time recovery using flash backup is described here. Use SCN determined

with Log Miner if relevant.
5. If for some reason on-disk backup cannot be used use on-tape one, automatic recovery script and Data Pump. Automatic recovery script details can be found here. Use time determined with Log Miner if relevant.

Other DDL statements

Required information:

- Type of DDL statement
- Affected database, schema and table(s)
- Name of the user who executed the statement
- Error time
- Whether the schema can be locked

Procedure:

1. If possible lock the affected schema (together with associated reader and writer schemas) and kill all the sessions connecting to it. Having a quiescent schema helps during recovery as it ensures that further changes are not being introduced by users and eliminates locking problems.

```
sqlsys_DB
SQL> alter user ... account lock; -- repeat for reader/writer schemas
SQL> select inst_id, sid, serial# from gv$session where username like '...%';
SQL> -- for each selected row execute the following kill session statement:
SQL> alter system kill session 'sid,serial#' immediate;
```

2. If possible determine the SCN of unwanted DDL using Log Miner.

```
sqlsys_DB    # It is much more convenient to use benthic or other GUI instead
SQL> SELECT SUPPLEMENTAL_LOG_DATA_MIN FROM V$DATABASE; -- If YES or IMPLICIT then Log Mine
SQL> -- Start log mining, choose the start time which is before error time
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SQL> EXECUTE SYS.DBMS_LOGMNR.START_LOGMNR(STARTTIME=>'26-Jan-2008 15:30:00', ENDTIME => sy
SQL> -- Query the data modifying appropriately the WHERE clause:
SQL> select SCN, TIMESTAMP, SEG_OWNER, SEG_NAME, TABLE_NAME, USERNAME, OPERATION, SQL_REDO
SQL> -- Look for rows containing faulty DDL statement in the SQL_REDO column
SQL> -- Finish log mining:
SQL> EXECUTE SYS.DBMS_LOGMNR.end_LOGMNR;
```

3. Find out if the damaged table depends on other tables or if other tables depend on it. Determine if those other tables need recovery. Drop or rename table(s) that will be recovered.
4. If possible use on-disk backup, point-in-time recovery and Data Pump to restore altered table and tables related to it. Point-in-time recovery using flash backup is described here. Use SCN determined with Log Miner if relevant.
5. If for some reason on-disk backup cannot be used use on-tape one, automatic recovery script and Data Pump. Automatic recovery script details can be found here. Use time determined with Log Miner if relevant.
6. Cleanup:
    1. unlock the schema(s)

This topic: PSSGroup > HandlingHumanErrors
Topic revision: r4 - 2008-02-10 - JacekWojcieszuk