Oracle GoldenGate - Supplemental Logging

Introduction

This page describes the implementation of supplemental logging in the Oracle database with particular emphasis on Oracle GoldenGate.

The research was performed in Oracle 11.2.0.3 on OEL 5.6 running in VirtualBox. The Oracle GoldenGate version was 11.2.1.0.1

Supplemental logging generates additional undo which is stored in the redo log. The additional information allows rows to be located when the ROWID is unavailable. Undo is used because we want to locate the row in the target database in order to apply the change. The change may update columns in the target database so we cannot use any values in the redo. This is required for logical standby databases for example.

The following definitions were derived from the Oracle Reference manual.

There are several types of supplemental logging:

- Minimal
- Primary Key
- Unique Key
- Foreign Key
- All
- Procedural Replication

Minimal Supplemental Logging

Minimal supplemental logging ensures that products leveraging LogMiner technology will have sufficient information to support chained rows and cluster tables.

Primary Key Supplemental Logging

Primary key supplemental logging includes the primary key for rows affected by UPDATE and DELETE changes.

Unique Key Supplemental Logging

Unique key supplemental logging includes all columns for a unique key are written to undo if any unique key columns are modified.

Foreign Key Supplemental Logging

Foreign key supplemental logging includes all other columns belonging to a foreign key will be logged in the undo if any foreign key columns are modified.

All Column Supplemental Logging

If no primary key or unique key is available then it is possible to specify that all columns are logged. In this case all columns in a row will be logged in the undo. When the row is replicated in the target database, equality predicates will be applied to all columns. LONG, LONG RAW and LOB columns will be omitted from the supplemental logging.

Procedural Replication Supplemental Logging

Procedural replication supplemental logging includes additional information in the redo log during invocation of procedures in Oracle-supplied packages for which procedural replication is supported. I have never investigated this option.

Supplemental Logging Levels

Oracle implements supplemental logging at database level, schema level and at table level. The three implementations are significantly different:

Database Level Supplemental Logging

Database level supplemental logging is configured in the control file. The parameter does not appear to be stored in the database itself.

Database level minimal supplemental logging can be enabled in SQL*Plus using:

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;

Database altered.
```

It is disabled again using:

```
SQL> ALTER DATABASE DROP SUPPLEMENTAL LOG DATA;

Database altered.
```

It does not appear to be possible to enable database level supplemental logging directly within GGSCI.

Example syntax:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (FOREIGN KEY) COLUMNS;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA FOR PROCEDURAL REPLICATION;
```

The current supplemental logging configuration is reported in V$DATABASE by the following columns:

- SUPPLEMENTAL_LOG_DATA_MIN
- SUPPLEMENTAL_LOG_DATA_PK
- SUPPLEMENTAL_LOG_DATA_UI
- SUPPLEMENTAL_LOG_DATA_FK
- SUPPLEMENTAL_LOG_DATA_ALL
- SUPPLEMENTAL_LOG_DATA_PL

Schema Level Supplemental Logging

Oracle does not implement schema level supplemental logging explicitly. It is enabled and/or disabled using the PREPARE_SCHEMA_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package.

This procedure takes two parameters:

Argument Name	Type	In/Out	Default
SCHEMA_NAME	VARCHAR2	IN	
SUPPLEMENTAL_LOGGING	VARCHAR2	IN	DEFAULT

The SUPPLEMENTAL_LOGGING parameter takes the following documented parameters:

- NONE - supplemental logging is not enabled for any columns in the schema. Existing supplemental logging specifications are not removed for any tables

- KEYS - supplemental logging is enabled for primary key, unique key, bitmap index and foreign key columns in tables in the schema and for any new tables created in this schema. Primary keys are logged unconditionally. Unique key, bitmap index and foreign keys are logged conditionally. Bitmap join index columns are not logged.

- ALL - supplemental logging is enabled for all columns in all existing tables in this schema and for any new tables created in this schema. The columns are logged unconditionally. Supplemental logging is not enabled for the following types of column:
    - LOB
    - LONG
    - LONG RAW
    - User-defined types
    - Oracle-supplied types

Schema level supplemental logging parameters are reported in the DBA_CAPTURE_PREPARED_SCHEMAS view.

This view has the following columns:

Column Name	Data Type
SCHEMA_NAME	VARCHAR2(30)
TIMESTAMP	DATE
SUPPLEMENTAL_LOG_DATA_PK	VARCHAR2(8)
SUPPLEMENTAL_LOG_DATA_UI	VARCHAR2(8)
SUPPLEMENTAL_LOG_DATA_FK	VARCHAR2(8)
SUPPLEMENTAL_LOG_DATA_ALL	VARCHAR2(8)

Possible values for the SUPPLEMENTAL_LOG_DATA flags are IMPLICIT EXPLICIT or NO

The above view has the following definition in DBA_VIEWS:

```
SELECT u.name, pd.timestamp,
       DECODE(BITAND(u.spare1, 1), 1,
              DECODE(BITAND(pd.flags, 1), 1, 'IMPLICIT', 'EXPLICIT'), 'NO'),
       DECODE(BITAND (u.spare1, 2), 2,
              DECODE(BITAND(pd.flags, 2), 2, 'IMPLICIT', 'EXPLICIT'), 'NO'),
       DECODE(BITAND (u.spare1, 4), 4,
              DECODE(BITAND(pd.flags, 4), 4, 'IMPLICIT', 'EXPLICIT'), 'NO'),
       DECODE(BITAND (u.spare1, 8), 8,
              DECODE(BITAND(pd.flags, 8), 8, 'IMPLICIT', 'EXPLICIT'), 'NO')
FROM streams$_prepare_ddl pd, user$ u
WHERE u.user# = pd.usrid
AND global_flag = 0
```

The flags correspond to the following bit values for SPARE1 in USER$

Column Name	Bit Value
SUPPLEMENTAL_LOG_DATA_PK	1
SUPPLEMENTAL_LOG_DATA_UI	2
SUPPLEMENTAL_LOG_DATA_FK	4
SUPPLEMENTAL_LOG_DATA_ALL	8

Two data dictionary tables support streams schema instantiation:

- STREAMS$_PREPARE_DDL
- STREAMS$_PREPARE_OBJECT

These tables are described below.

STREAMS$_PREPARE_DDL contains the following columns:

Column Name	Data Type
GLOBAL_FLAG	NUMBER
USRID	NUMBER
SCN	NUMBER
TIMESTAMP	DATE
FLAGS	NUMBER
SPARE1	NUMBER
SPARE2	VARCHAR2(1000)

STREAMS$_PREPARE_OBJECT contains the following columns:

Column Name	Data Type
OBJ#	NUMBER
CAP_TYPE	NUMBER
IGNORE_SCN	NUMBER
TIMESTAMP	DATE
FLAGS	NUMBER
SPARE1	NUMBER
SPARE2	VARCHAR2(1000)

KEYS

For example:

```
dbms_capture_adm.prepare_schema_instantiation ('US02','KEYS');
```

Setting the SUPPLEMENTAL_LOGGING parameter to KEYS has the following effect:

The above procedure sets the SPARE1 column of the USER$ table to 7 for user US02:

```
SQL> SELECT spare1 FROM sys.user$ WHERE name = 'US02';

SPARE1
------
     7
```

One row is added to STREAMS$_PREPARE_DDL for the user (in this case user# 87). The FLAGS column is set to 7.

```
SQL> SELECT global_flag,usrid,scn,timestamp,flags
FROM sys.streams$_prepare_ddl;

GLOBAL_FLAG      USRID        SCN TIMESTAMP      FLAGS
----------- ---------- ---------- --------- ----------
          0         87    1322469 17-MAR-13          7
```

One row is added to the STREAMS$_PREPARE_OBJECT table for each object owned by the user. In this example there are three tables in the US02 schema:

```
SQL> SELECT obj#,cap_type,ignore_scn,timestamp,flags
FROM sys.streams$_prepare_object;

      OBJ#   CAP_TYPE IGNORE_SCN TIMESTAMP      FLAGS
---------- ---------- ---------- --------- ----------
     76087          0    1322474 17-MAR-13          0
     76089          0    1322477 17-MAR-13          0
     76091          0    1322480 17-MAR-13          0
```

If a new table is created in the US02 schema, this object will automatically be added to the STREAMS$_PREPARE_OBJECTS table.

DBA_CAPTURE_PREPARED_SCHEMAS reports the following:

```
SQL> SELECT schema_name,
  supplemental_log_data_pk AS "PK",
  supplemental_log_data_pk AS "UI",
  supplemental_log_data_pk AS "FK",
  supplemental_log_data_pk AS "ALL"
FROM dba_capture_prepared_schemas;

SCHEMA_NAME                    PK       UI       FK       ALL
------------------------------ -------- -------- -------- --------
US02                           IMPLICIT IMPLICIT IMPLICIT NO
```

No rows are added to DBA_LOG_GROUPS

ALL

For example:

```
dbms_capture_adm.prepare_schema_instantiation ('US02','ALL');
```

Setting the SUPPLEMENTAL_LOGGING parameter to ALL has the following effect:

The above procedure sets the SPARE1 column of the USER$ table to 8 for user US02:

```
SQL> SELECT spare1 FROM sys.user$ WHERE name = 'US02';

SPARE1
------
     8
```

One row is added to STREAMS$_PREPARE_DDL for the user (in this case user# 87). The FLAGS column is set to 15.

```
SQL> SELECT global_flag,usrid,scn,timestamp,flags
FROM sys.streams$_prepare_ddl;

GLOBAL_FLAG      USRID        SCN TIMESTAMP      FLAGS
----------- ---------- ---------- --------- ----------
          0         87    1328053 17-MAR-13          8
```

One row is added to STREAMS$_PREPARE_OBJECT for each object owned by the user. In this example there are three tables in the US02 schema:

```
SQL> SELECT obj#,cap_type,ignore_scn,timestamp,flags
FROM sys.streams$_prepare_object;

      OBJ#   CAP_TYPE IGNORE_SCN TIMESTAMP      FLAGS
---------- ---------- ---------- --------- ----------
     76087          0    1328059 17-MAR-13          0
     76089          0    1328063 17-MAR-13          0
     76091          0    1328066 17-MAR-13          0
```

If a new table is created in the US02 schema, this object will automatically be added to the STREAMS$_PREPARE_OBJECTS table.

DBA_CAPTURE_PREPARED_SCHEMAS reports the following:

```
SQL> SELECT schema_name,
```

```
  supplemental_log_data_pk AS "PK",
  supplemental_log_data_pk AS "UI",
  supplemental_log_data_pk AS "FK",
  supplemental_log_data_pk AS "ALL"
FROM dba_capture_prepared_schemas;

SCHEMA_NAME                    PK       UI       FK       ALL
------------------------------ -------- -------- -------- --------
US02                           NO       NO       NO       IMPLICIT
```

No rows are added to DBA_LOG_GROUPS

KEY and ALL

For example:

```
dbms_capture_adm.prepare_schema_instantiation ('US02','KEY');
dbms_capture_adm.prepare_schema_instantiation ('US02','ALL');
```

Setting the SUPPLEMENTAL_LOGGING parameter to both KEY and ALL has the following effect:

The above procedure sets the SPARE1 column of the USER$ table to 15 for user US02:

```
SQL> SELECT spare1 FROM sys.user$ WHERE name = 'US02';

SPARE1
------
    15
```

One row is added to STREAMS$_PREPARE_DDL for the user (in this case user# 87). The FLAGS column is set to 15.

```
SQL> SELECT global_flag,usrid,scn,timestamp,flags
FROM sys.streams$_prepare_ddl;

GLOBAL_FLAG      USRID        SCN TIMESTAMP      FLAGS
----------- ---------- ---------- --------- ----------
          0         87    1321096 17-MAR-13         15
```

One row is added to STREAMS$_PREPARE_OBJECT for each object owned by the user. In this example there are three tables in the US02 schema:

```
SQL> SELECT obj#,cap_type,ignore_scn,timestamp,flags
FROM sys.streams$_prepare_object;

      OBJ#   CAP_TYPE IGNORE_SCN TIMESTAMP      FLAGS
---------- ---------- ---------- --------- ----------
     76087          0    1321102 17-MAR-13          0
     76089          0    1321105 17-MAR-13          0
     76091          0    1321108 17-MAR-13          0
```

If a new table is created in the US02 schema, this object will automatically be added to the STREAMS$_PREPARE_OBJECTS table.

DBA_CAPTURE_PREPARED_SCHEMAS reports the following:

```
SQL> SELECT schema_name,
  supplemental_log_data_pk AS "PK",
  supplemental_log_data_pk AS "UI",
  supplemental_log_data_pk AS "FK",
  supplemental_log_data_pk AS "ALL"
FROM dba_capture_prepared_schemas;
```

```
SCHEMA_NAME                    PK       UI       FK       ALL
------------------------------ -------- -------- -------- --------
US02                           IMPLICIT IMPLICIT IMPLICIT IMPLICIT
```

No rows are added to DBA_LOG_GROUPS

NONE

For example:

```
dbms_capture_adm.prepare_schema_instantiation ('US02','NONE');
```

Setting schema level supplemental logging to NONE in the PREPARE_SCHEMA_INSTANTIATION procedure has some unpredictable effects:

The SPARE1 flag in the USER$ table for the user US02 is unaffected. For example, if it is currently 15, it will not be changed.

The SCN column in the STREAMS$_PREPARE_DDL table will be updated for the specified schema.

The IGNORE_SCN column in the STREAMS$_PREPARE_OBJECTS table will be updated for all tables in the specified schema.

The TIMESTAMP in the DBA_CAPTURE_PREPARED_SCHEMAS view will be updated.

GoldenGate Schema Supplemental Logging

According to the documentation, schema level supplemental logging is enabled for GoldenGate using the ADD SCHEMADATA command in GGSCI. For example:

```
[oracle@vm4]$ ggsci

GGSCI (vm4) 2> ADD SCHEMATRANDATA us02

2013-03-17 01:17:49  INFO    OGG-01788  SCHEMATRANDATA has been added on schema
us02.
```

Tracing this command shows that it executes the following procedure:

```
dbms_capture_adm.PREPARE_SCHEMA_INSTANTIATION('us02','ALLKEYS_ON');
```

ALLKEYS_ON is an undocumented option for PREPARE_SCHEMA_INSTANTIATION.

The only modification made to the database by this command is to set the SPARE1 column of USER$ to 64 for the US02 user.

```
SQL> SELECT spare1 FROM sys.user$ WHERE name = 'US02';

SPARE1
------
    64
```

To remove schema level supplemental logging for GoldenGate, use the DELETE SCHEMADATA command in GGSCI.

```
[oracle@vm4]$ ggsci

GGSCI (vm4) 2> DELETE SCHEMATRANDATA us02

2013-03-17 00:40:41  INFO    OGG-01792  SCHEMATRANDATA has been deleted on schema
us02.
```

Tracing this command shows that it executes the following procedure:

```
dbms_capture_adm.PREPARE_SCHEMA_INSTANTIATION('us02','ALLKEYS_OFF');
```

ALLKEYS_OFF is another undocumented option for PREPARE_SCHEMA_INSTANTIATION.

The only modification made to the database by this command is to set the SPARE1 column of USER$ to 0 for the US02 user.

```
SQL> SELECT spare1 FROM sys.user$ WHERE name = 'US02';

SPARE1
------
     0
```

No rows are added to the STREAMS$_PREPARE_DDL table:

```
SQL> SELECT * FROM sys.streams$_prepare_ddl;

no rows selected
```

No rows are added to the STREAMS$_PREPARE_OBJECTS table:

```
SQL> SELECT * FROM sys.streams$_prepare_object;

no rows selected
```

No new rows appear in the DBA_CAPTURE_PREPARED_SCHEMAS view:

```
SQL> SELECT * FROM dba_capture_prepared_schemas;

no rows selected
```

Note that $ORACLE_HOME/rdbms/admin/dcore.bsq states that the above flags are defined in ktscts.h

Table Level Supplemental Logging

Oracle GoldenGate uses supplemental log groups to configure supplemental logging at table level.

Supplemental logging is enabled using the ADD TRANDATA command in GGSCI.

```
[oracle@vm4]$ ggsci

GGSCI (vm4) 1> DBLOGIN USERID us01 PASSWORD us01

GGSCI (vm4) 2> ADD TRANDATA t22

Logging of supplemental redo data enabled for table US01.T22.
```

When the above statement is executed, internally the table is locked using:

```
LOCK TABLE "US01"."T22" IN SHARE MODE NOWAIT
```

The following DDL statement is executed to create a supplemental log group for the specified table.

```
ALTER TABLE "US01"."T22" ADD SUPPLEMENTAL LOG GROUP "GGS_76111" ("C1") ALWAYS  /*
GOLDENGATE_DDL_REPLICATION */
```

In this example, the table T22 has a single primary key column C1 which is included in the supplemental logging.

The current status of supplemental logging for a table can be verified using the INFO TRANDATA command in GGSCI. For example:

```
[oracle@vm4]$ ggsci

GGSCI (vm4.juliandyke.com) 3> INFO TRANDATA t22

Logging of supplemental redo log data is enabled for table US01.T22.

Columns supplementally logged for table US01.T22: C1.
```

Supplemental logging is disabled using the DELETE TRANDATA command in GGSCI.

```
[oracle@vm4]$ ggsci

GGSCI (vm4) 1> DBLOGIN USERID us01 PASSWORD us01

GGSCI (vm4) 2> DELETE TRANDATA us01.t22

Logging of supplemental redo data disabled for table US01.T22.
```

Note that, unlike the ADD TRANDATA command, the schema name MUST be specified for the DELETE TRANDATA command. This appears to be a bug.

Also unlike the ADD TRANDATA command, it is not necessary for the table to be locked in order to disable supplemental log groups.

The following DDL statement is executed to create a supplemental log group for the specified table.

```
ALTER TABLE "US01"."T22" DROP SUPPLEMENTAL LOG GROUP "GGS_76111"
/* GOLDENGATE_DDL_REPLICATION */
```

Although rows are added to SYS.CON$, SYS.CDEF$ and SYS.CCOL$ for the new constraint, it is not reported in the DBA_CONSTRAINTS view.

This is because log group constraints (type 12) are specifically excluded from the view.

Supplemental Log Groups

Table level supplemental logging is defined by supplemental log groups. In this case each table has a dedicated supplemental log group.

Supplemental log groups are defined as constraints internally. Supplemental log group constraints have internal constraint type 12 and are excluded from the DBA_CONSTRAINTS view.

The examples in this section are based on the following table definitions:

```
CREATE TABLE t27
(
  c1 NUMBER,
  c2 VARCHAR2(30),
  c3 NUMBER,
  c4 NUMBER,
  c5 NUMBER,
  c6 VARCHAR2(10),
  c7 DATE
);

ALTER TABLE t27 ADD CONSTRAINT t27_pk PRIMARY KEY (c3,c4,c5);

ALTER TABLE t27 ADD CONSTRAINT t27_uk UNIQUE (c6);
```

The table already has a three column primary key and a unique constraint on a separate column.

Both object numbers for the table definition are 76129.

```
SQL> SELECT object_id,data_object_id FROM dba_objects
  2  WHERE owner = 'US01'
  3  AND object_name = 'T27';

 OBJECT_ID DATA_OBJECT_ID
---------- --------------
     76129          76129
```

The ADD TRANDATA command in GGSCI executes the following statement to enable supplemental logging:

```
ALTER TABLE "US01"."T27" ADD SUPPLEMENTAL LOG GROUP "GGS_76129" ("C3","C4","C5")
ALWAYS  /* GOLDENGATE_DDL_REPLICATION */
```

The user ID for the US01 user can be determined as follows:

```
SQL> SELECT user# FROM user$ WHERE name = 'US01'

     USER#
----------
        84
```

Within the data dictionary the constraint definitions are stored in three tables:

- CDEF$ - constraint definitions
- CCOL$ - constraint columns
- CON$ - constraints

The CDEF$ table contains one row for every constraint.

Column Name	Data Type
CON#	NUMBER
OBJ#	NUMBER
COLS	NUMBER
TYPE#	NUMBER
ROBJ#	NUMBER
RCON#	NUMBER
RRULES	VARCHAR2(3)
MATCH#	NUMBER
REFACT	NUMBER
ENABLED	NUMBER
CONDLENGTH	NUMBER
CONDITION	LONG
INTCOLS	NUMBER
MTIME	DATE
DEFER	NUMBER
SPARE1	NUMBER
SPARE2	NUMBER
SPARE3	NUMBER
SPARE4	VARCHAR2(1000)
SPARE5	VARCHAR2(1000)
SPARE6	DATE

Following creation of the supplemental log group, the CDEF$ table contains three constraint definitions for the T27 table.

The following query lists all constraint definitions for the T27 table.

```
SQL> SELECT con#, obj#, cols, type#, enabled, intcols,
mtime, defer, spare1, spare2, spare3
FROM cdef$
WHERE obj# = 76129;
```

The following table summarizes significant column values in CDEF$ for each of the three constraints:

Column Name	Primary Key	Unique Key	Supp Log Group
CON#	11152	11153	11154
OBJ#	76129	76129	76129
COLS	3	1	3
TYPE#	2	3	12
ENABLED	76130	76131	
INTCOLS	3	1	3
MTIME	17-MAR-13	17-MAR-13	17-MAR-13
DEFER	4	4	64
SPARE1	6	6	6
SPARE2	0	0	0
SPARE3	1341085	1341104	1341163

The following table summarizes values for the TYPE# column. These values are listed for the CDEF$ table in $ORACLE_HOME/rdbms/admin/dcore.bsq

Type#	Description
1	Check constraint
2	Primary key constraint
3	Unique constraint
4	Referential (foreign key) constraint
5	View with CHECK option constraint
6	View READ ONLY check constraint
7	NOT NULL constraint
8	Hash expression for hash cluster
9	Scoped REF column constraint
10	REF column WITH ROWID constraint
11	REF/ADT column with NOT NULL constraint
12	Log Groups for supplemental logging
13	Allow PK reference value storage in REF column
14	Primary key supplemental logging
15	Unique key supplemental logging
16	Foreign key supplemental logging
17	All column supplemental logging

Note that GoldenGate only appears to use type 12. It does not appear to use types 14-17.

The CCOL$ table has the following columns:

Column Name	Data Type
CON#	NUMBER
OBJ#	NUMBER
COL#	NUMBER
POS#	NUMBER
INTCOL#	NUMBER
SPARE1	NUMBER
SPARE2	NUMBER

SPARE3	NUMBER
SPARE4	VARCHAR2(1000)
SPARE5	VARCHAR2(1000)
SPARE6	DATE

```
SQL> SELECT con#, obj#, col# pos#, intcol#
FROM sys.ccol$
WHERE obj# = 76129
ORDER BY con#,pos#;

      CON#       OBJ#       COL#       POS#    INTCOL#
---------- ---------- ---------- ---------- ----------
     11152      76129          3          1          3
     11152      76129          4          2          4
     11152      76129          5          3          5
     11153      76129          6          1          6
     11154      76129          3          1          3
     11154      76129          4          2          4
     11154      76129          5          3          5

7 rows selected.
```

In the above result set,

- the first three rows are the columns for the primary key constraint
- the next row is the column for the unique key constraint
- the last three rows are the columns for the supplemental log group

The CON$ table has the following columns:

Column Name	Data Type
OWNER#	NUMBER
NAME	VARCHAR2(30)
CON#	NUMBER
SPARE1	NUMBER
SPARE2	NUMBER
SPARE3	NUMBER
SPARE4	VARCHAR2(1000)
SPARE5	VARCHAR2(1000)
SPARE6	DATE

```
SQL> SELECT owner#, name, con#, spare1
FROM sys.con$
WHERE con# BETWEEN 11152 AND 11154;

OWNER#   NAME               CON#    SPARE1
------   ----------------   -----   ------
84       T27_PK             11152        0
84       T27_UK             11153        0
84       GGS_76129          11154        0
```

Note that the GoldenGate constraint is named GGS_<object_id>

The DBA_LOG_GROUPS view contains the following columns:

Column Name	Data Type
OWNER	VARCHAR2(30)
LOG_GROUP_NAME	VARCHAR2(30)
TABLE_NAME	VARCHAR2(30)
LOG_GROUP_TYPE	VARCHAR2(28)
ALWAYS	VARCHAR2(11)
GENERATED	VARCHAR2(14)

The new log group is reported in the DBA_LOG_GROUPS view. For example:

```
SQL> SELECT log_group_name,log_group_type,always,generated
FROM dba_log_groups
WHERE owner = 'US01'
AND table_name = 'T27';

LOG_GROUP_NAME       LOG_GROUP_TYPE       ALWAYS      GENERATED
-------------------- -------------------- ----------- --------------
GGS_76129            USER LOG GROUP       ALWAYS      USER NAME
```

The DBA_LOG_GROUP_COLUMNS view contains the following columns:

Column Name	Data Type
OWNER	VARCHAR2(30)
LOG_GROUP_NAME	VARCHAR2(30)
TABLE_NAME	VARCHAR2(30)
COLUMN_NAME	VARCHAR2(4000)
POSITION	NUMBER
LOGGING_PROPERTY	VARCHAR2(6)

One row is reported in the DBA_LOG_GROUP_COLUMNS view for each column in the supplemental log group. For example:

```
SQL> SELECT log_group_name,column_name,position,logging_property
FROM dba_log_group_columns
WHERE owner = 'US01'
AND table_name = 'T27';

LOG_GROUP_NAME       COLUMN_NAME            POSITION LOGGING_PROPERTY
-------------------- -------------------- ---------- ----------------
GGS_76129            C3                            1 LOG
GGS_76129            C4                            2 LOG
GGS_76129            C5                            3 LOG
```

Supplemental Data

GoldenGate only appears to support supplemental log groups. However, Oracle also supports supplemental data logging.

Supplemental data logging is also enabled and disabled using the ALTER TABLE statement. Logging can be specified for primary key, unique key, foreign keys or for all columns in the table.

Primary Key Columns

For example:

```
CREATE TABLE k1
(
  c1 NUMBER,
  c2 NUMBER,
  c3 NUMBER,
  c4 NUMBER
);

ALTER TABLE k1 ADD CONSTRAINT k1_pk PRIMARY KEY (c2,c3);

ALTER TABLE k1 ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

The new table is object 76146.

```
SQL> SELECT obj# FROM sys.obj$
WHERE owner# =
(
  SELECT user#
  FROM sys.user$
  WHERE name = 'US01'
)
AND name = 'K1';

OBJ#
----
76146
```

A new constraint definition is added to CDEF$

```
SQL> SELECT con#, obj#, cols, type#, enabled, intcols,
mtime, defer, spare1, spare2, spare3
FROM sys.cdef$
WHERE obj# = 76146;
```

Column Name	Primary Key	Supplemental Data
CON#	11169	11170
OBJ#	76146	76146
COLS	2	2
TYPE#	2	14
ENABLED	76147	
INTCOLS	2	
MTIME	17-MAR-13	17-MAR-13
DEFER	4	72
SPARE1	6	6
SPARE2	0	0
SPARE3	1348619	1348632

Note that the for the supplemental data constraint

- TYPE is 14
- INTCOLS is NULL.

New constraint columns are added to CCOL$ for the primary key constraint. However, no rows are added for the supplemental data constraint.

```
SQL> SELECT con#, obj#, col# pos#, intcol#
FROM sys.ccol$
WHERE obj# = 76146
ORDER BY con#,pos#;

      CON#       OBJ#       COL#       POS#    INTCOL#
---------- ---------- ---------- ---------- ----------
     11169      76146          2          1          2
     11169      76146          3          2          3
```

A row is added to CON$ for the primary key constraint. However, no row is added for the supplemental data constraint.

```
SQL> SELECT owner#, name, con#, spare1
FROM sys.con$
WHERE con# = 11169;

OWNER#   NAME               CON#    SPARE1
------   ----------------   -----   ------
84       K1_PK              11169        0
```

The supplemental data constraint is not reported in DBA_CONSTRAINTS as this view excludes constraints of type 14.

Unique Columns

For example:

```
CREATE TABLE k2
(
  c1 NUMBER,
  c2 NUMBER,
  c3 NUMBER,
  c4 NUMBER
);

ALTER TABLE k2 ADD CONSTRAINT k2_uk UNIQUE (c3,c4);

ALTER TABLE k2 ADD SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
```

The new table is object 76148.

```
SQL> SELECT obj# FROM sys.obj$
WHERE owner# =
(
  SELECT user#
  FROM sys.user$
  WHERE name = 'US01'
)
AND name = 'K2';

OBJ#
----
76148
```

A new constraint definition is added to CDEF$

```
SQL> SELECT con#, obj#, cols, type#, enabled, intcols,
mtime, defer, spare1, spare2, spare3
FROM sys.cdef$
WHERE obj# = 76148;
```

Column Name	Unique Key	Supplemental Data
CON#	11171	11172
OBJ#	76148	76148
COLS	2	0
TYPE#	3	15
ENABLED	76149	
INTCOLS	2	
MTIME	17-MAR-13	17-MAR-13
DEFER	4	8
SPARE1	6	6
SPARE2	0	0
SPARE3	1349439	1349452

Note that the for the supplemental data constraint

- TYPE is 15
- INTCOLS is NULL.

New constraint columns are added to CCOL$ for the unique key constraint. However, no rows are added for the supplemental data constraint.

```
SQL> SELECT con#, obj#, col# pos#, intcol#
FROM sys.ccol$
WHERE obj# = 76148
ORDER BY con#,pos#;

      CON#       OBJ#       COL#       POS#    INTCOL#
---------- ---------- ---------- ---------- ----------
     11171      76148          3          1          3
     11171      76148          4          2          4
```

A row is added to CON$ for the unique key constraint. However, no row is added for the supplemental data constraint.

```
SQL> SELECT owner#, name, con#, spare1
FROM sys.con$
WHERE con# = 11171;

OWNER#   NAME               CON#    SPARE1
------   ----------------   -----   ------
84       K2_UK              11171        0
```

The supplemental data constraint is not reported in DBA_CONSTRAINTS as this view excludes constraints of type 15.

Foreign Key Columns

For example:

```
CREATE TABLE k0
(
  c1 NUMBER,
  c2 NUMBER,
  c3 NUMBER,
  c4 NUMBER
);

ALTER TABLE k0 ADD CONSTRAINT k0_pk PRIMARY KEY (c1,c2);

CREATE TABLE k3
(
  c1 NUMBER,
  c2 NUMBER,
  c3 NUMBER,
  c4 NUMBER
);

ALTER TABLE k3 ADD CONSTRAINT k3_fk FOREIGN KEY (c3,c4)
REFERENCES k0 (c1,c2);

ALTER TABLE k3 ADD SUPPLEMENTAL LOG DATA (FOREIGN KEY) COLUMNS;
```

The new table is object 76150.

```
SQL> SELECT obj# FROM sys.obj$
WHERE owner# =
(
  SELECT user#
  FROM sys.user$
  WHERE name = 'US01'
)
AND name = 'K3';

OBJ#
----
76150
```

A new constraint definition is added to CDEF$

```
SQL> SELECT con#, obj#, cols, type#, enabled, intcols,
mtime, defer, spare1, spare2, spare3
FROM sys.cdef$
WHERE obj# = 76150;
```

Column Name	Foreign Key	Supplemental Data
CON#	11173	11174
OBJ#	76150	76150
COLS	2	0
TYPE#	4	16
ENABLED	76140	
INTCOLS	2	
MTIME	17-MAR-13	17-MAR-13
DEFER	4	8
SPARE1	6	6
SPARE2	0	0
SPARE3	1349723	1349736

Note that the for the supplemental data constraint:

- TYPE is 16
- INTCOLS is NULL.

New constraint columns are added to CCOL$ for the foreign key constraint. However, no rows are added for the supplemental data constraint.

```
SQL> SELECT con#, obj#, col# pos#, intcol#
FROM sys.ccol$
WHERE obj# = 76150
ORDER BY con#,pos#;

      CON#       OBJ#       COL#       POS#    INTCOL#
---------- ---------- ---------- ---------- ----------
     11173      76150          3          1          3
     11173      76150          4          2          4
```

A row is added to CON$ for the foreign key constraint. However, no row is added for the supplemental data constraint.

```
SQL> SELECT owner#, name, con#, spare1
FROM sys.con$
WHERE con# = 11173;

OWNER#   NAME               CON#    SPARE1
------   ----------------   -----   ------
84       K3_FK              11173        0
```

The supplemental data constraint is not reported in DBA_CONSTRAINTS as this view excludes constraints of type 16.

All Columns

For example:

```
CREATE TABLE k4
(
  c1 NUMBER,
  c2 NUMBER,
  c3 NUMBER,
```

```
  c4 NUMBER
);

ALTER TABLE k4 ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

Note that in this example the table does not have any primary key, unique key or foreign key constraints.

The new table is object 76151.

```
SQL> SELECT obj# FROM sys.obj$
WHERE owner# =
(
  SELECT user#
  FROM sys.user$
  WHERE name = 'US01'
)
AND name = 'K4';

OBJ#
----
76151
```

A new constraint definition is added to CDEF$

```
SQL> SELECT con#, obj#, cols, type#, enabled, intcols,
mtime, defer, spare1, spare2, spare3
FROM sys.cdef$
WHERE obj# = 76151;
```

Column Name	Supplemental Data
CON#	11175
OBJ#	76151
COLS	0
TYPE#	17
ENABLED	
INTCOLS	
MTIME	17-MAR-13
DEFER	72
SPARE1	6
SPARE2	0
SPARE3	1349952

Note that the for the supplemental data constraint

- TYPE is 17
- INTCOLS is NULL.

No constraint columns are added to CCOL$

No constraints are added to CON$

The supplemental data constraint is not reported in DBA_CONSTRAINTS as this view excludes constraints of type 17.

Supplemental Logging in Redo Logs

Redo log and archive log contents can be dumped using the ALTER SYSTEM statement. For example:

```
ALTER SYSTEM DUMP LOGFILE '/u01/app/oradata/NORTH/redo01.log';
```

The above command dumps the entire contents of the redo log. However, if supplemental logging is enabled, the ALTER SYSTEM command will omit any supplemental data that has been included in the redo log. This does not mean the supplemental data is not there, the symbolic dump just skips over it.

In order to inspect the supplemental data sections in the redo logs, I extended my RedoAnalyzer tool to output supplemental data.

I created the following tables to illustrate the four different types for supplemental data logging. The tables are based on students and courses. The relationship between the tables is slightly artificial as I needed an example of a multi-column foreign key

The example tables were created as follows:

```
CREATE TABLE course
(
  university  VARCHAR2(30),
  subject     VARCHAR2(30),
  entry_year  NUMBER
);

ALTER TABLE course
ADD CONSTRAINT course_pk
PRIMARY KEY (university,subject,entry_year);

CREATE TABLE student
(
  student_key NUMBER,
  first_name  VARCHAR2(30),
  surname     VARCHAR2(30),
  gender      VARCHAR2(1),
  university  VARCHAR2(30),
  subject     VARCHAR2(30),
  entry_year  NUMBER,
  tuition_fee NUMBER
);

ALTER TABLE student
ADD CONSTRAINT student_pk
PRIMARY KEY (student_key);

ALTER TABLE student
ADD CONSTRAINT student_uk
UNIQUE (first_name,surname);

ALTER TABLE student
ADD CONSTRAINT student_course
FOREIGN KEY (university,subject,entry_year)
REFERENCES course (university,subject,entry_year);
```

The above tables were populated with the following initial data:

```
INSERT INTO course VALUES ('Oxford','Biology',2013);
INSERT INTO course VALUES ('Oxford','Biology',2014);
INSERT INTO course VALUES ('Cambridge','Physics',2013);
INSERT INTO course VALUES ('Cambridge','Chemistry',2013);

INSERT INTO student VALUES (1001,'Lucy','Brotherton','F',
'Cambridge','Chemistry',2013,9000);

INSERT INTO student VALUES (1002,'Rebecca','Brown','F',
'Oxford','Biology',2013,9000);

INSERT INTO student VALUES (1003,'Simon','Campbell','M',
'Cambridge','Physics',2013,7500);

INSERT INTO student VALUES (1004,'Jason','Robinson','M',
```

```
'Oxford','Biology',2013,7500);

COMMIT;
```

Primary Key Supplemental Logging

This example demonstrates primary key supplemental logging which is enabled as follows:

```
ALTER TABLE student ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

The following statement updates the non-key TUITION_FEE column:

```
UPDATE student SET tuition_fee = 6000
WHERE student_key = 1001;
```

The redo log dump for the above change is as follows:

```
REDO RECORD - Thread:1 RBA: 0x00002f.00000005.0010 LEN: 0x022c VLD: 0x05
SCN: 0x0000.0015b66d SUBSCN:  1 03/19/2013 04:33:05
(LWN RBA: 0x00002f.00000005.0010 LEN: 0002 NST: 0001 SCN: 0x0000.0015b66d)

CHANGE #1 TYP:0 CLS:19 AFN:3 DBA:0x00c00090 OBJ:4294967295 SCN:0x0000.0015b62c
SEQ:1 OP:5.2 ENC:0 RBL:0
ktudh redo: slt: 0x0001 sqn: 0x000003c6 flg: 0x0012 siz: 176 fbi: 0
            uba: 0x00c006e9.013e.16    pxid:  0x0000.000.00000000

CHANGE #2 TYP:0 CLS:20 AFN:3 DBA:0x00c006e9 OBJ:4294967295 SCN:0x0000.0015b62b
SEQ:2 OP:5.1 ENC:0 RBL:0
ktudb redo: siz: 176 spc: 4024 flg: 0x0012 seq: 0x013e rec: 0x16
            xid:  0x0002.001.000003c6
ktubl redo: slt: 1 rci: 0 opc: 11.1 [objn: 76381 objd: 76381 tsn: 4]
Undo type:  Regular undo        Begin trans    Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
             0x00000000  prev ctl uba: 0x00c006e9.013e.14
prev ctl max cmt scn:  0x0000.0015ae94  prev tx cmt scn:  0x0000.0015aec8
txn start scn:  0xffff.ffffffff  logon user: 84  prev brb: 12584673  prev bcl: 0
BuExt idx: 0 flg2: 0
KDO undo record:
KTB Redo
op: 0x03  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: Z
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003c3  hdba: 0x010003c2
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 0(0x0) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: 0
col  7: [ 2]  c2 5b

CHANGE #3 TYP:2 CLS:1 AFN:4 DBA:0x010003c3 OBJ:76381 SCN:0x0000.0015b5b6 SEQ:1
OP:11.5 ENC:0 RBL:0
KTB Redo
op: 0x11  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F  xid:  0x0002.001.000003c6    uba: 0x00c006e9.013e.16
Block cleanout record, scn:  0x0000.0015b66d ver: 0x01 opt: 0x02, entries follow...
  itli: 1  flg: 2  scn: 0x0000.0015b5b6
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003c3  hdba: 0x010003c2
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 0(0x0) flag: 0x2c lock: 2 ckix: 0
ncol: 8 nnew: 1 size: 0
col  7: [ 2]  c2 3d
```

```
CHANGE #4 MEDIA RECOVERY MARKER SCN:0x0000.00000000 SEQ:0 OP:5.20 ENC:0
session number   = 43
serial  number   = 717
transaction name =
version 186647296
audit sessionid 180679
Client Id =
login   username = US01
```

The above redo record consists of four changes:

Change #	Operation	Description
1	5.2	Allocate transaction slot in undo header
2	5.1	Undo for update (URP) operation
3	11.5	Redo for update (URP) operation
4	5.20	Media recovery marker

The above statement will generate supplemental data as follows:

```
Header (20 bytes)
  00011C01 00080001 00000008 00001000 00000000

Bytes 00-01: Number of columns (1)
Bytes 02-03: Unknown
Bytes 04-05: Column number of first column modified (8)
Bytes 06-07: Unknown
Bytes 08-0B: Column number of first column modified (8)
Bytes 0C-0D: Unknown
Bytes 0E-0F: Unknown
Bytes 10-13: Unknown
```

The column number is either SEGCOL# or INTCOL# in COL$ for the first column updated in the UPDATE statement. It is definitely not COL#.

```
Column List Header (2 bytes)
  0001

Bytes 00-01: INTCOL# of column 1 (1)

Column Size List (2 bytes)
  0003

Bytes 00-01: Length of column 1 (3 bytes)

Column 1 Value (3 bytes)
  C2 0B 02

col  0: [ 3]  c2 0b 02                    # STUDENT_KEY = 1001
```

The above column value is the value of the primary key for the row that has been updated.

Unique Key Supplemental Logging

This example demonstrates unique key supplemental logging which is enabled as follows:

```
ALTER TABLE student ADD SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
```

The following statement updates the unique key SURNAME column

```
UPDATE student SET surname = 'Harris'
WHERE student_key = 1002;
```

The redo log dump for the above change is as follows:

```
REDO RECORD - Thread:1 RBA: 0x000031.00000005.0010 LEN: 0x0250 VLD: 0x05
SCN: 0x0000.0015bdeb SUBSCN:  1 03/19/2013 05:27:24
(LWN RBA: 0x000031.00000005.0010 LEN: 0003 NST: 0001 SCN: 0x0000.0015bdeb)

CHANGE #1 TYP:0 CLS:27 AFN:3 DBA:0x00c000d0 OBJ:4294967295 SCN:0x0000.0015bda6
SEQ:1 OP:5.2 ENC:0 RBL:0
ktudh redo: slt: 0x0007 sqn: 0x00000483 flg: 0x0012 siz: 208 fbi: 0
            uba: 0x00c0021d.010e.18    pxid:  0x0000.000.00000000

CHANGE #2 TYP:0 CLS:28 AFN:3 DBA:0x00c0021d OBJ:4294967295 SCN:0x0000.0015bda5
SEQ:1 OP:5.1 ENC:0 RBL:0
ktudb redo: siz: 208 spc: 3920 flg: 0x0012 seq: 0x010e rec: 0x18
            xid:  0x0006.007.00000483
ktubl redo: slt: 7 rci: 0 opc: 11.1 [objn: 76381 objd: 76381 tsn: 4]
Undo type:  Regular undo        Begin trans    Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
             0x00000000  prev ctl uba: 0x00c0021d.010e.17
prev ctl max cmt scn:  0x0000.0015b409  prev tx cmt scn:  0x0000.0015b43d
txn start scn:  0xffff.ffffffff  logon user: 84  prev brb: 12583449  prev bcl: 0
BuExt idx: 0 flg2: 0
KDO undo record:
KTB Redo
op: 0x04  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: L  itl: xid:  0x0008.017.000003d1 uba: 0x00c003ca.00da.21
                      flg: C---    lkc:  0     scn: 0x0000.0015b933
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003c3  hdba: 0x010003c2
itli: 1  ispac: 0  maxfr: 4858
tabn: 0 slot: 1(0x1) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: -1
col  2: [ 5]  42 72 6f 77 6e

CHANGE #3 TYP:2 CLS:1 AFN:4 DBA:0x010003c3 OBJ:76381 SCN:0x0000.0015ba79 SEQ:1
OP:11.5 ENC:0 RBL:0
KTB Redo
op: 0x11  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F  xid:  0x0006.007.00000483    uba: 0x00c0021d.010e.18
Block cleanout record, scn:  0x0000.0015bdeb ver: 0x01 opt: 0x02, entries follow...
  itli: 2  flg: 2  scn: 0x0000.0015ba79
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003c3  hdba: 0x010003c2
itli: 1  ispac: 0  maxfr: 4858
tabn: 0 slot: 1(0x1) flag: 0x2c lock: 1 ckix: 0
ncol: 8 nnew: 1 size: 1
col  2: [ 6]  48 61 72 72 69 73

CHANGE #4 MEDIA RECOVERY MARKER SCN:0x0000.00000000 SEQ:0 OP:5.20 ENC:0
session number   = 43
serial  number   = 793
transaction name =
version 186647296
audit sessionid 180693
Client Id =
login   username = US01
```

The above redo record consists of four changes:

Change #	Operation	Description
1	5.2	Allocate transaction slot in undo header
2	5.1	Undo for update (URP) operation
3	11.5	Redo for update (URP) operation

4	5.20	Media recovery marker

The above statement will generate supplemental data as follows:

```
Header (20 bytes)
  00011C01 00030001 00000003 00001000 00000000

Bytes 00-01: Number of columns (1)
Bytes 02-03: Unknown
Bytes 04-05: Column number of first column modified (3)
Bytes 06-07: Unknown
Bytes 08-0B: Column number of first column modified (3)
Bytes 0C-0D: Unknown
Bytes 0E-0F: Unknown
Bytes 10-13: Unknown
```

The column number is either SEGCOL# or INTCOL# in COL$ for the first column updated in the UPDATE statement. It is definitely not COL#.

```
Column List Header (2 bytes)
  0002

Bytes 00-01: INTCOL# of column 1 (2)

Column Size List (2 bytes)
  0007

Bytes 00-01: Length of column 1 (7 bytes)

Column 1 Value (7 bytes)
  52 65 62 65 63 63 61

col  1: [ 7]  52 65 62 65 63 63 61       # First Name = Rebecca
```

The above column value is the unchanged value in the unique key for the column that has been updated. In this case Rebecca Brown has become Rebecca Harris, so Rebecca is the unchanged value.

Foreign Key Supplemental Logging

This example demonstrates foreign key supplemental logging which is enabled as follows:

```
ALTER TABLE student ADD SUPPLEMENTAL LOG DATA (FOREIGN KEY) COLUMNS;
```

The following statement updates the foreign key SUBJECT column

```
UPDATE student SET subject = 'Chemistry'
WHERE student_key = 1003;
```

The redo log dump for the above change is as follows:

```
REDO RECORD - Thread:1 RBA: 0x000033.00000005.0010 LEN: 0x0234 VLD: 0x05
SCN: 0x0000.0015c139 SUBSCN:  1 03/19/2013 05:57:42
(LWN RBA: 0x000033.00000005.0010 LEN: 0002 NST: 0001 SCN: 0x0000.0015c139)

CHANGE #1 TYP:0 CLS:29 AFN:3 DBA:0x00c000e0 OBJ:4294967295 SCN:0x0000.0015c0e3
SEQ:1 OP:5.2 ENC:0 RBL:0
ktudh redo: slt: 0x0018 sqn: 0x00000314 flg: 0x0012 siz: 220 fbi: 0
            uba: 0x00c00195.013b.0b    pxid:  0x0000.000.00000000

CHANGE #2 TYP:0 CLS:30 AFN:3 DBA:0x00c00195 OBJ:4294967295 SCN:0x0000.0015c0e2
SEQ:2 OP:5.1 ENC:0 RBL:0
ktudb redo: siz: 220 spc: 6846 flg: 0x0012 seq: 0x013b rec: 0x0b
            xid:  0x0007.018.00000314
```

```
ktubl redo: slt: 24 rci: 0 opc: 11.1 [objn: 76381 objd: 76381 tsn: 4]
Undo type:  Regular undo        Begin trans    Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
             0x00000000  prev ctl uba: 0x00c00195.013b.09
prev ctl max cmt scn:  0x0000.0015b667  prev tx cmt scn:  0x0000.0015b69d
txn start scn:  0xffff.ffffffff  logon user: 84  prev brb: 12583310  prev bcl: 0
BuExt idx: 0 flg2: 0
KDO undo record:
KTB Redo
op: 0x04  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: L  itl: xid:  0x0008.010.000003d2 uba: 0x00c003ef.00da.08
                      flg: C---    lkc:  0     scn: 0x0000.0015c079
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003c3  hdba: 0x010003c2
itli: 1  ispac: 0  maxfr: 4858
tabn: 0 slot: 2(0x2) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: -2
col  5: [ 7]  50 68 79 73 69 63 73

CHANGE #3 TYP:2 CLS:1 AFN:4 DBA:0x010003c3 OBJ:76381 SCN:0x0000.0015c099 SEQ:1
OP:11.5 ENC:0 RBL:0
KTB Redo
op: 0x01  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F  xid:  0x0007.018.00000314    uba: 0x00c00195.013b.0b
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003c3  hdba: 0x010003c2
itli: 1  ispac: 0  maxfr: 4858
tabn: 0 slot: 2(0x2) flag: 0x2c lock: 1 ckix: 0
ncol: 8 nnew: 1 size: 2
col  5: [ 9]  43 68 65 6d 69 73 74 72 79

CHANGE #4 MEDIA RECOVERY MARKER SCN:0x0000.00000000 SEQ:0 OP:5.20 ENC:0
session number   = 36
serial  number   = 1525
transaction name =
version 186647296
audit sessionid 180704
Client Id =
login   username = US01
```

The above redo record consists of four changes:

Change #	Operation	Description
1	5.2	Allocate transaction slot in undo header
2	5.1	Undo for update (URP) operation
3	11.5	Redo for update (URP) operation
4	5.20	Media recovery marker

The above statement will generate supplemental data as follows:

```
Header (20 bytes)
  00021C01 00060001 00000006 00001000 00000000

Bytes 00-01: Number of columns (2)
Bytes 02-03: Unknown
Bytes 04-05: Column number of first column modified (6)
Bytes 06-07: Unknown
Bytes 08-0B: Column number of first column modified (6)
Bytes 0C-0D: Unknown
Bytes 0E-0F: Unknown
Bytes 10-13: Unknown
```

The column number is either SEGCOL# or INTCOL# in COL$ for the first column updated in the UPDATE statement. It is definitely not COL#.

```
Column List Header (4 bytes)
  0005 0007

Bytes 00-01: INTCOL# of column 1 (5)
Bytes 02-03: INTCOL# of column 2 (7)

Column Size List (4 bytes)
  0009 0003

Bytes 00-01: Length of column 1 (9 bytes)
Bytes 02-03: Length of column 2 (3 bytes)

Column 1 Value (9 bytes)
  43 61 6D 62 72 69 64 67 65

col  4: [ 9]  43 61 6d 62 72 69 64 67 65       # University = Cambridge

Column 2 Value (3 bytes)
  C2 15 0E

col  6: [ 3]  c2 15 0e       # Entry Year = 2013
```

The above column values are the unchanged values in the foreign key for the column that has been updated. In this case the subject has changed to chemistry, but the university remains as Cambridge and the entry year is still 2013.

All Column Supplemental Logging

This example demonstrates all column supplemental logging which is enabled as follows:

```
ALTER TABLE student ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

The following statement updates the non-key TUITION_FEE column

```
UPDATE student SET tuition_fee = 9000
WHERE student_key = 1004;
```

The redo log dump for the above change is as follows:

```
REDO RECORD - Thread:1 RBA: 0x000034.00000005.0010 LEN: 0x0278 VLD: 0x05
SCN: 0x0000.0015c5f4 SUBSCN:  1 03/19/2013 06:34:08
(LWN RBA: 0x000034.00000005.0010 LEN: 0002 NST: 0001 SCN: 0x0000.0015c5f4)

CHANGE #1 TYP:0 CLS:31 AFN:3 DBA:0x00c000f0 OBJ:4294967295 SCN:0x0000.0015c583
SEQ:1 OP:5.2 ENC:0 RBL:0
ktudh redo: slt: 0x001a sqn: 0x000003d2 flg: 0x0012 siz: 252 fbi: 0
            uba: 0x00c003f3.00da.1d    pxid:  0x0000.000.00000000

CHANGE #2 TYP:0 CLS:32 AFN:3 DBA:0x00c003f3 OBJ:4294967295 SCN:0x0000.0015c581
SEQ:1 OP:5.1 ENC:0 RBL:0
ktudb redo: siz: 252 spc: 1430 flg: 0x0012 seq: 0x00da rec: 0x1d
            xid:  0x0008.01a.000003d2
ktubl redo: slt: 26 rci: 0 opc: 11.1 [objn: 76381 objd: 76381 tsn: 4]
Undo type:  Regular undo        Begin trans    Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
             0x00000000  prev ctl uba: 0x00c003f3.00da.12
prev ctl max cmt scn:  0x0000.0015bac4  prev tx cmt scn:  0x0000.0015bad8
txn start scn:  0xffff.ffffffff  logon user: 84  prev brb: 12583916  prev bcl: 0
BuExt idx: 0 flg2: 0
KDO undo record:
```

```
KTB Redo
op: 0x03  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: Z
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003c6  hdba: 0x010003c2
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 0(0x0) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: 0
col  7: [ 2]  c2 4c

CHANGE #3 TYP:2 CLS:1 AFN:4 DBA:0x010003c6 OBJ:76381 SCN:0x0000.0015c5bf SEQ:1
OP:11.5 ENC:0 RBL:0
KTB Redo
op: 0x11  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F  xid:  0x0008.01a.000003d2    uba: 0x00c003f3.00da.1d
Block cleanout record, scn:  0x0000.0015c5f4 ver: 0x01 opt: 0x02, entries follow...
  itli: 1  flg: 2  scn: 0x0000.0015c5bf
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003c6  hdba: 0x010003c2
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 0(0x0) flag: 0x2c lock: 2 ckix: 0
ncol: 8 nnew: 1 size: 0
col  7: [ 2]  c2 5b

CHANGE #4 MEDIA RECOVERY MARKER SCN:0x0000.00000000 SEQ:0 OP:5.20 ENC:0
session number   = 41
serial  number   = 1981
transaction name =
version 186647296
audit sessionid 180713
Client Id =
login   username = US01
```

The above redo record consists of four changes:

Change #	Operation	Description
1	5.2	Allocate transaction slot in undo header
2	5.1	Undo for update (URP) operation
3	11.5	Redo for update (URP) operation
4	5.20	Media recovery marker

The above statement will generate supplemental data as follows:

```
Header (20 bytes)
  00071C01 00080001 00000008 00001000 00000000

Bytes 00-01: Number of columns (7)
Bytes 02-03: Unknown
Bytes 04-05: Column number of first column modified (8)
Bytes 06-07: Unknown
Bytes 08-0B: Column number of first column modified (8)
Bytes 0C-0D: Unknown
Bytes 0E-0F: Unknown
Bytes 10-13: Unknown
```

The column number is either SEGCOL# or INTCOL# in COL$ for the first column updated in the UPDATE statement. It is definitely not COL#.

```
Column List Header (14 bytes)
  0001 0002 0003 0004 0005 0006 0007

Bytes 00-01: INTCOL# of column 1 (1)
```

```
Bytes 02-03: INTCOL# of column 2 (2)
Bytes 04-05: INTCOL# of column 3 (3)
Bytes 06-07: INTCOL# of column 4 (4)
Bytes 08-09: INTCOL# of column 5 (5)
Bytes 0A-0B: INTCOL# of column 6 (6)
Bytes 0C-0D: INTCOL# of column 7 (7)

Column Size List (14 bytes)
  0003 0005 0008 0001 0006 0007 0003

Bytes 00-01: Length of column 1 (3 bytes)
Bytes 02-03: Length of column 2 (5 bytes)
Bytes 04-05: Length of column 3 (8 bytes)
Bytes 06-07: Length of column 4 (1 bytes)
Bytes 08-09: Length of column 5 (6 bytes)
Bytes 0A-0B: Length of column 6 (7 bytes)
Bytes 0C-0D: Length of column 7 (3 bytes)

Column 1 Value (3 bytes)
  C2 0B 05

col  0: [ 3]  c2 0b 05       # Student Key = 1004

Column 2 Value (5 bytes)
  4A 61 73 6F 6E

col  1: [ 5]  4a 61 73 6f 6e# First Name = Jason

Column 3 Value (8 bytes)
  52 6F 62 69 6E 73 6F 6E

col  2: [ 8]  52 6f 62 69 6e 73 6f 6e# Surname = Robinson

Column 4 Value (1 byte)
  4D

col  3: [ 1]  4d# Gender = M

Column 5 Value (6 bytes)
  4F 78 66 6F 72 64

col  4: [ 6]  4f 78 66 6f 72 64       # University = Oxford

Column 6 Value (7 bytes)
  42 69 6F 6C 6F 67 69

col  5: [ 7]  42 69 6f 6c 6f 67 79      # Subject = Biology

Column 7 Value (3 bytes)
  C2 15 0E

col  6: [ 3]  c2 15 0e# Entry Year = 2013
```

The above column values are the unchanged value in the entire row for the column that has been updated. In this case all columns except the TUITION_FEE column are included in the supplemental log data.

Minimal Supplemental Logging

If minimal supplemental logging is enabled, additional structures are added the undo for each change made by an UPDATE statement for chained rows and index clusters.

Chained Rows

Supplemental logging is added for chained rows. This appears to be used to identify the location of the row header which may be on a different block from that being modified.

An additional 28 bytes is added for each UPDATE involving a chained row.

This example demonstrates supplemental logging for chained rows. Minimal supplemental logging must be enabled as follows:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

This example uses the following objects:

```
CREATE TABLE t35
(
  c1 NUMBER PRIMARY KEY,
  c2 VARCHAR2 (2000),
  c3 VARCHAR2 (2000),
  c4 VARCHAR2 (2000),
  c5 VARCHAR2 (2000),
  c6 VARCHAR2 (2000),
  c7 NUMBER,
  c8 VARCHAR2 (2000),
  c9 VARCHAR2 (2000),
  c10 VARCHAR2 (2000),
  c11 VARCHAR2 (2000),
  c12 VARCHAR2 (2000),
  c13 NUMBER
);

INSERT INTO t35 (c1) VALUES (1001);
INSERT INTO t35 (c1) VALUES (1002);
INSERT INTO t35 (c1) VALUES (1003);
INSERT INTO t35 (c1) VALUES (1004);

INSERT INTO t35 VALUES
(
  1005,
  LPAD ('X',2000,'X'),
  LPAD ('X',2000,'X'),
  LPAD ('X',2000,'X'),
  LPAD ('X',2000,'X'),
  LPAD ('X',2000,'X'),
  10,
  LPAD ('X',2000,'X'),
  LPAD ('X',2000,'X'),
  LPAD ('X',2000,'X'),
  LPAD ('X',2000,'X'),
  LPAD ('X',2000,'X'),
  20
);

COMMIT;
```

In the above example, the first four rows only contain primary key values; the remaining columns are null.

The fifth row contains five 2000 byte values followed by a numeric value followed by five further 2000 byte values. These have been chosen to ensure that the row occupies three blocks. With the default block size of 8192, these filler columns also guarantee that the numeric value will always appear on the second block, irrespective of the order in which the blocks are allocated. This means we know that the row piece containing the numeric value will be chained.

The following statement updates the numeric value for the fifth row (primary key 1005).

```
UPDATE t35 SET c7 = c7 + 10
WHERE c1 = 1005;
```

The above change generates two redo records.

The first redo record for the change is as follows:

```
REDO RECORD - Thread:1 RBA: 0x000036.00000006.0010 LEN: 0x01e8 VLD: 0x05
SCN: 0x0000.001663e4 SUBSCN:  1 03/22/2013 22:57:06
(LWN RBA: 0x000036.00000006.0010 LEN: 0002 NST: 0001 SCN: 0x0000.001663e4)

CHANGE #1 TYP:0 CLS:23 AFN:3 DBA:0x00c000b0 OBJ:4294967295 SCN:0x0000.001663af
SEQ:1 OP:5.2 ENC:0 RBL:0
ktudh redo: slt: 0x000e sqn: 0x00000331 flg: 0x000a siz: 136 fbi: 0
            uba: 0x00c000b3.01a4.01    pxid:  0x0000.000.00000000

CHANGE #2 TYP:1 CLS:24 AFN:3 DBA:0x00c000b3 OBJ:4294967295 SCN:0x0000.001663e4
SEQ:1 OP:5.1 ENC:0 RBL:0
ktudb redo: siz: 136 spc: 0 flg: 0x000a seq: 0x01a4 rec: 0x01
            xid:  0x0004.00e.00000331
ktubl redo: slt: 14 rci: 0 opc: 11.1 [objn: 76390 objd: 76390 tsn: 4]
Undo type:  Regular undo        Begin trans    Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
             0x00000000  prev ctl uba: 0x00c000b2.01a4.2e
prev ctl max cmt scn:  0x0000.00165735  prev tx cmt scn:  0x0000.00165746
txn start scn:  0xffff.ffffffff  logon user: 84  prev brb: 12583237  prev bcl: 0
BuExt idx: 0 flg2: 0
KDO undo record:
KTB Redo
op: 0x03  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: Z
KDO Op code: LKR row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003ff  hdba: 0x010003fa
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 0 to: 0

CHANGE #3 TYP:2 CLS:1 AFN:4 DBA:0x010003ff OBJ:76390 SCN:0x0000.001663de SEQ:1
OP:11.4 ENC:0 RBL:0
KTB Redo
op: 0x11  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F  xid:  0x0004.00e.00000331    uba: 0x00c000b3.01a4.01
Block cleanout record, scn:  0x0000.001663e4 ver: 0x01 opt: 0x02, entries follow...
  itli: 1  flg: 2  scn: 0x0000.001663de
KDO Op code: LKR row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003ff  hdba: 0x010003fa
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 0 to: 2

CHANGE #4 MEDIA RECOVERY MARKER SCN:0x0000.00000000 SEQ:0 OP:5.20 ENC:0
session number   = 32
serial  number   = 183
transaction name =
version 186647296
audit sessionid 190072
Client Id =
login   username = US01
```

The above redo record consists of four changes:

Change #	Operation	Description
1	5.2	Allocate transaction slot in undo header
2	5.1	Undo for Lock Row (LKR) operation
3	11.4	Redo for Lock Row (LKR) operation
4	5.20	Media recovery marker

The first step is to lock the row. The row lock byte is stored in the row header which is in the first piece. In this case this is block DBA 0x010003ff, slot 0.

The second redo record contains the following:

```
REDO RECORD - Thread:1 RBA: 0x000036.00000007.0010 LEN: 0x0144 VLD: 0x01
SCN: 0x0000.001663e4 SUBSCN:  1 03/22/2013 22:57:06

CHANGE #1 TYP:0 CLS:24 AFN:3 DBA:0x00c000b3 OBJ:4294967295 SCN:0x0000.001663e4
SEQ:2 OP:5.1 ENC:0 RBL:0
ktudb redo: siz: 116 spc: 8012 flg: 0x0022 seq: 0x01a4 rec: 0x02
            xid:  0x0004.00e.00000331
ktubu redo: slt: 14 rci: 1 opc: 11.1 objn: 76390 objd: 76390 tsn: 4
Undo type:  Regular undo       Undo type:  Last buffer split:  No
Tablespace Undo:  No
             0x00000000
KDO undo record:
KTB Redo
op: 0x03  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: Z
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003fd  hdba: 0x010003fa
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 0(0x0) flag: 0x00 lock: 0 ckix: 0
ncol: 4 nnew: 1 size: 0
col  1: [ 2]  c1 0b

CHANGE #2 TYP:2 CLS:1 AFN:4 DBA:0x010003fd OBJ:76390 SCN:0x0000.001663de SEQ:1
OP:11.5 ENC:0 RBL:0
KTB Redo
op: 0x11  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F  xid:  0x0004.00e.00000331    uba: 0x00c000b3.01a4.02
Block cleanout record, scn:  0x0000.001663e4 ver: 0x01 opt: 0x02, entries follow...
  itli: 1  flg: 2  scn: 0x0000.001663de
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003fd  hdba: 0x010003fa
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 0(0x0) flag: 0x00 lock: 2 ckix: 0
ncol: 4 nnew: 1 size: 0
col  1: [ 2]  c1 15
```

The above redo record consists of two changes:

Change #	Operation	Description
1	5.1	Undo for update (URP) operation
2	11.5	Redo for update (URP) operation

The second step is to update the row piece containing the C7 column. In this example, this has been engineered to be a different block to that containing the row header. In this case this is block with DBA 0x010003fd

As this change is part of an on-going transaction, it is not necessary to allocate a new slot in the undo header (operation 5.2) or to include a media recovery marker (operation 5.20).

The above statement will generate supplemental data as follows:

```
Header (28 bytes)
  00000401 00070001 00000007 00001000 00000000 010003FF 00000000

Bytes 00-01: Number of columns (0)
Bytes 02-03: Unknown
Bytes 04-05: Column number of first column modified (7)
Bytes 06-07: Unknown
Bytes 08-0B: Column number of first column modified (7)
Bytes 0C-0D: Unknown
Bytes 0E-0F: Unknown
```

```
Bytes 10-13: Unknown
Bytes 14-17: DBA of first row piece (0x0x10003FF)
Bytes 18-21: Slot number of first row piece (0)
```

Note that in this case the header is 28 bytes in length. The additional eight bytes contain the DBA and slot ID of the first row piece for the row.

We can confirm that the DBA is the first row piece, as opposed to the previous row piece using the following update to column C13 in the same row. For example:

```
UPDATE t35 SET c13 = c13 + 10
WHERE c1 = 1005;
```

The first redo record generated by this change is identical to that generated for the previous statement. It locks the first row piece in slot 0 of block DBA 0x010003ff.

The second redo record is as follows:

```
REDO RECORD - Thread:1 RBA: 0x000037.00000006.0020 LEN: 0x015c VLD: 0x01
SCN: 0x0000.001668c2 SUBSCN:  1 03/22/2013 23:31:55

CHANGE #1 TYP:0 CLS:22 AFN:3 DBA:0x00c00847 OBJ:4294967295 SCN:0x0000.001668c2
SEQ:1 OP:5.1 ENC:0 RBL:0
ktudb redo: siz: 140 spc: 7482 flg: 0x0022 seq: 0x0129 rec: 0x05
            xid:  0x0003.007.000003ce
ktubu redo: slt: 7 rci: 4 opc: 11.1 objn: 76390 objd: 76390 tsn: 4
Undo type:  Regular undo       Undo type:  Last buffer split:  No
Tablespace Undo:  No
             0x00000000
KDO undo record:
KTB Redo
op: 0x04  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: L  itl: xid:  0x0003.017.000003cd uba: 0x00c00847.0129.03
                      flg: C---    lkc:  0     scn: 0x0000.00166892
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003fc  hdba: 0x010003fa
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 4(0x4) flag: 0x04 lock: 0 ckix: 132
ncol: 4 nnew: 1 size: 0
col  3: [ 2]  c1 15

CHANGE #2 TYP:2 CLS:1 AFN:4 DBA:0x010003fc OBJ:76390 SCN:0x0000.001668b5 SEQ:1
OP:11.5 ENC:0 RBL:0
KTB Redo
op: 0x11  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F  xid:  0x0003.007.000003ce    uba: 0x00c00847.0129.05
Block cleanout record, scn:  0x0000.001668c2 ver: 0x01 opt: 0x02, entries follow...
  itli: 1  flg: 2  scn: 0x0000.001668b5
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003fc  hdba: 0x010003fa
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 4(0x4) flag: 0x04 lock: 2 ckix: 132
ncol: 4 nnew: 1 size: 0
col  3: [ 2]  c1 1f
```

The above redo record consists of two changes:

Change #	Operation	Description
1	5.1	Undo for update (URP) operation
2	11.5	Redo for update (URP) operation

The second redo record updates the row piece containing the C13 column. In this case this is block with DBA 0x010003fc

The above statement will generate supplemental data as follows:

```
Header (28 bytes)
  00000401 000D0001 0000000D 00001000 00000000 010003FF 00000000

Bytes 00-01: Number of columns (0)
Bytes 02-03: Unknown
Bytes 04-05: Column number of first column modified (0x0D=13)
Bytes 06-07: Unknown
Bytes 08-0B: Column number of first column modified (0x0D=13)
Bytes 0C-0D: Unknown
Bytes 0E-0F: Unknown
Bytes 10-13: Unknown
Bytes 14-17: DBA of first row piece (0x010003FF)
Bytes 18-21: Slot number of first row piece (0)
```

The above example proves that the DBA is the first row piece (0x010003FF), as opposed to the previous row piece which we know from the previous test would be 0x010003FD).

However, we do not have any evidence that the remaining four bytes contain the slot number as this is 0x00000000 in this example which is inconclusive. We need to generate a row with a non-zero slot number.

Fortunately we originally created four empty rows. The row headers and primary keys for these rows will all be in the first block of the table. If we update one of these rows, this will force it to be chained across at least three blocks.

For example:

```
UPDATE t35
SET
  c2 = LPAD ('Y',2000,'Y'),
  c3 = LPAD ('Y',2000,'Y'),
  c4 = LPAD ('Y',2000,'Y'),
  c5 = LPAD ('Y',2000,'Y'),
  c6 = LPAD ('Y',2000,'Y'),
  c7 = 30,
  c8 = LPAD ('Y',2000,'Y'),
  c9 = LPAD ('Y',2000,'Y'),
  c10 = LPAD ('Y',2000,'Y'),
  c11 = LPAD ('Y',2000,'Y'),
  c12 = LPAD ('Y',2000,'Y'),
  c13 = 40
WHERE c1 = 1004;
```

The above statement generates too much redo to be useful. However, we can now update column C7 for this row to create a more controlled example:

```
UPDATE t35 SET c7 = c7 + 10
WHERE c1 = 1004;
```

This statement generates two redo records. The first redo record locks the row.

```
REDO RECORD - Thread:1 RBA: 0x000038.00000005.0010 LEN: 0x0200 VLD: 0x05
SCN: 0x0000.00166a3f SUBSCN:  1 03/22/2013 23:46:36
(LWN RBA: 0x000038.00000005.0010 LEN: 0003 NST: 0001 SCN: 0x0000.00166a3f)

CHANGE #1 TYP:0 CLS:17 AFN:3 DBA:0x00c00080 OBJ:4294967295 SCN:0x0000.00166a05
SEQ:1 OP:5.2 ENC:0 RBL:0
ktudh redo: slt: 0x0011 sqn: 0x00000316 flg: 0x0012 siz: 160 fbi: 0
            uba: 0x00c00152.0192.24    pxid:  0x0000.000.00000000
```

```
CHANGE #2 TYP:0 CLS:18 AFN:3 DBA:0x00c00152 OBJ:4294967295
SCN:0x0000.00166a04 SEQ:1 OP:5.1 ENC:0 RBL:0
ktudb redo: siz: 160 spc: 1672 flg: 0x0012 seq: 0x0192 rec: 0x24
            xid:  0x0001.011.00000316
ktubl redo: slt: 17 rci: 0 opc: 11.1 [objn: 76390 objd: 76390 tsn: 4]
Undo type:  Regular undo        Begin trans    Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
             0x00000000  prev ctl uba: 0x00c00152.0192.23
prev ctl max cmt scn:  0x0000.00165ec1  prev tx cmt scn:  0x0000.00165ece
txn start scn:  0xffff.ffffffff  logon user: 84  prev brb: 12583085  prev bcl: 0
BuExt idx: 0 flg2: 0
KDO undo record:
KTB Redo
op: 0x04  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: L  itl: xid:  0x0007.01c.0000031f uba: 0x00c001c7.013e.0b
                      flg: C---    lkc:  0     scn: 0x0000.001668b5
KDO Op code: LKR row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003fc  hdba: 0x010003fa
itli: 1  ispac: 0  maxfr: 4858
tabn: 0 slot: 3 to: 0

CHANGE #3 TYP:2 CLS:1 AFN:4 DBA:0x010003fc OBJ:76390 SCN:0x0000.00166a07 SEQ:1
OP:11.4 ENC:0 RBL:0
KTB Redo
op: 0x11  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F  xid:  0x0001.011.00000316    uba: 0x00c00152.0192.24
Block cleanout record, scn:  0x0000.00166a3f ver: 0x01 opt: 0x02, entries follow...
  itli: 3  flg: 2  scn: 0x0000.00166a07
KDO Op code: LKR row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x010003fc  hdba: 0x010003fa
itli: 1  ispac: 0  maxfr: 4858
tabn: 0 slot: 3 to: 1

CHANGE #4 MEDIA RECOVERY MARKER SCN:0x0000.00000000 SEQ:0 OP:5.20 ENC:0
session number   = 45
serial  number   = 205
transaction name =
version 186647296
audit sessionid 190086
Client Id =
login   username = US01
```

The above redo record consists of four changes:

Change #	Operation	Description
1	5.2	Allocate transaction slot in undo header
2	5.1	Undo for Lock Row (LKR) operation
3	11.4	Redo for Lock Row (LKR) operation
4	5.20	Media recovery marker

From the above we can conclude that the row header is slot 3 on block 0x010003fc

The second redo record updates column C7:

```
REDO RECORD - Thread:1 RBA: 0x000038.00000006.0020 LEN: 0x0144 VLD: 0x01
SCN: 0x0000.00166a3f SUBSCN:  1 03/22/2013 23:46:36

CHANGE #1 TYP:0 CLS:18 AFN:3 DBA:0x00c00152 OBJ:4294967295 SCN:0x0000.00166a3f
SEQ:1 OP:5.1 ENC:0 RBL:0
ktudb redo: siz: 116 spc: 1510 flg: 0x0022 seq: 0x0192 rec: 0x25
            xid:  0x0001.011.00000316
ktubu redo: slt: 17 rci: 36 opc: 11.1 objn: 76390 objd: 76390 tsn: 4
```

```
Undo type:  Regular undo       Undo type:  Last buffer split:  No
Tablespace Undo:  No
             0x00000000
KDO undo record:
KTB Redo
op: 0x03  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: Z
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x0100040f  hdba: 0x010003fa
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 0(0x0) flag: 0x00 lock: 0 ckix: 0
ncol: 4 nnew: 1 size: 0
col  1: [ 2]  c1 1f

CHANGE #2 TYP:2 CLS:1 AFN:4 DBA:0x0100040f OBJ:76390 SCN:0x0000.00166a07 SEQ:1
OP:11.5 ENC:0 RBL:0
KTB Redo
op: 0x11  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F  xid:  0x0001.011.00000316    uba: 0x00c00152.0192.25
Block cleanout record, scn:  0x0000.00166a3f ver: 0x01 opt: 0x02, entries follow...
  itli: 1  flg: 2  scn: 0x0000.00166a07
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x0100040f  hdba: 0x010003fa
itli: 2  ispac: 0  maxfr: 4858
tabn: 0 slot: 0(0x0) flag: 0x00 lock: 2 ckix: 0
ncol: 4 nnew: 1 size: 0
col  1: [ 2]  c1 29
```

The above redo record consists of two changes:

Change #	Operation	Description
1	5.1	Undo for update (URP) operation
2	11.5	Redo for update (URP) operation

Column C7 is in a row piece in slot 0 on block 0x0100040f.

The above statement will generate supplemental data as follows:

```
Header (28 bytes)
  00000401 00070001 00000007 00001000 00000000 010003FC 00000003

Bytes 00-01: Number of columns (0)
Bytes 02-03: Unknown
Bytes 04-05: Column number of first column modified (7)
Bytes 06-07: Unknown
Bytes 08-0B: Column number of first column modified (7)
Bytes 0C-0D: Unknown
Bytes 0E-0F: Unknown
Bytes 10-13: Unknown
Bytes 14-17: DBA of first row piece (0x010003FC)
Bytes 18-21: Slot number of first row piece (3)
```

This test confirms that the final field does contain the slot number of the first row piece.

Note that only 16 bits are required to store the slot number. Therefore the remaining 16 bits may be used for other purposes (and 0 in this example) or may not be used at all.

Index Clusters

Supplemental logging is added for updates to table rows in index clusters. In this case the cluster key value is added to the redo log.

The supplemental logging consists of a 28 byte header, plus additional structures containing the cluster key value.

This example demonstrates supplemental logging for index-clustered rows. Minimal supplemental logging must be enabled as follows:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

This example uses the following objects:

```
CREATE CLUSTER cl1
(
  c1 NUMBER
)
SIZE 2048;

CREATE INDEX cl1_pk ON CLUSTER cl1;

CREATE TABLE t36
(
  c1 NUMBER,
  c2 NUMBER,
  c3 VARCHAR2(1000),
  c4 VARCHAR2(1000),
  c5 NUMBER
)
CLUSTER cl1 (c1);
```

The index cluster (CL1) contains one table (T36). Table T36 contains a couple of 1000 byte filler columns to ensure it occupies a few blocks.

The following PL/SQL block was used to populate table T36:

```
DECLARE
  PROCEDURE insert_row (p_key NUMBER,p_count NUMBER) IS
    l_str VARCHAR2(1000);
  BEGIN
    l_str := LPAD ('X',1000,'X');

    FOR f IN 1 .. p_count
    LOOP
      INSERT INTO t36 VALUES (p_key,f,l_str,l_str,f * 100);
    END LOOP;
  END;

BEGIN
  insert_row (1001,4);
  insert_row (1002,1);
  insert_row (1003,9);
  insert_row (1004,1);
END;
/
```

T36 contains the following initial values for the non-filler columns:

```
SQL> SELECT c1,c2,c5 FROM t36 ORDER BY c1,c2;

        C1         C2         C5
---------- ---------- ----------
      1001          1        100
      1001          2        200
      1001          3        300
      1001          4        400
      1002          1        100
```

```
      1003          1        100
      1003          2        200
      1003          3        300
      1003          4        400
      1003          5        500
      1003          6        600
      1003          7        700
      1003          8        800
      1003          9        900
      1004          1        100

15 rows selected.
```

The following statement was used to update a row in T36.

```
UPDATE t36 SET c5 = c5 + 1
WHERE c1 = 1003 AND c2 = 7;

COMMIT;
```

The above update generated the following redo:

```
REDO RECORD - Thread:1 RBA: 0x000039.00000005.0010 LEN: 0x0254 VLD: 0x05
SCN: 0x0000.001671c6 SUBSCN:  1 03/23/2013 00:48:35
(LWN RBA: 0x000039.00000005.0010 LEN: 0002 NST: 0001 SCN: 0x0000.001671c5)

CHANGE #1 TYP:0 CLS:33 AFN:3 DBA:0x00c00100 OBJ:4294967295 SCN:0x0000.00167172
SEQ:1 OP:5.2 ENC:0 RBL:0
ktudh redo: slt: 0x0019 sqn: 0x000003e3 flg: 0x0012 siz: 208 fbi: 0
            uba: 0x00c000cc.0143.0e    pxid:  0x0000.000.00000000

CHANGE #2 TYP:0 CLS:34 AFN:3 DBA:0x00c000cc OBJ:4294967295 SCN:0x0000.00167171
SEQ:2 OP:5.1 ENC:0 RBL:0
ktudb redo: siz: 208 spc: 1884 flg: 0x0012 seq: 0x0143 rec: 0x0e
            xid:  0x0009.019.000003e3
ktubl redo: slt: 25 rci: 0 opc: 11.1 [objn: 76394 objd: 76392 tsn: 4]
Undo type:  Regular undo        Begin trans    Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
             0x00000000  prev ctl uba: 0x00c000cc.0143.0c
prev ctl max cmt scn:  0x0000.001665a5  prev tx cmt scn:  0x0000.001665b9
txn start scn:  0xffff.ffffffff  logon user: 84  prev brb: 12583118  prev bcl: 0
BuExt idx: 0 flg2: 0
KDO undo record:
KTB Redo
op: 0x04  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: L  itl: xid:  0x0003.01b.000003cd uba: 0x00c00400.012a.11
                      flg: C---    lkc:  0     scn: 0x0000.00166f66
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x0100036f  hdba: 0x0100036a
itli: 2  ispac: 0  maxfr: 4858
tabn: 1 slot: 2(0x2) flag: 0x6c lock: 0 ckix: 0
ncol: 4 nnew: 1 size: -1
col  3: [ 2]  c2 08

CHANGE #3 TYP:2 CLS:1 AFN:4 DBA:0x0100036f OBJ:76392 SCN:0x0000.00166f66 SEQ:1
OP:11.5 ENC:0 RBL:0
KTB Redo
op: 0x11  ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F  xid:  0x0009.019.000003e3    uba: 0x00c000cc.0143.0e
Block cleanout record, scn:  0x0000.001671c6 ver: 0x01 opt: 0x02, entries follow...
  itli: 1  flg: 2  scn: 0x0000.00166f61
  itli: 2  flg: 2  scn: 0x0000.00166f66
KDO Op code: URP row dependencies Disabled
```

```
  xtype: XA flags: 0x00000000  bdba: 0x0100036f  hdba: 0x0100036a
itli: 2  ispac: 0  maxfr: 4858
tabn: 1 slot: 2(0x2) flag: 0x6c lock: 2 ckix: 0
ncol: 4 nnew: 1 size: 1
col  3: [ 3]  c2 08 02

CHANGE #4 MEDIA RECOVERY MARKER SCN:0x0000.00000000 SEQ:0 OP:5.20 ENC:0
session number   = 51
serial  number   = 171
transaction name =
version 186647296
audit sessionid 190103
Client Id =
login   username = US01
```

The above redo record consists of four changes:

Change #	Operation	Description
1	5.2	Allocate transaction slot in undo header
2	5.1	Undo for update (URP) operation
3	11.5	Redo for update (URP) operation
4	5.20	Media recovery marker

In the above redo the object ID differs from the data object ID. These are as follows:

Object ID	Owner	Object Name	Object Type
76392	US01	CL1	CLUSTER
76394	US01	T36	TABLE

The above statement will generate supplemental data as follows:

```
Header (28 bytes)
  00010C01 00050001 00000005 00001000 00000000 0100036F 00010002

Bytes 00-01: Number of columns (1)
Bytes 02-03: Unknown
Bytes 04-05: Column number of first column modified (5)
Bytes 06-07: Unknown
Bytes 08-0B: Column number of first column modified (5)
Bytes 0C-0D: Unknown
Bytes 0E-0F: Unknown
Bytes 10-13: Unknown
Bytes 14-17: DBA of cluster key (0x0100036F)
Bytes 18-19: Possibly table number of cluster key (1)
Bytes 20-21: Slot number of cluster key (2)

Column List Header (2 bytes)
  0001

Bytes 00-01: INTCOL# of column 1 (1)

Column Size List (2 bytes)
  0003

Bytes 00-01: Length of column 1 (3 bytes)

Column 1 Value (3 bytes)
  C2 0B 04

col  0: [ 3]  c2 0b 04                    # CLUSTER_KEY = 1003
```

The above column value is the value of the cluster key for the row that has been updated.