2024

Oracle 19c Database Administration

DAY 3

Day 3 Topics

Managing Storage Space

Space Management Features

- ✓ ASSM: Automatic management of segment space using bitmaps.
- ✓ LMT: Extent management at the tablespace level using bitmaps.
- ✓ ASM: Simplified and high-performance storage management.
- ✓ Extent Management: Uniform size or autoallocate extents for efficient space usage.
- ✓ Block Space Management: Efficient management of free space within blocks.
- ✓ Online Segment Shrink: Reclaims wasted space in segments.
- ✓ Oracle Data Pump: High-performance data movement utility.
- ✓ Compression: Reduces storage requirements and improves performance.

Automatic Segment Space Management (ASSM)

ASSM simplifies and improves the management of free space within segments (e.g., tables and indexes). It uses bitmaps to track free space, which provides better performance and concurrency than traditional freelists.

Benefits:
- ✓ Reduced contention for free space.
- ✓ Improved performance for DML operations.
- ✓ Simplified space management.

Usage:
ASSM is enabled by default for tablespaces created with the SEGMENT SPACE MANAGEMENT AUTO clause.

Example:
```
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

Locally Managed Tablespaces (LMT)

LMTs manage extents locally within the tablespace using bitmaps, rather than in the data dictionary. This approach improves performance and scalability.

Benefits:
- ✓ Reduced contention on the data dictionary.
- ✓ Faster extent allocation and deallocation.
- ✓ Improved scalability for large databases.

Usage:
LMT is the default and recommended mode for tablespaces.

Example:

```
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL;
```

Automatic Storage Management (ASM)

ASM is a volume manager and a file system for Oracle database files that provides high performance and reliability. It simplifies database storage management by automating file distribution and management.

Benefits:

- ✓ Simplified storage management.
- ✓ Improved performance and reliability.
- ✓ Automatic load balancing and redundancy.

Usage:
ASM is configured during database installation and managed through Oracle Grid Infrastructure.

Example: ASM Disk Group Creation:

```
CREATE DISKGROUP data_diskgroup NORMAL REDUNDANCY
DISK '/dev/sda1', '/dev/sdb1';
```

Extent Management

Extents are units of space allocation in a tablespace. Oracle manages extents automatically within LMTs.

Features:

- ✓ Uniform Extent Allocation: All extents are of a uniform size.
- ✓ Autoallocate: Extents are allocated in sizes optimized by Oracle.

Usage:
Uniform Extent Allocation Example:

```
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M;
```

Autoallocate Example:

```
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

Block Space Management

Block space management deals with the management of free space within database blocks. Oracle uses bitmaps for managing free space within blocks when ASSM is enabled.

Features:

- ✓ Reduced block contention.
- ✓ Efficient space utilization.

Online Segment Shrink

Online segment shrink reclaims wasted space in segments (tables, indexes) and returns it to the tablespace, without causing significant downtime.

Benefits:

- ✓ Reclaims wasted space.
- ✓ Improves space utilization.
- ✓ Minimal impact on database operations.

Usage:

```
ALTER TABLE employees SHRINK SPACE;
ALTER INDEX emp_idx SHRINK SPACE;
```

Oracle Data Pump

Oracle Data Pump is a high-performance data movement utility used for fast data transfer between databases.

Benefits:

- ✓ Efficient data export and import.
- ✓ Supports compression, encryption, and parallelism.
- ✓ Can be used for space management tasks such as reorganizing tablespaces.

Usage:

Export Example:

```
expdp system/password DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp FULL=YES
```

Import Example:

```
impdp system/password DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp FULL=YES
```

Compression

Oracle offers several types of data compression to reduce storage requirements and improve performance.

Types:
- ✓ Advanced Row Compression: Compresses table data.
- ✓ Hybrid Columnar Compression (HCC): Provides high compression ratios for data warehousing.

Usage:
Advanced Row Compression Example:

```
CREATE TABLE employees COMPRESS FOR OLTP AS SELECT * FROM employees;
```

HCC Example:

```
CREATE TABLE employees COMPRESS FOR QUERY HIGH AS SELECT * FROM employees;
```

Example Workflow

```
--Create a Tablespace with ASSM and LMT:
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;

--Add a Data File to the Tablespace:
ALTER TABLESPACE example_tbs
ADD DATAFILE '/u01/app/oracle/oradata/example_tbs02.dbf' SIZE 50M;

--Enable Autoextend for a Data File:
ALTER DATABASE DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf'
AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED;

--Shrink a Table to Reclaim Space:
ALTER TABLE employees SHRINK SPACE;

--Export Data Using Data Pump:
expdp system/password DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp FULL=YES
```

Row Chaining and Migration

Row Chaining
Row chaining occurs when a row is too large to fit into a single database block. In this case, Oracle stores the row across multiple blocks. This situation is typical with rows that contain large objects (LOBs) such as CLOBs and BLOBs.

Causes:
- The row size exceeds the block size (e.g., the sum of column sizes in a row is larger than the block size).
- Impact:
- Increases I/O operations because retrieving a chained row requires reading multiple blocks.
- Can lead to performance degradation, especially for full table scans and other I/O-intensive operations.

Detection:

- You can detect row chaining by querying the DBA_TABLES or USER_TABLES view, which includes the CHAIN_CNT column indicating the number of chained rows in a table.

Example:

```
SELECT table_name, chain_cnt
FROM dba_tables
WHERE chain_cnt > 0;
```

Row Migration

Row migration occurs when an update to a row causes it to no longer fit in its original block, and Oracle moves the entire row to a new block. A pointer remains in the original block, pointing to the new block.

Causes:

- Rows that initially fit into a block but grow too large due to updates (e.g., adding data to variable-length columns).
- Impact:
- Increases I/O operations because retrieving a migrated row requires reading the original block and the block containing the migrated row.
- Can lead to performance issues, especially for index range scans and other read-intensive operations.

Detection:

- You can detect row migration by analyzing the ANALYZE TABLE ... LIST CHAINED ROWS command, which generates a report of chained and migrated rows.

Example:

Create a table to store the report:

```
CREATE TABLE chained_rows (
  owner_name   VARCHAR2(30),
  table_name   VARCHAR2(30),
  cluster_name  VARCHAR2(30),
  partition_name VARCHAR2(30),
  subpartition_name VARCHAR2(30),
  head_rowid   ROWID,
  analyze_timestamp DATE
);
```

Analyze a table and store the result in the chained_rows table:

```
ANALYZE TABLE employees LIST CHAINED ROWS INTO chained_rows;
```

Query the chained_rows table:

```
SELECT * FROM chained_rows;
```

Addressing Row Chaining and Migration

Row Chaining

Increase Block Size:

```
--Create tablespaces with larger block sizes to accommodate larger rows.
CREATE TABLESPACE large_block_tbs
DATAFILE '/u01/app/oracle/oradata/large_block_tbs01.dbf' SIZE 100M
BLOCKSIZE 16K;
```

Use LOB Storage Options:

```
--Store large objects (LOBs) in separate tablespaces or use LOB storage options to optimize storage.
CREATE TABLE large_table (
   id NUMBER,
   large_data CLOB
) LOB (large_data) STORE AS SECUREFILE (
   TABLESPACE large_lob_tbs
   ENABLE STORAGE IN ROW
);
```

Row Migration

Increase PCTFREE:

```
--Adjust the PCTFREE parameter to leave more free space in each block for future updates.
ALTER TABLE employees PCTFREE 20;
```

Reorganize Tables:

```
--Use the ALTER TABLE ... MOVE command to reorganize the table and eliminate migrated rows.
ALTER TABLE employees MOVE;
```

Rebuild Indexes:

```
--After reorganizing a table, rebuild indexes to ensure they point to the correct rows.
ALTER INDEX emp_idx REBUILD;
```

Example Workflow

```
--Detect Row Chaining and Migration:
SELECT table_name, chain_cnt
FROM dba_tables
WHERE chain_cnt > 0;

--Analyze a Table for Chained and Migrated Rows:
CREATE TABLE chained_rows (
   owner_name    VARCHAR2(30),
   table_name    VARCHAR2(30),
   cluster_name  VARCHAR2(30),
   partition_name VARCHAR2(30),
   subpartition_name VARCHAR2(30),
   head_rowid    ROWID,
   analyze_timestamp DATE
);
```

```
ANALYZE TABLE employees LIST CHAINED ROWS INTO chained_rows;

--Increase PCTFREE to Reduce Future Row Migration:
ALTER TABLE employees PCTFREE 20;

--Reorganize a Table to Eliminate Row Migration:
ALTER TABLE employees MOVE;
ALTER INDEX emp_idx REBUILD;
```

Managing Undo Data

Key Concepts

- Undo Data: Information stored in undo segments that allows the database to rollback transactions and maintain read consistency.
- Undo Tablespace: A special tablespace that stores undo segments.
- Automatic Undo Management (AUM): Oracle's method of managing undo data automatically.

Configuring Undo Management

1. Creating an Undo Tablespace
When creating a database, you typically create an undo tablespace. You can also create an additional undo tablespace if needed.

```
CREATE UNDO TABLESPACE undo_tbs
DATAFILE '/u01/app/oracle/oradata/undo_tbs01.dbf' SIZE 500M
AUTOEXTEND ON NEXT 50M MAXSIZE UNLIMITED;
```

2. Setting the Undo Tablespace
To use a specific undo tablespace, set the UNDO_TABLESPACE parameter.

```
ALTER SYSTEM SET UNDO_TABLESPACE = undo_tbs;
```

Monitoring Undo Space Usage

1. Viewing Undo Tablespace Information
You can query the DBA_TABLESPACES and DBA_DATA_FILES views to get information about the undo tablespace.

```
SELECT tablespace_name, status
FROM dba_tablespaces
WHERE contents = 'UNDO';

SELECT file_name, bytes, autoextensible
FROM dba_data_files
WHERE tablespace_name = 'UNDO_TBS';
```

2. Monitoring Undo Space Usage
Oracle provides several views to monitor undo space usage, such as V$UNDOSTAT, DBA_UNDO_EXTENTS, and DBA_HIST_UNDOSTAT.

```
SELECT begin_time, end_time, undoblks, maxquerylen, ssolderrcnt
FROM v$undostat
ORDER BY begin_time DESC;

SELECT tablespace_name, extents, blocks
FROM dba_undo_extents
ORDER BY tablespace_name;
```

Configuring Automatic Undo Retention

The UNDO_RETENTION parameter specifies the time in seconds that Oracle attempts to retain undo data to support consistent reads and flashback features.

```
ALTER SYSTEM SET UNDO_RETENTION = 900; -- Retain undo data for 15 minutes
```

Tuning Undo Tablespace

1. Sizing the Undo Tablespace
To determine the appropriate size for the undo tablespace, consider the undo retention period and the transaction workload. Oracle's DBMS_UNDO_ADV package can help with this.

```
DECLARE
 min_undo_retention NUMBER;
 undo_tablespace_size NUMBER;
BEGIN
 DBMS_UNDO_ADV.advisor(UNDO_RETENTION=>1800, -- Desired undo retention period in seconds
             RETENTION_SIZE=>undo_tablespace_size,
             MIN_RETENTION=>min_undo_retention);
 DBMS_OUTPUT.put_line('Recommended Undo Tablespace Size: ' || undo_tablespace_size || ' bytes');
 DBMS_OUTPUT.put_line('Minimum Undo Retention: ' || min_undo_retention || ' seconds');
END;
/
```

2. Extending the Undo Tablespace
If the undo tablespace needs more space, you can add data files or extend existing ones.

Example: Adding a Data File

```
ALTER TABLESPACE undo_tbs
ADD DATAFILE '/u01/app/oracle/oradata/undo_tbs02.dbf' SIZE 200M;
```

Example: Extending an Existing Data File

```
ALTER DATABASE DATAFILE '/u01/app/oracle/oradata/undo_tbs01.dbf' RESIZE 1G;
```

Best Practices for Managing Undo Data

- ✓ Use Automatic Undo Management (AUM): This simplifies undo management by automatically managing undo segments and retention.

- ✓ Monitor Undo Space Usage: Regularly monitor undo space to avoid running out of space and ensure adequate undo retention.
- ✓ Set Appropriate Undo Retention: Adjust the UNDO_RETENTION parameter based on the needs of your workload and flashback requirements.
- ✓ Size Undo Tablespace Appropriately: Ensure the undo tablespace is sized to handle peak workloads and retention requirements.
- ✓ Use Multiple Undo Tablespaces (Optional): Consider using multiple undo tablespaces for different workloads or to isolate undo data.

Example Workflow

```
--Create and Configure an Undo Tablespace:
CREATE UNDO TABLESPACE undo_tbs
DATAFILE '/u01/app/oracle/oradata/undo_tbs01.dbf' SIZE 500M
AUTOEXTEND ON NEXT 50M MAXSIZE UNLIMITED;

ALTER SYSTEM SET UNDO_TABLESPACE = undo_tbs;

--Set Undo Retention:
ALTER SYSTEM SET UNDO_RETENTION = 1800; -- Retain undo data for 30 minutes

--Monitor Undo Space Usage:
SELECT begin_time, end_time, undoblks, maxquerylen, ssolderrcnt
FROM v$undostat
ORDER BY begin_time DESC;

SELECT tablespace_name, extents, blocks
FROM dba_undo_extents
ORDER BY tablespace_name;

--Extend the Undo Tablespace:
ALTER TABLESPACE undo_tbs
ADD DATAFILE '/u01/app/oracle/oradata/undo_tbs02.dbf' SIZE 200M;

ALTER DATABASE DATAFILE '/u01/app/oracle/oradata/undo_tbs01.dbf' RESIZE 1G;

--Tune Undo Tablespace Size:
DECLARE
 min_undo_retention NUMBER;
 undo_tablespace_size NUMBER;
BEGIN
 DBMS_UNDO_ADV.advisor(UNDO_RETENTION=>1800, -- Desired undo retention period in seconds
            RETENTION_SIZE=>undo_tablespace_size,
            MIN_RETENTION=>min_undo_retention);
 DBMS_OUTPUT.put_line('Recommended Undo Tablespace Size: ' || undo_tablespace_size || ' bytes');
 DBMS_OUTPUT.put_line('Minimum Undo Retention: ' || min_undo_retention || ' seconds');
END;
/
```

Moving Data

Oracle Data Pump

Oracle Data Pump is a high-performance data movement utility used for fast data export and import. It provides several features for managing large data transfers.

Key Components

- ✓ Data Pump Export (expdp): Exports data from the database to a dump file.
- ✓ Data Pump Import (impdp): Imports data from a dump file into the database.

Example: Full Database Export

```
expdp system/password@db FULL=YES DIRECTORY=dpump_dir DUMPFILE=full_db.dmp LOGFILE=full_db.log
```

Example: Schema Export

```
expdp system/password@db SCHEMAS=schema_name DIRECTORY=dpump_dir DUMPFILE=schema.dmp
LOGFILE=schema.log
```

Example: Table Export

```
expdp system/password@db TABLES=table_name DIRECTORY=dpump_dir DUMPFILE=table.dmp
LOGFILE=table.log
```

Example: Using Parallelism

```
expdp system/password@db SCHEMAS=schema_name DIRECTORY=dpump_dir DUMPFILE=schema%U.dmp
LOGFILE=schema.log PARALLEL=4
```

Example: Full Database Import

```
impdp system/password@db FULL=YES DIRECTORY=dpump_dir DUMPFILE=full_db.dmp
LOGFILE=full_db_imp.log
```

Example: Schema Import

```
impdp system/password@db SCHEMAS=schema_name DIRECTORY=dpump_dir DUMPFILE=schema.dmp
LOGFILE=schema_imp.log
```

Example: Table Import

```
impdp system/password@db TABLES=table_name DIRECTORY=dpump_dir DUMPFILE=table.dmp
LOGFILE=table_imp.log
```

Example: Using Parallelism

```
impdp system/password@db SCHEMAS=schema_name DIRECTORY=dpump_dir DUMPFILE=schema%U.dmp
LOGFILE=schema_imp.log PARALLEL=4
```

Example Workflow: Using Data Pump

```
--Create Directory Object:
CREATE OR REPLACE DIRECTORY dpump_dir AS '/path/to/dpump_dir';
GRANT READ, WRITE ON DIRECTORY dpump_dir TO system;

--Export Schema:
expdp system/password@db SCHEMAS=hr DIRECTORY=dpump_dir DUMPFILE=hr.dmp LOGFILE=hr_exp.log

--Import Schema:
impdp system/password@db SCHEMAS=hr DIRECTORY=dpump_dir DUMPFILE=hr.dmp LOGFILE=hr_imp.log
```

SQL Loader

SQL*Loader is a utility for loading data from external files into Oracle tables. It is particularly useful for bulk loading data.

Key Components

- ✓ Control File: Defines how data is to be loaded.
- ✓ Data File: Contains the data to be loaded.
- ✓ Log File: Logs the loading process.
- ✓ Bad File: Records any rows that SQL*Loader cannot load.
- ✓ Discard File: Records any rows that do not meet selection criteria.

Preparing the Control File
The control file contains the SQL*Loader instructions. Here’s a sample control file:

```
-- Example: load.ctl
LOAD DATA
INFILE 'data.csv'
INTO TABLE my_table
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(
 column1,
 column2,
 column3
)
```

Loading Data Using SQL*Loader
Example Command:

```
sqlldr username/password@db control=load.ctl log=load.log bad=load.bad discard=load.dsc
```

Example Workflow: Using SQL*Loader
Prepare Data File (data.csv):

```
1,John Doe,5000
2,Jane Smith,6000
3,Bob Johnson,7000
```

Prepare Control File (load.ctl):

```
LOAD DATA
INFILE 'data.csv'
INTO TABLE employees
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(
 employee_id,
 employee_name,
 salary
)
```

Load Data:

```
sqlldr system/password@db control=load.ctl log=load.log bad=load.bad discard=load.dsc
```

Backup and Recovery Concepts

Types of Failures

Types of Failures

1) User Errors
2) Statement Failures
3) Process Failures
4) Instance Failures
5) Media Failures
6) Network Failures

1. User Errors
User errors occur when a user performs an incorrect or unintended action, such as deleting or updating the wrong data. These errors do not result from system or hardware issues but from human mistakes.

Examples:

- Accidental deletion of critical data.
- Incorrect updates to records.
- Dropping a table or index by mistake.

Recovery Solutions:

- Flashback Technology: Use Flashback Query, Flashback Table, Flashback Transaction Query, and Flashback Database to revert the database to a previous state.
- Point-in-Time Recovery (PITR): Restore the database to a specific point in time before the error occurred.

2. Statement Failures
Statement failures occur when an SQL statement fails to execute successfully due to syntax errors, constraint violations, or other issues.

Examples:

- SQL syntax errors.
- Violations of integrity constraints (e.g., primary key or foreign key constraints).

- Runtime errors such as division by zero.

Recovery Solutions:

- Error Correction: Correct the SQL statement and re-execute it.
- Logging and Debugging: Review error logs and application logs to identify and correct the issues.

3. Process Failures
Process failures occur when a user process or background process terminates unexpectedly. This can be due to issues like hardware failures, software bugs, or resource exhaustion.

Examples:

- A user process gets terminated due to a session kill or network disconnection.
- A background process (e.g., DBWR, LGWR) crashes due to resource issues.

Recovery Solutions:

- Automatic Process Recovery: Oracle automatically detects and recovers from process failures by restarting the failed process.
- Monitoring Tools: Use Oracle Enterprise Manager (OEM) or other monitoring tools to detect and investigate process failures.

4. Instance Failures
Instance failures occur when the Oracle instance (i.e., the set of background processes and memory structures) crashes. This can be due to hardware failures, power outages, or software issues.

Examples:

- Power failure causing the server to shut down.
- Operating system crashes.
- Oracle internal errors causing instance termination.

Recovery Solutions:

- Automatic Instance Recovery: Oracle automatically performs instance recovery upon the next startup, using redo logs to apply committed changes and roll back uncommitted transactions.
- Manual Recovery: In rare cases, manual intervention may be required to complete the recovery process.

5. Media Failures
Media failures occur when a disk or storage device containing the database files fails. This can result in data corruption or loss.

Examples:

- Disk crash or hardware failure.
- Corruption of data files, control files, or redo logs.
- Logical corruption due to file system issues.

Recovery Solutions:

- Backup and Restore: Use RMAN (Recovery Manager) to restore the affected files from backups and apply redo logs to recover transactions.
- Data Guard: Use Oracle Data Guard to maintain standby databases that can be activated in case of primary database failures.
- Block Media Recovery: Use RMAN to recover specific corrupted blocks without restoring the entire file.

6. Network Failures
Network failures occur when there are issues with the network infrastructure that prevent communication between the database and its clients or between different database instances in a distributed environment.

Examples:

- Network connectivity loss or instability.
- Network hardware failures (e.g., routers, switches).
- Network configuration issues.

Recovery Solutions:

- Network Redundancy: Implement redundant network paths to ensure continuous connectivity.
- Network Monitoring: Use network monitoring tools to detect and resolve network issues promptly.
- Automatic Client Failover: Configure Transparent Application Failover (TAF) to redirect client connections to a surviving instance in a Real Application Clusters (RAC) environment.

Instance Recovery

Instance recovery is necessary to:

- Ensure the database is brought back to a consistent state after a failure.
- Apply any committed changes that were not yet written to the data files (redo logs).
- Roll back any uncommitted transactions to maintain data integrity.

How Instance Recovery Works
Instance recovery is automatically performed by Oracle when the database is restarted after a failure. The key steps involved in instance recovery are:

- Rolling Forward: Applying the redo log records to the data files to redo changes made by committed transactions.
- Rolling Back: Undoing changes made by uncommitted transactions to ensure data consistency.

Monitoring Instance Recovery
You can monitor the progress and details of instance recovery using various views and logs:

Alert Log:
The alert log file contains detailed information about the instance recovery process, including the time taken and the number of blocks processed.

```
tail -f alert_<SID>.log
```

V$INSTANCE_RECOVERY View:
This view provides information about the settings and progress of instance recovery.

```
SELECT * FROM v$instance_recovery;
V$LOG and V$LOGFILE Views:
```

These views provide information about the redo logs used during the recovery process.

```
SELECT * FROM v$log;
SELECT * FROM v$logfile;
```

Configuring Instance Recovery Parameters

You can configure instance recovery behavior by setting the following parameters:

```
FAST_START_MTTR_TARGET:
```

This parameter specifies the desired mean time to recover (MTTR) in seconds. Oracle uses this value to optimize the checkpointing frequency and minimize recovery time.

```
ALTER SYSTEM SET FAST_START_MTTR_TARGET = 300;
```

LOG_CHECKPOINT_INTERVAL:

Specifies the maximum number of redo log blocks that can be written before a checkpoint occurs.

```
ALTER SYSTEM SET LOG_CHECKPOINT_INTERVAL = 10000;
```

LOG_CHECKPOINT_TIMEOUT:

Specifies the maximum time in seconds between checkpoints.

```
ALTER SYSTEM SET LOG_CHECKPOINT_TIMEOUT = 300;
```

Example Workflow

```
--Configure Instance Recovery Parameters:
ALTER SYSTEM SET FAST_START_MTTR_TARGET = 300;
ALTER SYSTEM SET LOG_CHECKPOINT_INTERVAL = 10000;
ALTER SYSTEM SET LOG_CHECKPOINT_TIMEOUT = 300;

--Monitor Instance Recovery:
SELECT * FROM v$instance_recovery;

--Check the Alert Log for Recovery Details:
tail -f alert_<SID>.log
```

The Checkpoint (CKPT) Process

The Checkpoint (CKPT) process is responsible for signaling the Database Writer (DBWn) to write all dirty buffers (buffers that contain data that has been modified but not yet written to disk) to data files. This ensures that the database can be recovered to a consistent state after an instance failure.

Key Functions:

- ✓ Updates the control file with the checkpoint information.
- ✓ Signals the DBWn process to write dirty buffers to data files.
- ✓ Reduces the amount of recovery work needed by periodically flushing data to disk.

Configuration:

Checkpointing behavior can be influenced by parameters like FAST_START_MTTR_TARGET, LOG_CHECKPOINT_INTERVAL, and LOG_CHECKPOINT_TIMEOUT.

Example:

```
ALTER SYSTEM SET FAST_START_MTTR_TARGET = 300; -- Target mean time to recover (MTTR) in seconds
```

Redo Log Files and the Log Writer (LGWR)

Redo Log Files:
Redo log files store all changes made to the database as redo entries. These files are essential for recovering the database to a consistent state after a failure.

Log Writer (LGWR):
The LGWR process writes redo entries from the redo log buffer in the SGA (System Global Area) to the redo log files on disk. LGWR ensures that committed transactions are safely stored in redo log files.

Key Functions:

- ✓ Writes redo entries to the online redo log files.
- ✓ Ensures that redo entries are written to disk before the user process is notified of a committed transaction.
- ✓ Writes occur during commit operations, when the redo log buffer is one-third full, or before DBWn writes dirty buffers to disk.

Example: Monitoring Redo Log Files:

```
SELECT * FROM v$log;
SELECT * FROM v$logfile;
```

Automatic Instance Recovery or Crash Recovery

Automatic instance recovery, or crash recovery, is performed by Oracle when an instance fails unexpectedly. This process is initiated automatically when the database is restarted.

Steps Involved:

- ✓ Rolling Forward: Apply all redo log entries generated after the last checkpoint to ensure all committed transactions are applied.
- ✓ Rolling Back: Undo any uncommitted transactions using undo segments to ensure data consistency.

Monitoring:
Check the alert log file for details on instance recovery.

```
tail -f alert_<SID>.log
```

Using the MTTR Advisor

The Mean Time to Recover (MTTR) Advisor helps DBAs determine the optimal settings for the FAST_START_MTTR_TARGET parameter, which specifies the target time for instance recovery.

Usage:
The MTTR Advisor provides recommendations based on the current workload and system performance.

Example: Enabling MTTR Advisor:

```
ALTER SYSTEM SET FAST_START_MTTR_TARGET = 300;
```

Example: Using the MTTR Advisor:

```
SELECT target_mttr, estimated_mttr
FROM v$instance_recovery;
```

Complete Recovery Process

Complete recovery involves restoring all data files from a backup and applying all redo log files to bring the database to the most current state.

Steps Involved:

- ✓ Restore Backup: Restore all data files from the most recent backup.
- ✓ Apply Redo Logs: Apply all archived and online redo logs.

Example:

```
rman target /
RMAN> SHUTDOWN IMMEDIATE;
RMAN> STARTUP MOUNT;
RMAN> RESTORE CONTROLFILE FROM AUTOBACKUP;
-- or specify a backup piece
-- RMAN> RESTORE CONTROLFILE FROM '/path/to/backup_piece';
RMAN> ALTER DATABASE MOUNT;
RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

The Point-in-Time Recovery Process

PITR restores the database to a specific point in time before an undesired event occurred, such as a user error.

Steps Involved:

- ✓ Restore Backup: Restore data files from a backup taken before the point in time.
- ✓ Apply Redo Logs: Apply redo logs up to the desired point in time.

Example: Tablespace Point-in-Time Recovery

```
--Restore and Recover a Specific Tablespace:
RMAN> RESTORE TABLESPACE users;
RMAN> RECOVER TABLESPACE users;

--Open the Database:
RMAN> ALTER DATABASE OPEN;
```

Example: Point-in-Time Recovery

```
--Set the Time for Recovery:
RMAN> SET UNTIL TIME "TO_DATE('2023-05-15 12:00:00', 'YYYY-MM-DD HH24:MI:SS')";

--Restore and Recover the Database:
RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;

--Open the Database with RESETLOGS:
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

Flashback

Flashback technology allows you to quickly revert the database to a previous state without restoring from backups. This includes Flashback Database, Flashback Table, Flashback Query, Flashback Drop, and Flashback Transaction Query.

Flashback Database:
Allows you to rewind the entire database to a previous point in time.

```
-- Enable Flashback Database
ALTER DATABASE FLASHBACK ON;

-- Perform Flashback Database
FLASHBACK DATABASE TO TIMESTAMP (SYSDATE - 1/24); -- Rewind to one hour ago
```

Flashback Table:
Allows you to revert one or more tables to a previous point in time.

```
FLASHBACK TABLE employees TO TIMESTAMP (SYSDATE - 1/24); -- Rewind to one hour ago
```

Flashback Query:
Allows you to query the historical state of a table at a specific point in time.

```
SELECT * FROM employees AS OF TIMESTAMP (SYSDATE - 1/24); -- Query data as of one hour ago
```

Flashback Drop:
Allows you to undo a DROP TABLE operation.

```
FLASHBACK TABLE employees TO BEFORE DROP;
```

Flashback Transaction Query:
Allows you to view changes made by a specific transaction.

```
SELECT xid, operation, undo_sql
FROM flashback_transaction_query
WHERE xid = '0002000300000001';
```

Enable Flashback Feature

```
C:\> sqlplus / as sysdba
SQL> ALTER SYSTEM SET db_recovery_file_dest = 'C:\oracle\flash_recovery_area' SCOPE=BOTH;

System altered.

SQL> ALTER SYSTEM SET db_recovery_file_dest_size = 10G SCOPE=BOTH;

System altered.

SQL> ALTER SYSTEM SET db_flashback_retention_target = 1440 SCOPE=BOTH;

System altered.

SQL> SHUTDOWN IMMEDIATE;
Database closed.
Database dismounted.
ORACLE instance shut down.

SQL> STARTUP MOUNT;
ORACLE instance started.

Total System Global Area       2516582400 bytes
Fixed Size                     9132224 bytes
Variable Size                  654311424 bytes
Database Buffers               1845493760 bytes
Redo Buffers                   7634944 bytes
Database mounted.

SQL> ALTER DATABASE ARCHIVELOG;

Database altered.

SQL> ALTER DATABASE FLASHBACK ON;

Database altered.

SQL> ALTER DATABASE OPEN;

Database altered.

SQL> SELECT flashback_on FROM v$database;

FLASHBACK_ON
------------------
YES
```

Backup and Restore an Accidental Record Deletion

Prerequisites

- ✓ Oracle Database in ARCHIVELOG mode.
- ✓ Flashback Data Archive (optional, for specific use cases).
- ✓ Proper backups using RMAN or other methods.

Using Flashback Query
Flashback Query allows you to view the data as it was at a specific point in time.

1. Enable Flashback on the Table (if not already enabled)

```
ALTER TABLE employees ENABLE ROW MOVEMENT;
```

2. Simulate Data Deletion

```
DELETE FROM employees WHERE employee_id = 101;
COMMIT;
```

3. Use Flashback Query to Recover the Deleted Record
You can use the AS OF TIMESTAMP clause to view the table as it was at a previous time and insert the deleted record back into the table.

```
-- Check the current SCN
SELECT current_scn FROM v$database;

-- Find the approximate time of deletion
SELECT SYSTIMESTAMP FROM dual;

-- Recover the deleted record
INSERT INTO employees
SELECT * FROM employees AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '10' MINUTE)
WHERE employee_id = 101;
```

4. Verify the Restoration
Check if the record has been restored.

```
SELECT * FROM employees WHERE employee_id = 101;
```

Using RMAN Backup
If flashback features are not enabled or suitable, you can restore the table from an RMAN backup.

1. Backup the Database
Ensure you have a recent backup of the database.

```
rman target /

RMAN> BACKUP DATABASE;
```

2. Simulate Data Deletion

```
DELETE FROM employees WHERE employee_id = 101;
COMMIT;
```

3. Restore the Database to a Point in Time
Start RMAN and Connect to the Target Database

```
rman target /
```

Shutdown and Mount the Database

```
RMAN> SHUTDOWN IMMEDIATE;
RMAN> STARTUP MOUNT;
```

Set the Time for Point-in-Time Recovery

```
RMAN> SET UNTIL TIME "TO_DATE('2023-06-01 10:00:00', 'YYYY-MM-DD HH24:MI:SS')";
```

Restore and Recover the Database

```
RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;
```

Open the Database with RESETLOGS

```
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

Example Session for RMAN Point-in-Time Recovery

```
rman target /

RMAN> SHUTDOWN IMMEDIATE;
RMAN> STARTUP MOUNT;

-- Assuming the deletion happened at '2023-06-01 10:00:00'
RMAN> SET UNTIL TIME "TO_DATE('2023-06-01 09:59:00', 'YYYY-MM-DD HH24:MI:SS')";

RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

Verification

After performing the recovery, verify that the deleted record is restored.

```
SELECT * FROM employees WHERE employee_id = 101;
```