Oracle GoldenGate - Sequences

This page describes Oracle GoldenGate support for sequences.

The page is based on Oracle GoldenGate version 11.2.1.0.1. For a basic configuration I used two Linux VMs (OEL5U6) running single instance Oracle 11.2.0.3 databases. I created both databases using DBCA.

This configuration uses the following hosts and databases:

	Source	Target
Hostname	vm4	vm5
Database Name	NORTH	SOUTH

The configuration includes the following on both nodes:

- Creation of a GoldenGate schema owner called GG01.
- Specification of GGSCHEMA as GG01 in GoldenGate parameters
- Creation of GOLDENGATE tablespace which is default tablespace for GG01

The GoldenGate process names are:

	Source	Target
Extract	ex1	
Data Pump	dp1	
Replicat		rep1

Sequence Setup script

In each database run the sequence setup script. This script creates four procedures for use by GoldenGate processes. Specify the GoldenGate schema name (in this case GG01) when prompted:

```
[oracle@vm4]$ cd /home/oracle/goldengate

[oracle@vm4]$ sqlplus / as sysdba

SQL> @sequence
Please enter the name of a schema for the GoldenGate database objects:
GG01
Setting schema name to GG01

UPDATE_SEQUENCE STATUS:

Line/pos   Error
---------- -------------------------------------------------------------------
No errors  No errors

GETSEQFLUSH

Line/pos   Error
---------- -------------------------------------------------------------------
No errors  No errors

SEQTRACE

Line/pos   Error
---------- -------------------------------------------------------------------
No errors  No errors

REPLICATE_SEQUENCE STATUS:

Line/pos   Error
```

```
---------- ------------------------------------------------------------------
No errors  No errors

STATUS OF SEQUENCE SUPPORT
------------------------------------------------------------
SUCCESSFUL installation of Oracle Sequence Replication support
SQL>
```

The sequence.sql script creates the following procedures in the GG01 schema:

- SEQTRACE
- GETSEQFLUSH
- REPLICATESEQUENCE
- UPDATESEQUENCE

SEQTRACE

This procedure has the following definition:

```
CREATE OR REPLACE PROCEDURE seqTrace
(
   traceStmt IN VARCHAR2,
   traceUser IN VARCHAR2
)
```

SEQTRACE outputs trace to trace file. In SQL*Plus, if SERVEROUTPUT is enabled, it also displays code to trace.

Parameters are:

Parameter Name	Description
TRACESTMT	SQL statement to be executed and optionally to be traced
TRACEUSER	Name of schema performing trace

In SQL*Plus, if SERVEROUTPUT is enabled then this procedure outputs TRACESTMT using the DBMS_OUTPUT.PUT_LINE procedure.

It then executes TRACESTMT using the PL/SQL EXECUTE IMMEDIATE statemernt.

SEQFLUSH

This procedure has the following definition:

```
CREATE OR REPLACE PROCEDURE getSeqFlush
(
   seqName IN VARCHAR2,
   isCycle IN NUMBER,
   flushStmt OUT VARCHAR2,
   isTrace IN NUMBER,
   traceUser IN VARCHAR2
)
```

SEQFLUSH creates a DDL statement to flush a sequence.

Parameters are:

Parameter Name	Description
SEQNAME	Name of sequence
ISCYCLE	Set to 1 if sequence is cyclic

FLUSHSTMT	The ALTER SEQUENCE statement generated by the procedure
ISTRACE	Set to 1 if trace is enabled. For tracing Oracle DDL replication must be installed, but not enabled
TRACEUSER	Name of the tracing user

The procedure uses the ALTER SEQUENCE table to modify the sequence (if necessary) to ensure that the next call to NEXTVAL will increase the high water mark for the sequence.

The procedure is only called if the DBOPTIONS is set to _AUTOMATICSEQUENCEFLUSH; Flushing is disabled by default.

REPLICATESEQUENCE

This procedure has the following definition:

```
CREATE OR REPLACE PROCEDURE replicateSequence
(
   sourceHWM IN  NUMBER,
   maxDistance  IN NUMBER,
   seqFlush IN  NUMBER,
   seqOwner IN  VARCHAR2,
   seqUpdate IN  NUMBER,
   seqName IN  VARCHAR2,
   loggedUser IN  VARCHAR2,
   racTarget IN  NUMBER,
   isTrace IN  NUMBER,
   traceUser IN  VARCHAR2
)
```

REPLICATESEQUENCE replicates a source sequence update to the target database. The procedure is called from REPLICAT. It can also be called directly by Oracle technical support for debugging purposes.

Parameters are:

Parameter Name	Description
SOURCEHWM	Source high water mark
MAXDISTANCE	Maximum percentage of sequence space (ahead) allocated for target database. Defaults to the value of the DBOPTIONS _MAXSEQUENCEDISTANCE parameter. If trace is 1, then DDLOPTIONS _TRACESEQUENCE is used
SEQFLUSH	If 1, sequence each time will be flushed each time this procedure is called i.e. next call to NEXTVAL will increase HWM. AUTOMATICSEQUENCEFLUSH is disabled by default, but is automatically enabled for RAC databases to ensure all instances of RAC have same HWM.
SEQOWNER	Owner of sequence
SEQUPDATE	Maximum update on source database (cache size * increment * number of RAC instances)
SEQNAME	Name of sequence
LOGGEDUSER	Name of user logged on
RACTARGET	Number of RAC instances in target database
ISTRACE	Set to 1 if trace is enabled. For tracing Oracle DDL replication must be installed, but not enabled
TRACEUSER	Name of trace user. This will normally be the GoldenGate schema

This procedure is executed by the REPLICAT each time it encounters a sequence record in the GoldenGate trail. The sequence record is generated when the extract encounters an update to the SEQ$ table in the data dictionary. This occurs when a new set of sequence numbers is obtained for the instance.

UPDATESEQUENCE

This procedure has the following definition:

```
CREATE OR REPLACE PROCEDURE updateSequence
```

```
(
    schemaName IN VARCHAR2,
    seqName IN VARCHAR2,
    isTrace IN NUMBER,
    traceUser IN VARCHAR2,
    sessUser IN VARCHAR2
)
```

UPDATESEQUENCE updates all sequences in a specified schema so that HWM is increased by a minimal amount.

Parameters are:

Parameter Name	Description
SCHEMANAME	Name of schema for which sequences will be updated
SEQNAME	Name of sequence to be updated. If this value is NULL, the HWM will be updated for all sequences specified by SCHEMANAME
ISTRACE	Set to 1 if trace is enabled. For tracing Oracle DDL replication must be installed, but not enabled
TRACEUSER	Username of trace user
SESSUSER	Username of calling session

Sequence DDL

Sequences are managed in the database using the following DDL statements.

To create a sequence use:

```
CREATE SEQUENCE [<schema_name>.]<sequence_name>
  [ INCREMENT BY  ]
  [ START WITH  ]
  [ MAXVALUE  | NOMAXVALUE ]
  [ MINVALUE  | NOMINVALUE ]
  [ CYCLE| NOCYCLE ]
  [ CACHE  | NOCACHE ]
  [ ORDER | NOORDER ]
```

Default values are

- INCREMENT BY 1
- START WITH 1 (or MINVALUE if specified)
- NOMINVALUE
- NOMAXVALUE
- NOCYCLE
- CACHE 20
- NOORDER

To alter a sequence use:

```
ALTER SEQUENCE [<schema_name>.]<sequence_name>
  [ INCREMENT BY  ]
  [ MAXVALUE  | NOMAXVALUE ]
  [ MINVALUE  | NOMINVALUE ]
  [ CYCLE| NOCYCLE ]
  [ CACHE  | NOCACHE ]
  [ ORDER | NOORDER ]
```

To drop a sequence use:

```
DROP SEQUENCE [<schema_name>.]<sequence_name>
```

Data Dictionary

SEQ$

Within the data dictionary sequence metadata is stored in the SYS.SEQ$ table. This table contains one row for each sequence.

The SEQ$ table has the following columns:

Column Name	Data Type
OBJ#	NUMBER
INCREMENT#	NUMBER
MINVALUE	NUMBER
MAXVALUE	NUMBER
CYCLE#	NUMBER
ORDER#	NUMBER
CACHE	NUMBER
HIGHWATER	NUMBER
AUDIT$	VARCHAR2(38)
FLAGS	NUMBER

DBA_SEQUENCES

Sequence metadata is externalized by the DBA_SEQUENCES data dictionary view which contains the following columns:

Column Name	Data Type
SEQUENCE_OWNER	VARCHAR2(30)
SEQUENCE_NAME	VARCHAR2(30)
MIN_VALUE	NUMBER
MAX_VALUE	NUMBER
INCREMENT_BY	NUMBER
CYCLE_FLAG	VARCHAR2(1)
ORDER_FLAG	VARCHAR2(1)
CACHE_SIZE	NUMBER
LAST_NUMBER	NUMBER

V$_SEQUENCES

Sequence objects in the library cache is externalized by the V$_SEQUENCES dynamic performance view. This view is based on X$KGLOB and has the following columns:

Column Name	Data Type
SEQUENCE_OWNER	VARCHAR2(64)
SEQUENCE_NAME	VARCHAR2(1000)
OBJECT#	NUMBER
ACTIVE_FLAG	VARCHAR2(1)
REPLENISH_FLAG	VARCHAR2(1)
WRAP_FLAG	VARCHAR2(1)
NEXTVALUE	NUMBER
MIN_VALUE	NUMBER
MAX_VALUE	NUMBER
INCREMENT_BY	NUMBER
CYCLE_FLAG	VARCHAR2(1)

ORDER_FLAG	VARCHAR2(1)
CACHE_SIZE	NUMBER
HIGHWATER	NUMBER
BACKGROUND_INSTANCE_LOCK	VARCHAR2(1)
INSTANCE_LOCK_FLAGS	NUMBER

Examples

This page uses the following example sequences to demonstrate GoldenGate operation. The sequences were created by the US03 user as follows:

```
CREATE SEQUENCE seq1;

CREATE SEQUENCE seq2 NOCACHE;

CREATE SEQUENCE seq3 CACHE 5 MAXVALUE 20 CYCLE;
```

Each sequence must be created in both the source and target databases. For any given sequence all attributes must be identical e.g. cache size, cycle, minimum and maximum values etc.

The above objects have the following properties:

Sequence	Cache Size	Cache	Cycle	Max Value
SEQ1	20	CACHE	NOCYCLE	Undefined
SEQ2	1	NOCACHE	NOCYCLE	Undefined
SEQ3	5	CACHE	CYCLE	20

In all cases the first value returned by the sequence will be 1.

Sequence numbers are generated by specifying the NEXTVAL pseudo-column for the sequence object. For example:

```
SELECT seq1.NEXTVAL FROM dual;
```

The current value for a sequence can be obtained using the CURRVAL pseudo-column as follows:

```
SELECT seq1.CURRVAL FROM dual;
```

Note that the CURRVAL pseudo-column is only available after the sequence has been incremented within the current session. If the sequence has not been incremented by the current session then an Oracle error will be returned. For example:

```
ORA-08002: sequence SEQ1.CURRVAL is not yet defined in this session
```

Each sequence has a highwater mark. This is the next value that will be allocated by the sequence. In the SEQ$ data dictionary table this value is stored in the HIGHWATER column.

For a non-caching sequence, the row for that sequence in SEQ$ will be updated every time a new value is generated for that sequence. This can lead to block-level contention on highly concurrent systems and causes inter-instance contention in RAC environments.

To reduce contention, sequence values are cached at instance-level. By default the cache size is 20. The first time a sequence is accessed by an instance, the high-water mark in SEQ$ will be incremented by the cache size and the first value will be returned. Subsequent sequence numbers are allocated directly from the SGA until the cache is exhausted thereby avoiding additional DML on the SEQ$ table.

In a RAC database, each instance maintains a separate set of sequence numbers within the local SGA. For example in a database with three RAC instances, instance A might issue sequence numbers 1-20, instance B

sequence numbers 21-40 and instance C sequence numbers 41-60. If instance B exhausts its cache, then it may reserve the next set of sequence numbers i.e. 61-80.

The first time a sequence is accessed by a session within the instance, it is loaded into the library cache. Sequences are not pinned by default in the library cache and can therefore be aged out at any time.

If the sequence object is aged out of the SGA or the instance is shutdown, any unallocated numbers currently in the sequence cache will be lost. Therefore it is not possible to guarantee that every sequence number will be allocated. Even if the NOCACHE option is specified sequence numbers can be lost due to transaction rollbacks.

GoldenGate Parameters

Sequences are configured for extract processes using the SEQUENCE parameter.

Example for extract process ex1:

```
EXTRACT ex1
USERID gg01, PASSWORD gg01
EXTTRAIL /home/oracle/goldengate/dirdat/ex
SEQUENCE us03.*;
TABLE us03.*;
```

Example for extract process dp1:

```
EXTRACT dp1
USERID gg01, PASSWORD gg01
RMTHOST vm5, MGRPORT 7809
RMTTRAIL /home/oracle/goldengate/dirdat/rt
SEQUENCE us03.*;
TABLE us03.*;
```

For replicat processes, sequences are configured using the MAP/TARGET parameters.

Example for replicat process rep1:

```
REPLICAT rep1
USERID gg01, PASSWORD gg01
ASSUMETARGETDEFS
DISCARDFILE /home/oracle/goldengate/discards, PURGE
MAP US03.SEQ1, TARGET US03.SEQ1;
MAP US03.SEQ2, TARGET US03.SEQ2;
MAP US03.SEQ3, TARGET US03.SEQ3;
MAP US03.T*, TARGET US03.*;
```

It is not actually necessary to specify sequence names explicitly if all sequences are to be replicated. In the above example, the mappings for the tables and sequences can be combined. For example:

```
REPLICAT rep1
USERID gg01, PASSWORD gg01
ASSUMETARGETDEFS
DISCARDFILE /home/oracle/goldengate/discards, PURGE
MAP US03.*, TARGET US03.*;
```

GoldenGate Trail

An entry is created in the GoldenGate trail every time the SEQ$ table is updated for a sequence that is being extracted.

- If the NOCACHE option has been specified for a sequence then an entry will appear in the GoldenGate trail every time the sequence is incremented.

- If the CACHE option has been specified for the sequence then an entry will appear in the GoldenGate trail every time the high water mark is increased.

The following is an example of TrailAnalyzer output for a cached sequence (US03.SEQ2):

```
# Header: Type=47 (G) Flag=1 Len=156 (4 bytes)

  # Body
  # Header: Type=48 (H) Flag=0 Len=44 (4 bytes)
  <47_48>
    # Row Header (44 bytes)
    # Flags - 0x84 Sequence
    # Timestamp 2013:04:04 17:06:57 000000 ms
    # Object US03.SEQ1

  # Header: Type=44 (D) Flag=0 Len=44 (4 bytes)
  <47_44>
    # Row Data (44 bytes)
      # Slot 0 (8 bytes)
      00 00 00 04 55 53 30 33         # Sequence Owner = US03
      # Slot 1 (8 bytes)
      00 00 00 04 53 45 51 31         # Sequence Name = SEQ1
      # Slot 2 (6 bytes)
      00 00 00 02 34 31               # High Water Mark = 41
      # Slot 3 (6 bytes)
      00 00 00 02 32 30               # Cache Size = 20

  # Header: Type=54 (T) Flag=0 Len=48 (4 bytes)
  <47_54>
    # Row Metadata (48 bytes)
    # Header: Type=52 (R) Flag=0 Len=20 (4 bytes)
    <47_54_52>
      # ROWID: 20 bytes
      AAAABKAABAAAAKbAAj 1

    # Header: Type=4C (L) Flag=0 Len=7 (4 bytes)
    <47_54_4C>
      # Commit SCN (7 bytes)
      1779289

    # Header: Type=36 (6) Flag=0 Len=9 (4 bytes)
    <47_54_36>
      # Transaction ID (9 bytes)
      8.21.1084

# Header: Type=5A (Z) Flag=1 Len=156 (4 bytes)
```

In the above example the cache size is 20. When current sequence value has reached 20, a new range of sequence values was issued (21-40) and the high water mark was advanced to 41. This value was updated in the HIGHWATER column of the SEQ$ table.

The UPDATE statement is captured from the online redo log by GoldenGate and stored in the trail as shown above.

The following is an example of TrailAnalyzer output for a non-cached sequence (US03.SEQ2):

```
# Header: Type=47 (G) Flag=1 Len=153 (4 bytes)

  # Body
  # Header: Type=48 (H) Flag=0 Len=44 (4 bytes)
  <47_48>
    # Row Header (44 bytes)
    # Flags - 0x84 Sequence
    # Timestamp 2013:04:04 16:38:42 000000 ms
    # Object US03.SEQ2
```

```
  # Header: Type=44 (D) Flag=0 Len=43 (4 bytes)
  <47_44>
    # Row Data (43 bytes)
      # Slot 0 (8 bytes)
      00 00 00 04 55 53 30 33         # Sequence Owner = US03
      # Slot 1 (8 bytes)
      00 00 00 04 53 45 51 32         # Sequence Name = SEQ2
      # Slot 2 (6 bytes)
      00 00 00 02 31 31               # High Water Mark = 11
      # Slot 3 (5 bytes)
      00 00 00 01 30                  # Cache Size = 0

  # Header: Type=54 (T) Flag=0 Len=46 (4 bytes)
  <47_54>
    # Row Metadata (46 bytes)
    # Header: Type=52 (R) Flag=0 Len=20 (4 bytes)
    <47_54_52>
      # ROWID: 20 bytes
      AAAABKAABAAAAKbAAk 1

    # Header: Type=4C (L) Flag=0 Len=7 (4 bytes)
    <47_54_4C>
      # Commit SCN (7 bytes)
      1778485

    # Header: Type=36 (6) Flag=0 Len=7 (4 bytes)
    <47_54_36>
      # Transaction ID (7 bytes)
      4.2.914

# Header: Type=5A (Z) Flag=1 Len=153 (4 bytes)
```

When the NOCACHE option is specified a sequence creates a recursive transaction to update SEQ$ each time a new value for the sequence is generated during the NEXTVAL pseudo-column.

GoldenGate Replication

If sequence number updates have been captured by the extract process and propagated by the data pump, they will be replicated by the replicat process if included in a MAP parameter.

To verify this behaviour, we can enable 10046 trace for the replicat process. When the replicat process is running, it creates an Oracle server process. The SID of this server process can be determined as follows:

```
SELECT sid,serial#,module FROM v$session
WHERE module LIKE 'OGG%';

       SID    SERIAL# MODULE
---------- ---------- ------------------------------
        48       1123 OGG-REP1-OPEN_DATA_SOURCE
```

Trace can be enabled for the server process using the SESSION_TRACE_ENABLE procedure in the DBMS_MONITOR package:

```
EXECUTE dbms_monitor.session_trace_enable (48,1123,binds=>TRUE);
```

Increment a sequence on the source database. In this case the non-caching sequence SEQ2 has been incremented to ensure that SEQ$ is updated and that the change is propagated to the target database.

```
SQL> SELECT seq2.NEXTVAL FROM dual;

   NEXTVAL
----------
        11
```

The OBJECT_ID for SEQ2 can be determined from DBA_OBJECTS as follows:

```
SQL> SELECT object_id FROM dba_objects
  2  WHERE owner = 'US03'
  3  AND object_name = 'SEQ2';

 OBJECT_ID
----------
     76757
```

Note that the DATA_OBJECT_ID coluimn in DBA_OBJECTS is NULL for sequences.

```
SQL> SELECT highwater FROM sys.seq$
  2  WHERE obj# = 76757;

 HIGHWATER
----------
        12
```

In the target database determine the trace file name for the replicat process:

```
SELECT p.tracefile
FROM v$session s,v$process p
WHERE s.paddr = p.addr
AND s.sid = 48
AND s.serial# = 1123;

TRACEFILE
------------------------------------------------------------
/u01/app/oracle/diag/rdbms/south/SOUTH/trace/SOUTH_ora_1580.trc
```

The following statements are executed in the target database following the detection of a change to SEQ$ in the propagated GoldenGate trail.

The replicat process calls the REPLICATESEQUENCE procedure. Note that it uses literal values instead of bind variables. Therefore this PL/SQL block must be hard-parsed every time it is invoked.

```
BEGIN gg01 .replicateSequence (TO_NUMBER(12), TO_NUMBER(20), TO_NUMBER(1), 'US03',
TO_NUMBER(0),
'SEQ2', UPPER('gg01'), TO_NUMBER(1), TO_NUMBER (0), ''); END;
```

REPLICATESEQUENCE calls the following recursive SQL:

The first statement sets the current user:

```
ALTER SESSION SET CURRENT_SCHEMA="US03"
```

The next statement determines the object number and other properties for the sequence:

```
SELECT S.HIGHWATER, S.MINVALUE, S.MAXVALUE, S.INCREMENT$, S.CYCLE# , O.OBJ#,
S.CACHE
FROM SYS.SEQ$ S, SYS.OBJ$ O, SYS.USER$ U
WHERE U.NAME=:B2
AND O.OWNER#=U.USER#
AND O.NAME=:B1
AND S.OBJ#=O.OBJ#
```

Bind variables are:

#	Name	Type	Value
0	B2	VARCHAR2	US03

1	B1	VARCHAR2	SEQ3

The next statement determines the high water mark for the sequence object.

```
SELECT HIGHWATER FROM SYS.SEQ$ WHERE OBJ#=:B1
```

Bind variables are:

#	Name	Type	Value
0	B1	NUMBER	76583

In the above example, 76583 is the OBJECT_ID of SEQ2.

The next statement uses the NEXTVAL pseudo-column to generate the next value for SEQ2.

```
SELECT "SEQ2".NEXTVAL FROM DUAL
```

The above statement calls three following statements in a recursive transaction:

The first recursive statement selects all column values from the row in SEQ$ for the specified sequence

```
select increment$,minvalue,maxvalue,cycle#,order$,cache,highwater,audit$,flags
from seq$
where obj#=:1
```

#	Name	Type	Value
0	OBJ#	NUMBER	76583

The second recursive statement checks that the user has privileges to increment the sequence:

```
select grantee#,privilege#,nvl(col#,0),max(mod(nvl(option$,0),2))
from objauth$ where obj#=:1
group by grantee#,privilege#,nvl(col#,0)
order by grantee#
```

#	Name	Type	Value
0	OBJ#	NUMBER	76583

The third and final recursive statement updates the row in the SEQ$ table for the specified sequence.

```
update seq$ set
increment$=:2,minvalue=:3,maxvalue=:4,cycle#=:5,order$=:6,cache=:7,highwater=:8,aud
it$=:9,flags=:10 where obj#=:1
```

#	Name	Type	Value
0	INCREMENT$	NUMBER	1
1	MINVALUE	NUMBER	1
2	MAXVALUE	NUMBER	999999999999999999999999999
3	CYCLE#	NUMBER	0
4	ORDER$	NUMBER	0
5	CACHE	NUMBER	0
6	HIGHWATER	NUMBER	12
7	AUDIT$	VARCHAR2	--------------------------------
8	FLAGS	NUMBER	8
9	OBJ#	NUMBER	76583

The next statement generates the DDL statement to set the cyclic properties for the sequence:

```
SELECT 'ALTER SEQUENCE "' || :B2 || '" ' || DECODE (:B1 , 1, 'CYCLE', ' NOCYCLE  /*
GOLDENGATE_DDL_REPLICATION */') FROM DUAL
```

#	Name	Type	Value
0	B2	VARCHAR2	SEQ2
1	B1	NUMBER	0

The next step is to execute the statement generated in the previous step:

```
ALTER SEQUENCE "SEQ2"  NOCYCLE  /* GOLDENGATE_DDL_REPLICATION */
```

Finally the session is reset to the GGSCHEMA.

```
ALTER SESSION SET CURRENT_SCHEMA="GG01"
```

The REPLICATESEQUENCE procedure is always called after SEQ2 has been modified. The procedure contains a significant amount of conditional logic and therefore there will be some variation in the actual statements executed depending mainly on the attributes that have been specified for the sequence.

More important, however, is that the REPLICATESEQUENCE procedure requires a hard parse before each execution and makes a number of recursive calls to SQL procedures. Therefore the target database is performing significantly more work when maintaining sequences than the source database.

If relatively large cache values are specified for the sequence, then it should only be necessary to obtain new ranges of sequence values occasionally and the impact on the target database should be minimal. However, if the NOCACHE option is specified for the sequence, the cost of maintaining sequences on the target database is likely to exceed that on the source database.