30
Managing Transactions

Managing transactions include tasks such as setting transaction priority and automatically rolling back transactions.

- Priority Transactions
  The Oracle database allows transactions to be automatically rolled back and includes parameters to control this behavior.
- Automatic Transaction Quarantine
  Oracle Database quarantines, or isolates, the recovery of transactions that could potentially cause a system crash. These transactions must be manually resolved by the DBA so that row locks are released.

30.1 Priority Transactions

The Oracle database allows transactions to be automatically rolled back and includes parameters to control this behavior.

A row lock is a lock on a single row of a table. A transaction acquires a row lock for each row modified by one of the following statements: `INSERT`, `UPDATE`, `DELETE`, `MERGE`, and `SELECT ... FOR UPDATE`. The row lock exists until the transaction commits or rolls back. Transactions can hold row locks for a long duration in certain cases. For example, the application modifies some rows but doesn't commit or terminate the transaction because of an exception in the application. Traditionally, when a transaction is blocked on a rowlock by another transaction for a long time, it required the database administrator to manually terminate the blocking transaction by using the `ALTER SYSTEM KILL SESSION` command.

Starting with Oracle Database 23ai, the database provides parameters to control when and which transactions holding rowlocks can be automatically rolled back. Oracle database rolls back the transaction but the session stays alive. The application must acknowledge the automatic rollback of the transaction by issuing a `ROLLBACK` SQL statement.

Applications can specify the priority of their transactions. If a low priority transaction blocks a high priority transaction on rowlocks, Oracle database will automatically roll back the low priority transaction to let the high priority transaction(s) progress.

The database administrator can configure the time after which the low priority transaction is rolled back.

Note that if a transaction is holding a rowlock and not blocking any transaction, such a transaction is never rolled back.

- Using Priority Transactions
  Setting the parameters to automatically roll back transactions is detailed in this section.
- Monitoring Priority Transactions
  Fixed views provide the information to assist in monitoring transaction priority and wait targets.
- Priority Transaction Behavior
  The behavior of priority transactions must be understood for distributed and XA transactions.

- Priority Transaction Restrictions
  There are restrictions on the use of priority transactions.

Related Topics

- Terminating an Active Session

30.1.1 Using Priority Transactions

Setting the parameters to automatically roll back transactions is detailed in this section.

- Setting Transaction Priority
  Oracle Database provides session settings to control the transaction priority.
- Setting System-Level Wait Targets
  Oracle Database provides system parameters to control after what time a transaction holding row lock can be automatically rolled back.
- Acknowledging the Automatic Rollback
  The automatic rollback of the transaction must be acknowledged before its session can continue executing further SQLs. The acknowledgment can be provided by issuing a transaction rollback.
- Setting Priority Transaction Mode
  Priority Transaction supports two modes so that you can try out this feature before enabling it.
- Using Priority Transaction Mode to Determine System-Level Wait Targets
  Priority transaction mode can be used to help determine system-level wait targets.

30.1.1.1 Setting Transaction Priority

Oracle Database provides session settings to control the transaction priority.

Transaction priority is set at session level using `ALTER SESSION` command. Once the transaction priority is set, it will remain the same for all the transactions created in that session.

For example, to set the priority of all transactions in the current session to high, use the following command:

```
ALTER SESSION SET "txn_priority" = "HIGH";
```

The valid values for `txn_priority` are `LOW`, `MEDIUM`, and `HIGH`. All the transactions get a default priority of `HIGH`, that is, no transaction will be rolled back by default. If this parameter is modified after the transaction has started, then current transaction's priority will not be changed dynamically. The next transaction created in the session will use the updated priority.

If a `HIGH` priority transaction is blocked for a row lock, Oracle database can roll back the transaction that is holding the row lock only if the holder is `LOW` or `MEDIUM` priority.

If a `MEDIUM` priority transaction is blocked for a row lock, Oracle database can roll back the transaction that is holding the row lock only if the holder is `LOW` priority.

If a `LOW` priority transaction is blocked for a row lock, Oracle database will not attempt to roll back the transaction holding the row lock irrespective of its priority.

Oracle database never rolls back a `HIGH` priority transaction.

This parameter should be set by the application based on understanding the criticality of the transaction.

Related Topics

- TXN_PRIORITY

30.1.1.2 Setting System-Level Wait Targets

Oracle Database provides system parameters to control after what time a transaction holding row lock can be automatically rolled back.

`PRIORITY_TXNS_HIGH_WAIT_TARGET` and `PRIORITY_TXNS_MEDIUM_WAIT_TARGET` set the maximum time duration, in seconds, a transaction with priority `HIGH` and `MEDIUM` will wait before the database rolls back a lower priority transaction holding a row lock. The blocker transaction is rolled back but its corresponding session is not killed and stays alive. The application must acknowledge this automatic rollback by catching `ORA-63300` and issuing a `ROLLBACK` SQL statement. If `ROLLBACK` is not issued, then all the SQL statements in the session will keep receiving `ORA-63302`. There is no low priority wait target parameter provided since Oracle database doesn't roll back a blocker transaction if waiter's priority is `LOW`.

The priority transactions feature is only enabled when both the transaction priority and the wait target parameters are set. Setting the transaction priority with no wait target time does not enable the feature.

To set the parameter using the `ALTER SYSTEM` command, specify the parameters with the wait time values. In this example, if a `HIGH` priority transaction is blocked for at least 15 seconds on a row lock held by a medium or low priority transaction, the database will automatically attempt to roll back the blocking lower priority transaction.

```
ALTER SYSTEM SET priority_txns_high_wait_target = 15;
```

When a higher priority transaction is blocked by a lower priority transaction, the system waits for at least the specified time before rolling back the blocking transaction. The wait time may be longer than the target time specified when there are multiple blocked transactions trying to get the same row lock. For example, assume the default high priority wait target is set to 20 seconds. The following actions take place:

1. At time t1, transaction 1, a low priority transaction, locks a specific row.
2. Ten seconds later at time t2, transaction 2, a low priority transaction, attempts to lock the same row and waits.
3. Five seconds later at time t3, transaction 3, a high priority transaction, attempts to update the same row.

Assuming that no transaction performs a commit, the high priority transaction waits at least 20 seconds (from time t3) after which the first transaction is rolled back. After this, transaction 2 gets the row lock, since it has requested the row lock before transaction 3. So transaction 3 would wait for another 20 secs from the time transaction 2 got the row lock, after which transaction 2 is rolled back. Therefore, the wait target parameter values do not imply the maximum time a high priority waiter will wait before it gets the row locks.

Related Topics

- PRIORITY_TXNS_HIGH_WAIT_TARGET
- PRIORITY_TXNS_MEDIUM_WAIT_TARGET

30.1.1.3 Acknowledging the Automatic Rollback

The automatic rollback of the transaction must be acknowledged before its session can continue executing further SQLs. The acknowledgment can be provided by issuing a transaction rollback.

When a transaction is automatically rolled back, the currently executing SQL in an active session or the next SQL statement in an idle session will get `ORA-63300`. The subsequent SQL statements will throw `ORA-63302` until a rollback is issued. Therefore, the application logic must be structured to catch both of the two errors `ORA-63300` and `ORA-63302`, and then issue the rollback.

The following table lists the available methods to acknowledge the rollback:

Table 30-1 Available Rollback Methods for Priority Transactions

Transaction Type/ Client	SQL*Plus and SQLcl	OCI	JDBC
Local and Distributed Transactions	Use the `ROLLBACK` SQL statement	Use the `OCITransRollback()` function, or `OCIStmtPrepare` and `OCIStmtExecute` of `ROLLBACK` SQL statement	Use `connection.rollback()`, or execute the rollback statement in JDBC using `connection.createStatement().execute("rollback")`
XA Transactions	Execute `DBMS_XA.XA_END()` followed by `DBMS_XA.XA_ROLLBACK()`. If the XA transaction is suspended, then execute `DBMS_XA.XA_ROLLBACK()` only.	Execute `xaoend()` followed by `xaorollback()`. If the XA transaction is suspended, then execute `xaorollback()` only.	Execute `xa_resource.end()` followed by `xa_resource.rollback()`, where xa_resource is of type XAResource. If the XA transaction is suspended, then execute `xa_resource.rollback()` only.

30.1.1.4 Setting Priority Transaction Mode

Priority Transaction supports two modes so that you can try out this feature before enabling it.

The default mode for priority transactions is ROLLBACK. In this mode, if `PRIORITY_TXNS_HIGH_WAIT_TARGET` and `PRIORITY_TXNS_MEDIUM_WAIT_TARGET` are appropriately configured, transactions that are holding row locks and blocking higher priority transactions would be automatically rolled back and allow higher priority waiting transactions to progress.

The purpose of `TRACK` mode is for database administrators to try out the priority transactions feature. In `TRACK` mode, the database will increment statistics in `V$SYSSTAT` (discussed in Setting System-Level Wait Targets), reflecting how many transactions this feature would have rolled back, instead of actually rolling back any transactions. Applications can use this mode to tune the right wait target value before switching to `ROLLBACK` mode.

To set priority transaction mode to `TRACK`, use the following command:

```
ALTER SYSTEM SET "priority_txns_mode"="TRACK";
```

To set priority transaction mode to `ROLLBACK`, use the following command:

```
ALTER SYSTEM SET "priority_txns_mode"="ROLLBACK";
```

Related Topics

- PRIORITY_TXNS_MODE

30.1.1.5 Using Priority Transaction Mode to Determine System-Level Wait Targets

Priority transaction mode can be used to help determine system-level wait targets.

To aid in setting the appropriate values for `PRIORITY_TXNS_HIGH_WAIT_TARGET` and `PRIORITY_TXNS_MEDIUM_WAIT_TARGET`, you can set `PRIORITY_TXNS_MODE` to `TRACK` and monitor the row lock contention wait event time.

Run your regular workload in `TRACK` mode for a few hours (or whatever is appropriate) and monitor the time transactions of a certain priority typically wait for the row lock. For example, if you observe that your HIGH priority transactions typically wait for a maximum of 10 seconds on the row lock, then it is recommended to set the value of `PRIORITY_TXNS_HIGH_WAIT_TARGET` to a value above 90 seconds so that Oracle Database doesn't rollback any legitimate transactions holding the row lock while doing meaningful operations on the database. After determining the appropriate values for these parameters, you can turn off the `TRACK` mode and switch to the `ROLLBACK` mode, configure the system-level wait target parameters with these values, and start using priority transactions.

When there is a contention for the row lock, the transactions waiting for the row lock wait on a common wait event `enq: TX - row lock contention`. With priority transactions enabled by setting both the `txn_priority` parameter for transactions and `wait_target` parameter for the system, waiting transactions would wait on the wait events based on the priority of the waiting transaction.

Table 30-2 Wait Events

Waiting Transaction's Priority	Wait Event
HIGH	`enq:TX - row lock contention (HIGH pri)`
MEDIUM	`enq: TX - row lock contention (MEDIUM pri)`
LOW	`enq: TX - row lock contention (LOW pri)`

In the example below, session (`sid 204`) has a HIGH priority transaction holding the row lock. You can see other transactions wanting the same row lock waiting on different wait events based on their priority.

```
SQL> SELECT TO_CHAR(xidusn) || '.' || TO_CHAR(xidslot) || '.' ||
TO_CHAR(xidsqn) AS transaction_id, sid, event, seconds_in_wait,
blocking_session
     FROM   v$session, v$transaction
     WHERE  event LIKE '%enq%' AND v$session.saddr = v$transaction.ses_addr;

TRANSACTION_ID  SID     EVENT                                SECONDS_IN_WAIT
```

```
BLOCKING_SESSION
--------------- ------ ---------------------------------- ---------------
---------------
2.17.1619          187 enq: TX - row lock (HIGH priority)
361               204
5.32.1557           51 enq: TX - row lock (LOW priority)
359               204
```

Related Topics

- PRIORITY_TXNS_MODE

30.1.2 Monitoring Priority Transactions

Fixed views provide the information to assist in monitoring transaction priority and wait targets.

Two columns are available in `V$TRANSACTION` to aid in monitoring transactions. `TXN_PRIORITY` shows the transaction priority and `PRIORITY_TXNS_WAIT_TARGET` shows the wait target for the transaction specified in seconds.

Alerts are shown in the alert log whenever a transaction is terminated. For example:

```
Transaction (sid: 203, serial: 39661, xid: 7.23.1161, txn_priority: "LOW")
terminated by transaction (sid: 204, serial: 9266, xid: 13.15.3, txn_priority:
"HIGH") because of the parameter "priority_txns_high_wait_target = 10"
```

`TXN_PRIORITY` cannot be set for a scheduler job. If it is set for a scheduler job, error `ORA-63303` is thrown and reported in the alert log.

- Statistics Incremented in ROLLBACK Mode
  Specific statistics are incremented for Priority Transactions when in ROLLBACK mode.
- Statistics Incremented in TRACK Mode
  Specific statistics are incremented for Priority Transactions when in TRACK mode.

Related Topics

- V$TRANSACTION

30.1.2.1 Statistics Incremented in ROLLBACK Mode

Specific statistics are incremented for Priority Transactions when in ROLLBACK mode.

The following statistics are incremented only in `ROLLBACK` mode. These statistics are incremented each time a transaction is rolled back because of a higher priority waiter transaction.

```
SQL> select name
     from   V$SYSSTAT
     where  name like '%txns rollback%';

NAME
----------------------------------------------------------------
txns rollback priority_txns_high_wait_target
txns rollback priority_txns_medium_wait_target
```

For example, if a MEDIUM or LOW priority transaction is rolled back because of a HIGH priority transaction, then `txns rollback priority_txns_high_wait_target` will be incremented.

30.1.2.2 Statistics Incremented in TRACK Mode

Specific statistics are incremented for Priority Transactions when in TRACK mode.

The following statistics are incremented only in `TRACK` mode. These statistics are incremented each time a transaction would have rolled back because of a higher priority waiter transaction.

```
SQL> select name
     from   V$SYSSTAT
     where  name like '%txns track mode%';

NAME
----------------------------------------------------------------
txns track mode priority_txns_high_wait_target
txns track mode priority_txns_medium_wait_target
```

For example, if a MEDIUM or LOW priority transaction would have rolled back because of a HIGH priority transaction, then `txns track mode priority_txns_high_wait_target` will be incremented.

Related Topics

- V$TRANSACTION

30.1.3 Priority Transaction Behavior

The behavior of priority transactions must be understood for distributed and XA transactions.

- Behavior of Priority Transactions for Distributed Transactions
  This section describes the behavior of priority transactions for distributed transactions.
- Behavior for XA Transactions
  This section describes the behavior of priority transactions for XA transactions.

30.1.3.1 Behavior of Priority Transactions for Distributed Transactions

This section describes the behavior of priority transactions for distributed transactions.

Participant Priority

For distributed transactions, the priority of the coordinator branch is inherited by all the participants (remote branches). That is, if a distributed transaction `txn1` starts on `db1` and has a participant `txn2` on `db2`, then `txn2` inherits `txn1`'s `TXN_PRIORITY` parameter value.

Acknowledgment Behavior

The automatic rollback of distributed transactions is acknowledged the same way as local transactions. However, the automatic rollback acknowledgment behavior of distributed transactions differs from local transactions in the following ways:

- When a remote branch of a distributed transaction is automatically rolled back, the coordinator session continues normally until a SQL statement is issued from the

coordinator to the automatically rolled back remote branch. At that point, the coordinator will throw ORA-63300 and ORA-63302 until a rollback acknowledgment is received.

- When the coordinator, or local, branch of a distributed transaction is automatically rolled back, any new SQL statements in the coordinator session will immediately throw ORA-63300 and ORA-63302. This behavior is the same as a local transaction.

30.1.3.2 Behavior for XA Transactions

This section describes the behavior of priority transactions for XA transactions.

Once an XA transaction is automatically rolled back, then the XA client will receive the XAER_RMFAIL error for any subsequent XA statement until the rollback acknowledgment is received. Clients are expected to handle XAER_RMFAIL and then execute the XA_END and XA_ROLLBACK statements.

If the XA transaction was already in a suspended state due to XA_END with the TMSUSPEND flag set, then the acknowledgment requires only executing the XA_ROLLBACK statement.

30.1.4 Priority Transaction Restrictions

There are restrictions on the use of priority transactions.

TXN_PRIORITY cannot be set for a scheduler job, otherwise error ORA-63303 is thrown. The error is reported in the alert log.

30.2 Automatic Transaction Quarantine

Oracle Database quarantines, or isolates, the recovery of transactions that could potentially cause a system crash. These transactions must be manually resolved by the DBA so that row locks are released.

About Redo Application

Database buffers in the buffer cache in the SGA are written to disk only when necessary, using a least-recently-used (LRU) algorithm. Because of the way that the database writer process uses this algorithm to write database buffers to datafiles, datafiles may contain some data blocks modified by uncommitted transactions and some data blocks missing changes from committed transactions.

Crash Recovery and Instance Recovery

Crash recovery is used to recover from a failure either when a single-instance database crashes or all instances of an Oracle Real Application Clusters database crashes. Instance recovery refers to the case where a surviving instance recovers a failed instance in an Oracle Real Application Clusters database.

The goal of crash and instance recovery is to restore the data block changes located in the cache of the dead instance and to close the redo thread that was left open. Instance and crash recovery use only online redo log files and current online datafiles.

Two potential problems can result if an **instance failure** occurs:

- Data blocks modified by a transaction might not be written to the datafiles at commit time and may only appear in the redo log. Therefore, the redo log contains changes that must be reapplied to the database during recovery.

- After the roll forward phase, the datafiles may contain changes that had not been committed at the time of the failure. These uncommitted changes must be rolled back to ensure transactional consistency. These changes were either saved to the datafiles before the failure or introduced during the roll forward phase.

To solve this dilemma, two separate steps are generally used by Oracle for a successful recovery of a system failure: rolling forward with the redo log (cache recovery) and rolling back with the rollback or undo segments (transaction recovery).

Cache Recovery

The online redo log is a set of operating system files that record all changes made to any database buffer, including data, index, and rollback segments, whether the changes are committed or uncommitted. All changes to Oracle blocks are recorded in the online log.

The first step of recovery from an instance or disk failure is called cache recovery or rolling forward and involves reapplying all of the changes recorded in the redo log to the datafiles. Because rollback data is also recorded in the redo log, rolling forward also regenerates the corresponding undo segments.

Rolling forward proceeds through as many redo log files as necessary to bring the database forward in time. Rolling forward usually includes online redo log files (instance recovery or media recovery) and may include archived redo log files (media recovery only).

After rolling forward, the data blocks contain all committed changes. They may also contain uncommitted changes that were either saved to the datafiles before the failure or were recorded in the redo log and introduced during cache recovery.

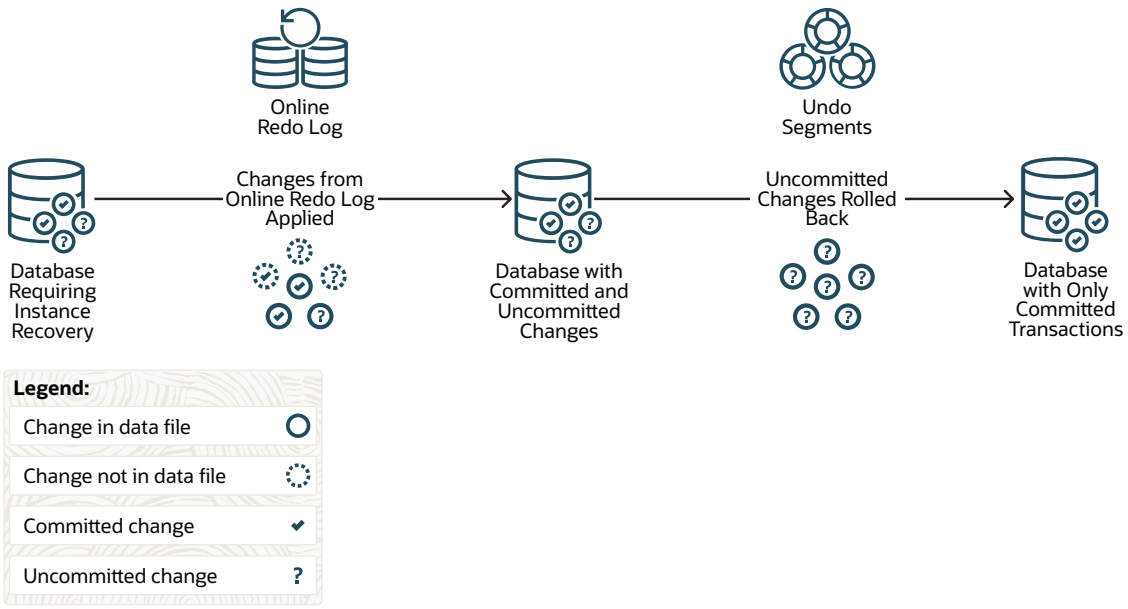
Transaction Recovery

Undo tablespaces (in automatic undo management mode) contain undo segments that record the before-image of changes to the database. In database recovery, the undo blocks inside the undo segments roll back the effects of uncommitted transactions previously applied by the rolling forward phase.

After the roll forward, any changes that were not committed must be undone. Oracle applies undo blocks to roll back uncommitted changes in data blocks that were either written before the crash or introduced by redo application during cache recovery. This process of rolling back uncommitted transactions in the database is called transaction recovery.

The following figure illustrates rolling forward and rolling back, the two steps necessary to recover from any type of system failure.

Figure 30-1 Rolling Forward and Rolling Back



Failure During Transaction Recovery

Transaction recovery can fail due to the following reasons:

- Physical data corruption of database blocks (ORA-01578, ORA-28304)
- Logical data corruption (ORA-00600)
- Memory corruption (ORA-00602, ORA-07445)
- State Corruptions (ORA-00600)

A failure during transaction recovery can be irrecoverable to the entire database instance and bring down the entire container database (CDB) including its pluggable databases. Inability to recover all the transactions in the system leads to rowlocks being held by unrecovered transactions for longer. This severely impacts critical business operations.

Starting with Oracle Database 23ai, transactions that fail to recover are quarantined and left un-recovered until the DBA can resolve the issue. This increases the availability of the database. The Database Developer is notified about the quarantined transaction and must take immediate action so that the row locks held by quarantined transactions can be released.

Transaction quarantines are maintained in a persistent data dictionary table inside the database. Therefore, you can manage quarantines from any RAC instance in the database.

When a DML operation tries to access rows locked by a quarantined transaction error `ORA-60451` will be raised as the DML operation cannot be executed while the rows are still locked.

Quarantined Transaction and Replication

Since Oracle Data Guard uses logical replication, quarantine metadata is not replicated to the standby server when using Oracle Data Guard. Therefore, contents of transaction quarantine views, such as `DBA_QUARANTINED_TRANSACTIONS`, on the standby server may be different than the entries on the primary server.

When running with Active Data Guard (ADG), the replication is physical which means that for the transaction quarantine feature, both the dead transaction and the catalog representation of the quarantine will be replicated to the standby database.

- Monitoring Quarantined Transactions
  Alerts and data dictionary views warn the database developer of quarantined transactions.
- Resolving Quarantined Transactions
  The database developer will be alerted when a transaction quarantine is generated. Quarantines should be monitored and resolved quickly to prevent row locks from being held for a long time.
- Dropping Quarantined Transactions
  After the issue related to the quarantine has been fixed, the quarantine must be dropped.
- Transaction Quarantine Escalation
  When the transaction quarantine limit is reached (default of 3) for a PDB, it is automatically shut down on all RAC instances so that the database developer can resolve the issue. The other PDBs in the CDB are not affected.

30.2.1 Monitoring Quarantined Transactions

Alerts and data dictionary views warn the database developer of quarantined transactions.

Oracle Database warns DBAs of quarantined transactions in several ways, which include:

- **ALERT_QUE** - the transaction quarantine alert is sent to the persistent alert queue `SYS.ALERT_QUE`. This alert is automatically displayed in the data dictionary views `DBA_OUTSTANDING_ALERTS` and `DBA_ALERT_HISTORY`, as well as Enterprise Manager Cloud Control and the AWR report.
- **Attention log** - introduced in Oracle 21c, the attention log contains information about critical and highly visible database events. Starting with Oracle Database 23ai, it includes the transaction quarantine information as well.
- **Alert log** - an incident will be generated for the internal error and traced in the alert log. The DBA can monitor the quarantine incident in `V$DIAG_ALERT_EXT`.

Views named `DBA_QUARANTINED_TRANSACTIONS` and `CDB_QUARANTINED_TRANSACTIONS` monitor all active quarantined transactions. These views provides all the necessary information to resolve the quarantine.

Table 30-3 DBA_QUARANTINED_TRANSACTIONS View Columns

Column	Datatype	Null?	Description
USN	NUMBER	Not Null	Undo segment number of the quarantined transaction.
SLT	NUMBER	Not Null	Slot number of the quarantined transaction.
SQN	NUMBER	Not Null	The sequence number of the quarantined transaction.
UNDO_TSN	NUMBER		The undo tablespace number for the quarantined transaction.
TXN_START_SCN	NUMBER		Start SCN of the quarantined transaction.

Table 30-3 (Cont.) DBA_QUARANTINED_TRANSACTIONS View Columns

Column	Datatype	Null?	Description
INCIDENT_TIME	VARCHAR2(64)		Identifies the timestamp when the incident happened.
REASON	VARCHAR2(256)		The reason why this transaction failed to recover.
TRACE_FILE_NAME	VARCHAR2(4096)		The trace file name that contains the reason and diagnosability information for this transaction's recovery failure.
UBA_RDBA	NUMBER		Block number of the current undo block being applied for rollback.
UBA_SQN	NUMBER		Undo block sequence number.
UBA_RECORD_NUMBER	NUMBER		Undo record number.
UNDO_RECORD_OBJN	NUMBER		Dictionary object number of the object (OBJN).
UNDO_RECORD_OBJD	NUMBER		Dictionary object number of the segment that contains the object (OBJD).
PREV_UNDO_BLOCK_DBA	NUMBER		Previous undo block address which was used to rollback.
DATA_BLOCK_TSN	NUMBER		Tablespace ID for the object.

The view DBA_QUARANTINED_TRANSACTIONS view can be joined with GV$TRANSACTION and GV$FAST_START_TRANSACTIONS to get the details of the transaction and its recovery progress. Note that GV$TRANSACTION will lose its information on a database instance restart because fixed views are not persistent. Since transaction recovery begins after a database instance restart, GV$TRANSACTION shows the progress of any active transaction recovery even after a database restart.

30.2.2 Resolving Quarantined Transactions

The database developer will be alerted when a transaction quarantine is generated. Quarantines should be monitored and resolved quickly to prevent row locks from being held for a long time.

Quarantines can be monitored using DBA_QUARANTINED_TRANSACTIONS. The REASON column of the view shows why the transaction was quarantined. For example:

```
SQL> select usn, slt, sqn, reason, undo_record_objn
     from   dba_quarantined_transactions;

   USN    SLT    SQN                 REASON    UNDO_RECORD_OBJN
------ ------ ------ ---------------------- -------------------
```

```
6     18     10    ORA-00600[ktubko_1]              73646
7     20     13              ORA-28304              73650
```

Once the reason for the transaction quarantines has been identified (`ORA-00600[ktubko_1]` and `ORA-28304` in the example above), then refer to the Primary MOS note for Automatic Transaction Quarantine (Doc ID 3005962.1) where detailed instructions are provided for how to resolve the different causes of transaction quarantines.

30.2.3 Dropping Quarantined Transactions

After the issue related to the quarantine has been fixed, the quarantine must be dropped.

Transaction Recovery cannot be retried until the issue concerning the transaction quarantine is fixed. Therefore, after fixing the quarantine with the corrective action, the quarantine must be manually dropped for transaction recovery to restart for the quarantined transaction. The following DDL syntax can be used to drop the quarantine:

```
ALTER DATABASE DROP TRANSACTION QUARANTINE
<xid_undo_seg_no> <xid_slot_no> <xid_sequence_no>;
```

where

- `xid_undo_seg_no` is the undo segment number of the quarantined transaction (`USN` column of view `DBA_QUARANTINED_TRANSACTIONS`)
- `xid_slot_no` is the slot number of the quarantined transaction (`SLT` column of view `DBA_QUARANTINED_TRANSACTIONS`)
- `xid_sequence_no` is the sequence number of the quarantined transaction (`SQN` column of view `DBA_QUARANTINED_TRANSACTIONS`)

For example, to drop the quarantine for xid 8.20.275, use command:

```
ALTER DATABASE DROP TRANSACTION QUARANTINE 8 20 275;
```

30.2.4 Transaction Quarantine Escalation

When the transaction quarantine limit is reached (default of 3) for a PDB, it is automatically shut down on all RAC instances so that the database developer can resolve the issue. The other PDBs in the CDB are not affected.

Transaction quarantine is designed to help in cases when the failure, such as memory, data, or state corruption, is confined to a single transaction. That is, the inactive transaction that fails to recover is quarantined, other inactive transactions can be recovered, and there's no need to shut down the PDB or the CDB.

When failures happen across multiple transactions or span the entire PDB, such as physical corruption of multiple blocks, a PDB SGA corruption, or a logical data corruption due to an internal error, quarantining the failed inactive transaction recovery may or may not help. It depends on whether the root cause for those failures is the same or not, because recovering other inactive transactions might run into the same issue. The system keeps on running in an inconsistent state even after quarantining a few transactions. It can be dangerous when the failure is due to logical data corruption, because it spreads over time. To prevent this from happening, there is a transaction quarantine limit of three (3), after which the quarantine is escalated to the database level and the PDB will be terminated using `shutdown abort` if archive logging is enabled for the PDB and it is feasible to shut down the PDB. Transaction

recovery for the PDB is automatically disabled so that the database developer can correct problems on the next PDB startup.

When an escalation occurs, perform the following steps:

1. Open the PDB.
2. Query the view DBA_QUARANTINED_TRANSACTIONS to get information about the quarantined transactions.
3. For each quarantined transaction in the database, resolve the cause of the transaction quarantine (Resolving Quarantined Transactions) and then drop the transaction quarantine (see Dropping Quarantined Transactions).
4. Enable transaction recovery for the PDB.

To enable transaction recovery, use the command:

```
ALTER SYSTEM SET TRANSACTION_RECOVERY=ENABLED sid='*';
```

The `SCOPE` clause is not necessarily required. The default values for `SCOPE` are:

- For PDBs, the default value is `SCOPE=BOTH`.
- For CDB$ROOT, if a server parameter file was used to start the database, then the default is `SCOPE=BOTH`. If a parameter file was used to start the database, then the default is `SCOPE=MEMORY`.

These default values for `SCOPE` will re-enable transaction recovery for automatic transaction quarantine.

To determine if transaction quarantines were escalated to the PDB, alerts are published to all the alert channels described in Monitoring Quarantined Transactions (SYS.ALERT_QUE, Attention log, and Alert log).