

# **Oracle Distributed Transactions**

**Written and presented  
by Joel Goodman**



## About Me



- Email: [Joel.Goodman@oracle.com](mailto:Joel.Goodman@oracle.com)
- Blog: [dbatrain.wordpress.com](http://dbatrain.wordpress.com)
- EMEA Curriculum and Certification Specialist
- Global DBA Certification Exam Development Team Leader
- Global DBA Curriculum Quality Review Team
- OCP DBA – 7.3 to 12c
- OCM DBA – 9i to 11g
- Member of Oak Table
- Oracle University 1997 to Present
- Oracle Support 1994 to 1996
- IT Training Mainframe Technology 1986 to 1993
- Software Development and Consultancy 1983 to 1986
- Application Development and Support 1976 to 1982

# Mini-Lesson Objectives

- **Connectivity Overview**
- **Distributed Database Characteristics**
- **Remote and Distributed Queries**
- **Remote and Distributed Transactions**
- **Distributed Transaction Branches and RAC**
- **Two-Phase In-Doubt Transactions**
- **The RECO Process In-Doubt Resolution**
- **Manual In-Doubt Resolution**
- **Data Dictionary Entries and Cleanup**
- **2 Phase Commit Crash Recovery**
- **Distributed Systems Recovery**

# Distributed Database Architecture

- **A distributed database system allows users to access data from local and remote databases.**
- **The two types of distributed database systems are:**
  - **Homogenous distributed database systems**
  - **Heterogeneous distributed database systems using Heterogeneous Services commonly referred to as Gateways**

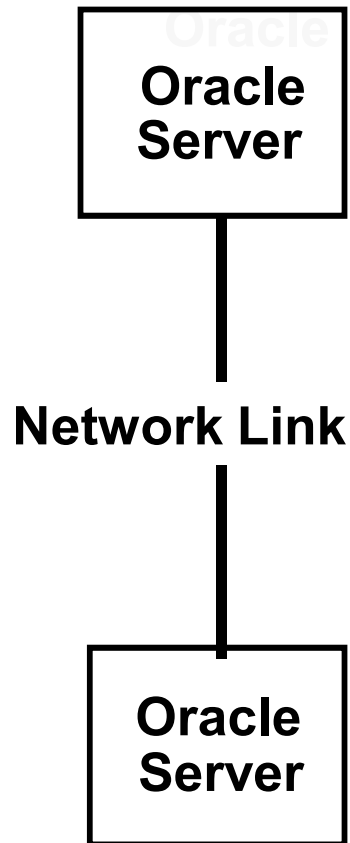
# Oracle Distributed Databases

- **Local autonomy**
- **No reliance on a central site**
- **Continuous operation**
- **Location independence**
- **Fragmentation independence**
- **Replication independence**

# Oracle Distributed Database

- **Distributed query processing**
- **Distributed transaction management**
- **Hardware independence**
- **Operating system independence**
- **Network independence**
- **DBMS independence**

# Server-Server Connectivity



```
SQL> CREATE DATABASE LINK  
ANGEL CONNECT TO HR  
IDENTIFIED BY HR USING  
'ANGEL' ;
```

```
SQL> SELECT * FROM  
EMPLOYEES@ANGEL ;
```

**Was Covered in the Database  
Links Master Class**

# Remote and Distributed Systems Concepts

- **Remote Queries**
- **Distributed Queries**
- **Remote Transactions**
- **Distributed Transactions**
- **Transaction Branches**
- **Single Phase Commit**
- **Two Phased Commit**
- **Commit Point Site**
- **Global System Change Numbers**
- **In-Doubt Transaction Resolution**



# Remote Queries

- **Session is connected to a local instance and a database link is used to query data from one remote database instance.**
- **The local instance sends the entire query to the remote database instance for processing**
- **Query may access one or more tables, or views possibly using synonyms**
- **The remote instance returns data to the local instance**

## Distributed Queries

- **Session is connected to a local instance and database links are used to query data from two or more database instances possibly including the local one**
- **The query is always executed from the local database instance**
- **The local instance decomposes the query into subqueries to be sent to each remote database instance.**
- **The local instance retrieves data from remote instances and performs any necessary post-processing.**

## Remote Transactions

- **Session is connected to a local instance and a DB link is used to insert, update or delete data using one instance of a remote database.**
- **The local instance sends the entire DML statement to the remote database instance for processing.**
- **The statement may access one or more tables, or views possibly using synonyms**
- **The remote instance returns DML results to the local instance**
- **Single Phase Commit is used at transaction end**

# Distributed Transactions

- **Session is connected to a local database instance and DB links used to perform DML on tables in two or more of the databases**
- **Only one instance used for each remote database**
- **Multiple DMLs may be sent to any of the databases involved in the transaction**
- **The local instance retrieves the status from the remote instances for each DML**
- **Two Phased Commit is used to Commit or Rollback the transaction guaranteeing consistency across all the databases involved in the transaction**

# Distributed Transactions

- The following DDL and DML commands are supported in distributed transactions:
- supported in distributed transactions:
  - CREATE TABLE AS SELECT
  - DELETE
  - INSERT
  - LOCK TABLE
  - SELECT
  - SELECT FOR UPDATE

# Distributed Transaction Locking

- Normal Deadlock detection mechanism is not available when two or more instances are involved
- Use `DISTRIBUTED_LOCK_TIMEOUT` parameter to specify the maximum time in seconds that the instance waits for locked resources
- Defaults to 60 seconds

## Distributed Transaction Branches and RAC

- In RAC all transaction branches involving the same database must be connected to the same instance
- Instance specific DB Links from a local Instance permitting connection to two or more remote instances of the same database leads to errors when doing commit or rollback of a distributed transaction
- DB Links from one RAC instance to another when used to try a “distributed” transaction leads to errors when doing commit or rollback
- Not protected by the RAC services model and DTP=TRUE.

# Single Phase Commit

- **Used for:**
  - **Single Database Commits and Rollbacks**
  - **Remote Transactions**
  - **Tightly coupled transaction with XA**
- **The local instance sends a commit request to a single remote instance**
- **The remote instance either commits or rolls back**
- **Result of the request at the remote instance is returned to the local instance**
- **DB Links use loosely coupled transactions and require 2 phase commit**



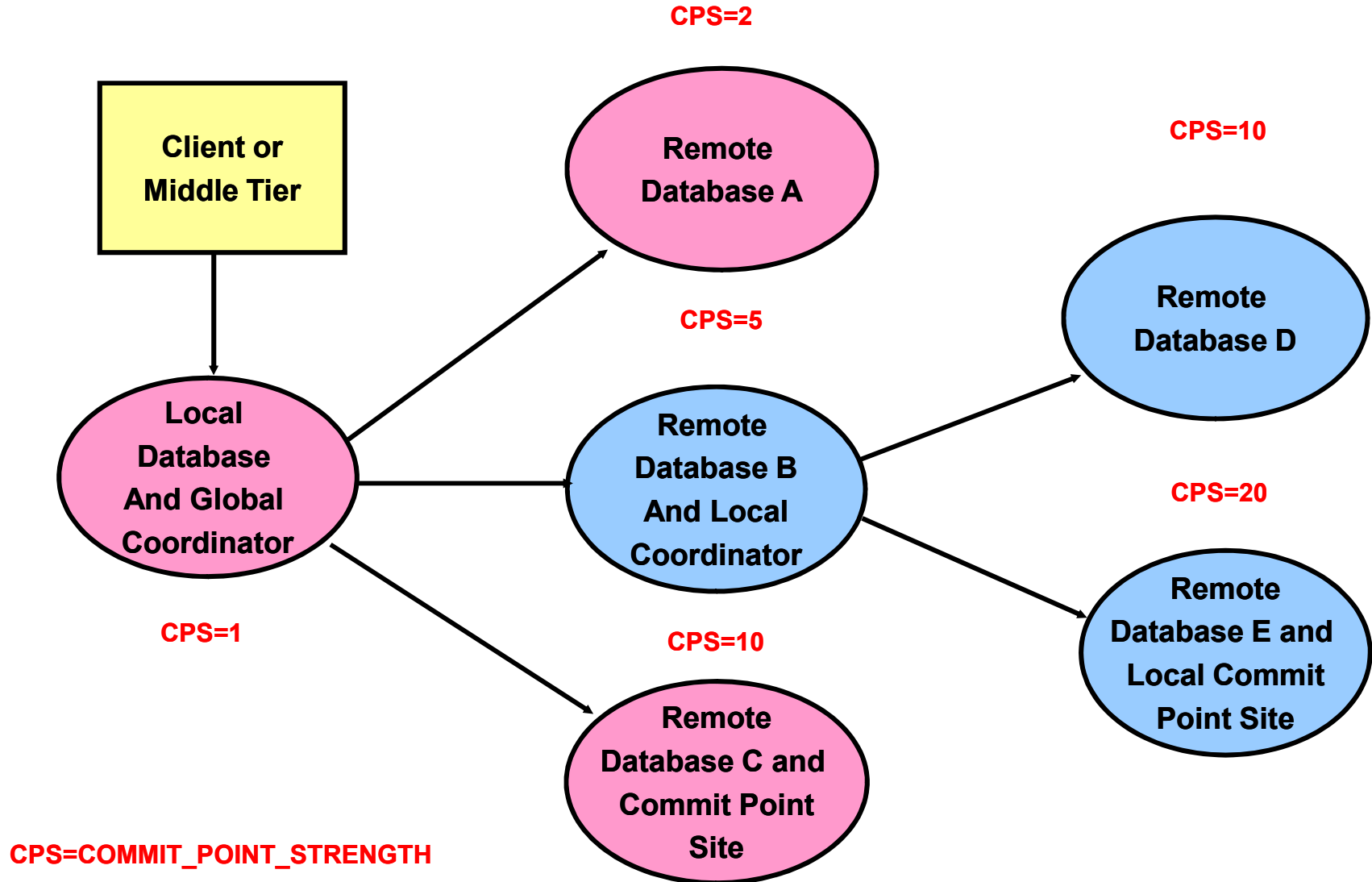
## Two Phase Commit

- Each database instance involved in a two phase commit operation performs one or more roles based on its position in the Session Tree:
  - **Client** - Issues transactional requests over DB Links to one or more Servers
  - **Database Server** - Receives transactional requests over a DB Link
  - **Global Coordinator** – Originating node for the distributed transaction. Sends prepare, commit or rollback statements to other adjacent nodes
  - **Local Coordinator** – References nodes which are non-adjacent to the Global Coordinator and coordinates those nodes to prepare, commit or rollback.
  - **Commit Point Site** – Node which is first to commit as part of the two-phase commit process

## The Commit Point Site

- In a two phase commit process, one site or node is the designated “**Commit Point Site**”
- This is determined by the values of the `COMMIT_POINT_STRENGTH` parameter in all the database instances involved in the transaction which are adjacent to and include the **Global Coordinator**
- The values range from 0 to 255 and if not set then the is determined by the software
- The `COMMIT_POINT_STRENGTH` parameter in non-adjacent nodes is only used to resolve who will be the recursive “**Commit Point Site**” when a **Local Coordinator** is the main “**Commit Point Site**”

# Distributed Transaction Roles



## Global System Change Numbers

- **Determined by the highest SCN from amongst all the databases participating in the distributed transaction**
- **The commit global SCN is sent to all the databases participating in the distributed transaction**
- **Each database also has its own local SCN and this will equal the global SCN one of the databases**
- **If failures occur during two phased commit, the Global Commit SCN is used to coordinate recovery so that all the databases are consistent with each other**

# Two-Phase Commit Phases

- **There are three steps in Two-Phase Commit processing:**
  - **Prepare**
  - **Commit**
  - **Forget**

# The Prepare Phase

- **Steps in the Prepare phase:**
  - **Global Coordinator** sends a request to each instance except for the **Commit Point Site** , asking for a guarantee that a commit or rollback request at a later stage will be done if requested even if an intervening instance failure occurs
  - When asked to `PREPARE` a site does the following:
    - Replies with “`READ ONLY`” if no updates done on node
    - Locks all tables in the transaction for **both read and write**
    - Flushes Redo to disk to guarantee ability to commit or rollback later when asked even if the instance fails after this point
    - Exchanges SCNs with all involved Instances to determine the Global Commit SCN for the transaction
    - If a site is a **Local Coordinator** then it recursively issues `PREPARE` commands to its own adjacent nodes
  - The **Commit Point Site** is not asked to prepare. If all sites prepare then it will be asked to commit. If at least one site rejects the prepare, then it will be asked to rollback

# The Commit Phase

- **Steps in the Commit phase when committing:**
  - The **Global Coordinator** sends a commit request to the **Commit Point Site**
  - If the **Commit Point Site** is also a **Local Coordinator** then it must recursively prepare all but one of its own adjacent nodes and then issue a commit to its local **Commit Point Site**
  - The **Commit Point Site** commits, releases locks and forgets about the transaction removing it from the dictionary
  - The **Global Coordinator** sends commit requests to adjacent prepared nodes
  - If any adjacent nodes are **Local Coordinators** then commits are propagated to their adjacent nodes
  - These nodes then commit, release locks and signal that they have committed
- Note that “**READ ONLY**” sites are not asked to commit

# The Commit Phase

- **Steps in the Commit phase when Rolling Back:**
  - The **Global Coordinator** sends a commit request to the **Commit Point Site**
  - If the **Commit Point Site** is also a **Local Coordinator** then it must recursively prepare all but one of its own adjacent nodes and then issue a commit to its local **Commit Point Site**
  - If any site is unable to commit for any reason, then it rolls back, releases locks and propagates the rollback to the **Local Coordinator** or **Global Coordinator** as appropriate
  - Any **Local Coordinators** sends rollback requests to adjacent subordinate prepared nodes and back to the **Global Coordinator**
  - The **Global Coordinator** then rolls back and sends rollback requests to all the adjacent nodes



# The Forget Phase

- **Steps in the Forget Phase:**
  - Any site that is the **Global Coordinator** updates the data dictionary to remove reference to the transaction when all have finished committing or rolling back
  - Any site that is not a coordinator may forget the transaction when it has either committed or rolled back

# In-Doubt Transaction Failure

- Failure Reported During Commit

```
SQL> Commit;
```

```
ORA-02054 transaction 1.44.99 in-doubt
```

# Resolving In-Doubt Transactions

- **In Doubt Transaction may be caused by:**
  - **Network Failure**
  - **System Failure**
  - **System Crash**
- **Tables Remain Locked for Reads and Writes until In-Doubt state is resolved**
- **Resolving In-Doubt Transactions**
  - **Automatically done by the RECO Process**
  - **Manually using COMMIT FORCE**
  - **Manually using ROLLBACK FORCE**

# RECO Process In-Doubt Resolution

- If a failure occurs during any of the phases
  - Data integrity may be compromised
- RECO resolves failures when systems restored:
  - If `PREPARE` sent by **Global Coordinator** but crash occurs before **Commit Point Site** commits then `ROLLBACK` is done when connectivity is restored
  - If `COMMIT` sent by **Global Coordinator** and crash occurs after **Commit Point Site** receives it from **Local** or **Global Coordinator** then `COMMIT` is done when connectivity is restored
  - If `ROLLBACK` sent by **Global Coordinator** and crash occurs before **Commit Point Site** receives request from the **Local** or **Global Coordinator** then `ROLLBACK` is done when connectivity is restored
  - If `FORGET` is sent and a crash occurs then it is done after recovery
- RECO then removes entries from the dictionary

# Manual In-Doubt Resolution

- **Manual In-Doubt resolution should only be done if:**
  - The in-doubt transaction has locks on critical data or undo segments
  - The cause of the machine, network, or software failure cannot be resolved quickly
  - If one of the participating databases is lost forever
- **Requires Knowledge of the Data and Current State at each remaining database**
- **Uses Views:**
  - `DBA_2PC_PENDING` based on `pending_trans$`
  - `DBA_2PC_NEIGHBORS` based on `ps1$` & `pss1$`
  - Also check `v$global_transaction`
- **May require manual removal from Data Dictionary**
- **Contact Oracle Support if you are uncertain**

## DBA\_2PC\_PENDING Data Dictionary View

- **local\_tran\_id** Local tran ID (format x.y.z)
- **global\_tran\_id** Global tran ID format global\_name.hhhhhhhh.local\_txn\_id
- **state** Collecting, Prepared, Committed, Forced Commit, Forced Abort
- **mixed** YES means Mixed Outcome on different nodes
- **advice** C (commit), R (rollback), or null for no advice
- **tran\_comment** Contains comments from application **COMMIT** command
- **fail\_time** Timestamp of when the row was inserted
- **force\_time** Timestamp of manual force decision or else null
- **retry\_time** Timestamp of when **RECO** last tried to recover the transaction
- **os\_user** Operating system account for the user
- **os\_terminal** Operating system terminal ID for the user
- **host** Name of the host machine
- **db\_user** Oracle username at the commit point site
- **Commit#** Global commit number for committed transactions

## DBA\_2PC\_PENDING STATES

- The STATE column of DBA\_2PC\_PENDING may contain:
  - **Collecting:** Applies to coordinators. The node is collecting information from adjacent database servers before deciding whether it can prepare.
  - **Prepared:** The node has prepared and may or may not have acknowledged this to its local coordinator but no **COMMIT** has been received. The node continues holding any local resource locks necessary for the transaction to commit
  - **Committed:** The node has committed the transaction, but the transaction is still pending at one or more sites
  - **Forced Commit:** A pending transaction was manually force committed at the local node by the DBA
  - **Forced Abort:** A pending transaction was manually force Rolled Back at the local node by the DBA

## DBA\_2PC\_NEIGHBORS Data Dictionary View

- **local\_tran\_id** Local transaction ID (format x.y.z)
- **in\_out** IN for incoming transaction, OUT for outgoing transactions
- **database**
  - Incoming transactions - the name of the client database
  - Outgoing transactions - the name of the database link
- **dbuser\_owner |**
  - Incoming transactions - the name of the local user
  - Outgoing transactions - the owner of the database link
- **interface**
  - C - request for commit
  - N - prepared or read-only
- **dbid** Remote database ID
- **sess#** Local session number
- **branch** Local transaction branch ID



# V\$GLOBAL\_TRANSACTION View

- **FORMATID** Format identifier of the global transaction
- **GLOBALID** Global transaction identifier of the global transaction
- **BRANCHID** Branch qualifier of the global transaction
- **BRANCHES** Total number of branches in the global transaction
- **REFCOUNT** Number of siblings for the global transaction is same as branches
- **PREPARECOUNT** Number of branches of the global transaction that have prepared
- **STATE** State of the branch of the global transaction
- **FLAGS** The numerical representation of the state
- **COUPLING** are branches are FREE, LOOSELY COUPLED, or TIGHTLY COUPLED
- **CON\_ID** container identifier for multitenant architecture

# Manual In-Doubt Resolution Procedure

```
SQL> COMMIT;
```

```
ORA-02054 transaction 1.44.99 in-doubt
```

- Note the transaction ID number from the error
- Query the local database `DBA_2PC_PENDING` view

```
SQL> SELECT LOCAL_TRAN_ID, GLOBAL_TRAN_ID,  
           STATE, COMMIT#  
FROM SYS.DBA_2PC_PENDING  
WHERE LOCAL_TRAN_ID = '1.44.99';
```

# Manual In-Doubt Resolution Procedure

- If the **STATE = COMMIT** then local Database committed
  - Examine the **global\_tran\_id** and **commit#**
  - Compare with the same on other nodes when accessible
  - If RECO resolved no matches will exist on those nodes
  - If matching **PREPARED** rows found they may be committed
- If the **STATE = PREPARED** then local Database not committed
  - Examine the **global\_tran\_id** and **commit#**
  - Examine **DBA\_2PC\_NEIGHBORS** for other Databases
  - Compare with the same on other nodes when accessible
  - If no other nodes are found prepared it is safe to **COMMIT**
  - If other nodes prepared they may all be committed or rolled back
- Use the **Global SCN** to Force commit

## Manual In-Doubt Resolution Advice

- **Developers may provide advice to DBAs**
- **This appears in `DBA_2PC_PENDING.ADVISE`**
- **May contain “R”, “C” or nulls**
- **Application May set this with:**

```
SQL> ALTER SESSION ADVISE [COMMIT | ROLLBACK | NOTHING];  
  
or  
  
DBMS_TRANSACTION.ADVISE_[COMMIT | ROLLBACK | NOTHING];
```

- **Developer must know the implications of forced commit or rollback.**

## Force COMMIT

- To force COMMIT a transaction use the command:

```
SQL> COMMIT COMMENT 'XXX' [WORK] FORCE
      'tranid', 'integer';
      or
DBMS_TRANSACTION.COMMIT_FORCE
      ('tranid', 'integer')
```

- 'tranid' is the local transaction ID
- 'integer' is the SCN number
- 'XXX' is any comment and is optional
- Requires the FORCE [ANY] TRANSACTION Privilege

## Force ROLLBACK

- To force ROLLBACK a transaction use the command:

```
SQL> ROLLBACK [WORK] FORCE 'tranid' ;  
      or  
DBMS_TRANSACTION.ROLLBACK_FORCE ( 'tranid' )
```

- '*tranid*' is the local transaction ID
  - '*XXX*' is any comment and is optional
- Requires the FORCE [ANY] TRANSACTION Privilege

# Data Dictionary Cleanup - Purging Mixed

- **Required after Mixed Result**
  - “Force” has been done resulting in a Mixed Result
  - `DBA_2PC_PENDING.MIXED` will be ‘YES’
  - RECO unable to clean up
  - Cleaned up with `DBMS_TRANSACTION.PURGE_MIXED`

```
EXEC DBMS_TRANSACTION.PURGE_MIXED ('tranid')
```

- ‘*tranid*’ is the local transaction ID

# Data Dictionary Cleanup - Purging Lost DB

- **Required after Lost or Recreated Database**
  - **DBA\_2PC\_PENDING.MIXED will be 'YES'**
  - **RECO unable to clean up**
  - **New Database will have new DBID**
  - **Attempt to recover – ORA02062**
  - **Cleaned up with**
    - **DBMS\_TRANSACTION.PURGE\_LOST\_DB\_ENTRY**

```
EXEC DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY ('tranid')
```

- **'*tranid*' is the local transaction ID**



# Summary of Dictionary Cleanup Options

<b>Pending State Column</b> -----	<b>State of Global Transaction</b> -----	<b>State of Local Transaction</b> -----	<b>Normal DBA Action</b> -----	<b>Alternative DBA Action</b> -----
collecting	rolled back	rolled back	none	<code>purge_lost_db_entry</code>
committed	committed	committed	none	<code>purge_lost_db_entry</code>
prepared	unknown	prepared	none	force commit or rollback
forced commit	unknown	committed	none	<code>purge_lost_db_entry</code>
forced rollback	unknown	rolled back	none	<code>purge_lost_db_entry</code>
forced commit	mixed	committed	Manually remove inconsistencies then use <code>purge_mixed</code>	-----
forced rollback	mixed	rolled back	Manually remove inconsistencies then use <code>purge_mixed</code>	-----

# Practicing Manual Resolution With Crash Tests

- Use the 2PC Crash Test Facility:
  - First Disable RECO

```
SQL> ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

```
SQL> COMMIT COMMENT 'ORA-2PC-CRASH-TEST-<num> '
```

- 1 - Crash commit point site after collect
- 2 - Crash non-commit point site after collect
- 3 - Crash non-commit point site before prepare
- 4 - Crash non-commit point site after prepare
- 5 - Crash commit point site before commit
- 6 - Crash commit point site after commit
- 7 - Crash non-commit point site before commit
- 8 - Crash non-commit point site after commit
- 9 - Crash commit point site before forget
- 10 - Crash non-commit point site before forget

# Recovery Strategy for Distributed Databases

- **Complete Recovery**
  - All changes committed up to the moment of failure are recovered
- **Tablespace or Table Point in Time Recovery**
  - Committed changes up to a point *before* the failure are recovered
  - Time, LogSeq or SCN
- **Database Point in Time Recovery**
  - Committed changes up to a point *before* the failure are recovered
  - Time, Cancel or SCN

# Complete Database Recovery

- **Complete recovery Implications:**
- **Distributed transaction metadata restored to state prior to failure**
- **RECO then automatically resolves any in-doubt transactions**
- **It is autonomously performed so there is no impact on dependencies between databases**
- **May be done by the Recovering Instance in RAC**

# Tablespace or Table Point In Time Recovery

- **May be required for corruption or logical errors**
- **Data in one or more tablespaces or tables are restored and recovered to a point in the past**
- **Tables involved in 2PC may have lost updates**
- **Data in other Databases must be restored and recovered to same point**
  - **Using TSPITR in Remote Database**
  - **Using TPITR in Remote Databases**
  - **Using DBPITR in Remote Database**
- **Use the SCN or Time of recovery from the first TSPITR to do the remaining recovery operations**

# Database Point In Time Recovery - DBPITR

- **Also known as Incomplete Recovery**
- **Data in one Database are restored and recovered to a point in the past**
- **Tables involved in 2PC may have lost updates**
- **Data in one or more additional Databases must be restored and recovered to same point with DBPITR**
  - **Using DBPITR in Remote Database**
  - **Obtain SCN from recovered Database**
    - **Alert log**
    - **V\$DATABASE.RESETLOGS\_CHANGE#**
  - **On all other Databases do:**

```
SQL> RECOVER DATABASE UNTIL CHANGE <SCN>;
```

# Summary

- **Connectivity Overview**
- **Distributed Database Characteristics**
- **Remote and Distributed Queries**
- **Remote and Distributed Transactions**
- **Distributed Transaction Branches and RAC**
- **Two-Phase In-Doubt Transactions**
- **The RECO Process In-Doubt Resolution**
- **Manual In-Doubt Resolution**
- **Data Dictionary Entries and Cleanup**
- **2 Phase Commit Crash Recovery**
- **Distributed Systems Recovery**