Handling Tables Without Primary Keys (PK) or Unique Indexes (UI) using KEYCOLS (Doc ID 1379932.1)

In this Document

APPLIES TO:

Oracle GoldenGate - Version 4.0.0 and later
Information in this document applies to any platform.

PURPOSE

This document describes how Oracle GoldenGate (OGG) works with tables lacking unique keys.

SCOPE

Helpful information for everybody replicating data without unique keys.

DETAILS

Oracle GoldenGate (OGG) capture (extract) does not have information if or how an application is using SQL to update rows.
If an application does an update with a where clause that updates five rows, each row is logged by the database as a single update, resulting in five individual updates. The information how these updates are done (e.g. by using a PL/SQL exec statement) is not available to GoldenGate.
When the replicat gets this data from a GoldenGate trail record, it first tries to identify this particular row. If a primary key or unique index exists, the replicat will use the columns that are defined as the key or index to locate the row and subsequently perform the update.
If no key or index exists, the replicat will by default use all the columns of that table as a key. Because the database records a single row update, the replicat will always attempt a single row update. When the Replicat tries to apply the update or delete, it actually uses all the columns in the where clause plus rownum = 1. This will guarantee the update or delete will only touch the first record that matches the where clause. If there are multiple rows that have exactly same values, then it doesn't matter which row will update or delete, as long as there is one that is touched.

Oracle GoldenGate makes it more efficient for users to update tables without PK or UI by having the KEYCOLS features. This allows the user to designate one or more columns as a pseudo key. That means there are business rules which state that this column or the combination of these columns will uniquely identify a row. As far as the database is concerned, the table remains without keys or indexes and if a mistake is made by the application or user to insert two identical rows the database will allow this.
However, there are potential issues for tables without PK or UI and solutions.

1. Initial loading.

Please consult these KM documents on how to initially load such tables:
Document 1350948.1 OGG can insert duplicate rows when there is no unique index primary key UI PK defined
Document 969550.1 Using SCN To Do The Initial Load From Oracle To Oracle Database with GoldenGate Versions before v10
Document 1276058.1 Oracle GoldenGate Best Practices: Instantiation from an Oracle Source Database

Note: If the table doesn't have any UI or PK, but it has columns that could not be used as keycols by GoldenGate, such as LOB, Long, user defined data type and etc., GoldenGate will use all the columns as keycols except those (Lob/long/UDT). In this case, even using the above KM documents to initially load the table, the integrity of the target table cannot be guaranteed especially when applications update those LOB/LONG/UDT columns, the replicat will update them to the wrong row if there are multiple rows with same values in the table.

2. Applications on source machine updating rows without using values from columns, such as using rowid.

Example:
If there are three rows in the source and target table mytable that have exactly the same values

COL1	COL2	COL3	COL4
A	B	C	D
A	B	C	D
A	B	C	D

In one transaction, the application updated those rows to below in three updates

COL1	COL2	COL3	COL4
A	B	C	X
A	B	C	Y
A	B	C	Z

But when the transaction comes to the trail, when Replicat picks it up, it might all be applied to the first row, and you will see this in target

COL1	COL2	COL3	COL4
A	B	C	Z
A	B	C	D
A	B	C	D

Since just the first row gets updated to ABCX/ABCY/ABCZ including commit, that will bring the table out of sync from the source even if it started with the same rows as the source.

Reason:

Not all four columns can be used as key (a sub set of 4 columns are lob/xml/udt..)
Example:
Col1 varchar2(10), Col2 varchar2(20), Col3 varchar2(30), Col4 **CLOB**.
There is no PK/UI and the first 3 columns will be used as key, because the CLOB column cannot be used as key column.
Initially, the three rows have the same values, A, B, C, D

In the source, col4 is changed toX,Y,Z, respectively for all 3 rows, and you get in the source:
A, B, C, X
A, B, C, Y
A, B, C, Z

In the target, following three updates will be run:
Update mytable set col1=A, col2=B, col3=C, col4=X where col1=A and col2=B and col3=C and rownum=1;
Update mytable set col1=A, col2=B, col3=C, col4=Y where col1=A and col2=B and col3=C and rownum=1;
Update mytable set col1=A, col2=B, col3=C, col4=Z where col1=A and col2=B and col3=C and rownum=1;

If more than one of above three updates are on the same row (it is very likely all three updates will be on the same row), it may cause the target data to be different from source data. E.g., if all three updates are on the same row (let's say the first row), you will get :
A, B, C, Z
A, B, C, D
A, B, C, D

If all four columns can be used as key then the replication will be successfully because in the target, following three updates will be run:
Update mytable set col1=A, col2=B, col3=C, col4=X where col1=A and col2=B and col3=C and col4=D and rownum=1;
Update mytable set col1=A, col2=B, col3=C, col4=Y where col1=A and col2=B and col3=C and col4=D and rownum=1;
Update mytable set col1=A, col2=B, col3=C, col4=Z where col1=A and col2=B and col3=C and col4=D and rownum=1;

This will successfully replicate the changes for all three columns.
A, B, C, X
A, B, C, Y
A, B, C, Z

How good are your KEYCOLS columns?

To make sure that your chosen columns are able to uniquely identify a row, the following Oracle database SQL statement can be used (other database types may have slightly different syntax):
Select , count(*)
From mytable
Group by <list of key columns, separated by commas☐>
Having count(*) >1;

This is a snapshot view and later duplicates may occur but at least you know that your data in the current table(s) fits your theory. Running this occasionally on the source and target database is recommended.

Performance

Generally GoldenGate replication for tables without Primary keys (PK) or Unique Indexes (UI) is not as efficient as for tables with unique indexes or keys. There are however ways to improve the performance and some of these are listed below:
1) Introduce Primary Keys or Unique Indexes for these tables. It is also good practice in database management irrespective of whether you are doing replication.
2) Choose columns that will uniquely identify the each row and use the GoldenGate KEYCOLS feature.
3) Batch deletes may cause replicat performance problem (if there is no index at all): because one (or a few) SQL statements in the source may result in many individual statements in target. The workaround is to add non-unique index on a relatively unique column(s).

Notes:

If you attempt to create a supplemental log group with more than 33 columns in it an ORA-02257 will be reported.
Please review How to Overcome the Limitation 33 Columns per Supplemental Log Group (Doc ID 466439.1)

REFERENCES

NOTE:1271578.1 - How to Handle Tables Without Primary Keys or Unique Indexes With Oracle GoldenGate
NOTE:1276058.1 - Oracle GoldenGate Best Practices: Instantiation from an Oracle Source Database
NOTE:1350948.1 - OGG Can Insert Duplicate Rows When There Is No Unique Index Primary Key UI PK Defined
NOTE:969550.1 - Using SCN To Do The Initial Load From Oracle To Oracle Database with GoldenGate Versions before v10
NOTE:466439.1 - How to Overcome the Limitation 33 Columns per Supplemental Log Group