Construction

COSC 434 Database Administration

Okanagan College
Computer Science Department
Kelowna, BC

November 22 2025

Keona Gagnier, Cody Jorgenson, Alex Anthony,
Emilio Iturbide Gonzalez, Cade Dempsey

Table of Contents

Team Member	Roles
Keona Gagnier	Security, Backup and Recovery
Cody Jorgenson	Scrum master, Product Owner
Alex Anthony	Backup and Recovery, Performance Manager
Emilio Iturbide Gonzalez	Performance Manager, User and Developer Support
Cade Dempsey	User and Developer Support, Security

Physical Database Structure

Tablespaces

The principal schema of the database is maintained under the stock_user account, which possesses ownership of both the stocks table and its associated indexes. All users have been granted SELECT access on the stocks table.

The stocks table resides within the stocks_data tablespace, whereas its indexes are allocated to the stocks_index tablespace. The stock_user account has been granted unlimited quota on both the stocks_data and stocks_index tablespaces.

Refer to **Appendix B: Tablespace Creation** to see the SQL used to create the tablespaces.

Tables and Indexes

DDL for stocks table and indexes on it:

```
CREATE TABLE stock_user.stocks (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    symbol VARCHAR2(10) NOT NULL,
    trade_date TIMESTAMP NOT NULL,
    open NUMBER(10,4),
    low NUMBER(10,4),
    high NUMBER(10,4),
    close NUMBER(10,4),
    volume NUMBER(19),
    CONSTRAINT stocks_symbol_date_uniq UNIQUE (symbol, trade_date)
)
TABLESPACE stocks_data;
```

```
CREATE INDEX stock_user.STOCKS_SYMBOL_IDX ON stock_user.stocks
(symbol) TABLESPACE stocks_index;

CREATE INDEX stock_user.STOCKS_DATE_IDX ON stock_user.stocks
(trade_date) TABLESPACE stocks_index;
```

Partitioning

The stocks table is partitioned quarterly by date ranges. The stocks with a trade date before the year 2025 will belong to the `STOCKS_BEFORE_2025` partition while the 2025 stocks are assigned to a quarterly partition, e.g. `STOCKS_Q1_2025` (Jan 1, Mar 31), `STOCKS_Q2_2025` (Apr 1, Jun 30), etc. Future rows added with a trade date past 2025 will be added to partitions that are generated with an interval of 3 months.

Refer to **Appendix E: Partitioning Tables** to see the SQL used to partition the table.

Storage Strategy

Data files are stored in the following directory: /u01/app/oracle/product/19.0.0/dbhome_1/dbs/

We separated the stocks table and its indexes into distinct tablespaces to optimize performance and improve manageability. By allocating the table to the stocks_data tablespace and the indexes to the stocks_index tablespace, we can reduce I/O contention, tailor storage growth and quotas independently, and streamline maintenance tasks such as index rebuilding or tablespace backups. This separation also enables more efficient use of storage resources and provides greater flexibility for recovery operations, ensuring that table and index management can be handled with minimal impact on overall database performance.

Physical Parameters

All physical parameters are configured by the DBA team to optimize performance while maintaining simplicity and stability.

Block Size

- Parameter: DB_BLOCK_SIZE
- Value: 8 KB (default)
- Rationale: The 8 KB block size is the Oracle-recommended default for mixed OLTP and reporting workloads.

Memory Management – Automatic Memory Management (AMM)

- Parameters: MEMORY_TARGET, MEMORY_MAX_TARGET

- Value: Set at CDB level by the DBA team (AMM fully enabled)
- SGA_TARGET = 0, PGA_AGGREGATE_TARGET = 0

- Verification of AMM (run as any DBA user):

```
SELECT name, value
FROM   v$parameter
WHERE  name IN
('memory_target','memory_max_target','sga_target','pga_aggregate_t
arget');

Output from 2025-11-20:
NAME                         VALUE
---------------------------- ---------------
memory_target                3968M
memory_max_target            3968M
sga_target                   0
pga_aggregate_target         0
```

- Rationale: When SGA_TARGET and PGA_AGGREGATE_TARGET are both zero, Oracle takes full control of memory allocation. The instance automatically grows or shrinks the buffer cache, shared pool, Java pool, large pool, and PGA based on current demand.

Individual Memory Components

- Parameters: DB_CACHE_SIZE, SHARED_POOL_SIZE, LARGE_POOL_SIZE, JAVA_POOL_SIZE
- Value: All set to 0 (not manually configured)
- Rationale: All components are dynamically managed under AMM.

Log Buffer

- Parameter: LOG_BUFFER
- Value: System-determined default (~15–30 MB in our VM)
- Verification:

```
SELECT value/1024/1024 AS log_buffer_mb FROM v$sga WHERE name =
'Log Buffer';
```

- Rationale: The default size is automatically calculated based on SGA size and is more than sufficient for our append-only workload.

Operational Procedures

Backup Strategy

Overview

Important Directories

Purpose	Directory
Flash Recovery Area	$ORACLE_BASE/oradata/ORCL/FRA
Monthly and Yearly Archival Backups	/u02/rman/cdb1/stable_archives
RMAN Backup Scripts	$ORACLE_BASE/admin/orcl/scripts/rman
Logs generated by cron jobs for backup scripts	$ORACLE_BASE/admin/orcl/logs/rman
Backup and Recovery Test Scripts	$ORACLE_BASE/admin/orcl/scripts/test_scripts/backup_recovery_tests
Logs generated by backup scripts themselves	$ORACLE_BASE/admin/cdb1/logs/rman
Holds copy of the control file on a separate disk	/mnt/backups

Type of Backup

RMAN is used for full, cold backups.

Since our workload is append only (stock data should not be UPDATED or DELETED once it has been inserted into the database), full backups simplify the backup process.

Control file autobackup is on.

This ensures the control file and SPFILE are always backed up, making recovery possible even if they are lost.

The default device type is disk.

Disk is the fastest and simplest backup target in our local VM environment.

The backup type for disks is backup sets.

Backup sets reduce storage usage and also let RMAN skip unused blocks.

Flash Recovery Area is configured, based on recommendations in the Configuring for Recoverability [4] and Backup and Recovery Concepts [3] powerpoints shown in class.

DB_RECOVERY_FILE_DEST = /u01/app/oracle/oradata/ORCL/FRA
DB_RECOVERY_FILE_DEST_SIZE = 100G

```
-- IN RMAN
CONFIGURE CONTROLFILE AUTOBACKUP ON;
```

```
CONFIGURE DEFAULT DEVICE TYPE TO DISK;
CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO BACKUPSET;
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;
```

Frequency and Schedule

1. **Daily Full Backups** (stored in the flash recovery area at $ORACLE_BASE/oradata/ORCL/FRA)
    - Captures newly added stock market data for the day.
    - Maintained by a 7 day recovery window.
    - Full backups were chosen over incremental backups for simplicity and easier management.
    - Daily backups are not compressed to speed up recovery.

2. **Stable Monthly Redundant Full Backup** (stored at /u02/rman/cdb1/stable_archives/)
    - Its purpose is to preserve a stable fallback copy in case corruption or human error goes unnoticed for several months. Since regular database users will only have SELECT access on the stocks table, it is unlikely to experience data corruption through user error.
    - Has the KEEP clause specified to 90 days (3 months) from the current day.
    - Backup is compressed to save storage.

3. **Stable Yearly Redundant Full Backup** (stored at /u02/rman/cdb1/stable_archives/yearly)
    - Retained for 2 years.
    - Has the KEEP clause specify 2 years from the current day.
    - Backup is compressed to save storage.

RMAN Backup Scripts Summary

The backup strategy is executed via two scheduled shell scripts scheduled via cron. For the full, runnable code of these scripts, refer to **Appendix A: RMAN Backup Scripts.**

Backup scripts are automated through cron scheduling.

Script Purpose	**Filename**	**Execution Frequency**
Daily Full Backup	rman_daily_cold_fullbu_cdb1.sh	Daily
Monthly Full Backup, Compressed	rman_monthly_stable_cold_fullbu_cdb1.sh	Monthly
Yearly Full Backup, Compressed	rman_yearly_stable_cold_fullbu_cdb1.sh	Yearly

Retention Policy

- Number of backups kept: Enough for PITR within the past 7 days, around 3 monthly backups, and 2 yearly backups.

- Retention Method: Oracle RMAN Recovery Window of 7 days, and a specified KEEP clause for the monthly and yearly backups, where monthly backups are retained for 3 months and yearly backups are retained for 2 years.
- Backup Type: Cold, full database backups. Compressed backups for the monthly and yearly backups in an effort to save space.
- Archive logs: Retained only as needed to support the daily backups and are deleted after the backup completes. Archive logs are also included in the monthly and yearly backups.

This policy avoids long-term accumulation of old backups and minimizes storage usage while still providing meaningful recovery options.

As our database stands with ~29 million records, one full backup of the database is 8.8GB, so for the 7 day recovery window the FRA will store roughly 61.6GB.

Rationale for Strategy

- Historical stock data does not change once loaded, only new records arrive. This gets rid of the need for complex incremental backup changes.
- Keeping only 7 days worth of PITR dramatically reduces space consumption compared to multi-week or multi-month retention windows for daily backups.
- Simpler management for the DBA team.
- Straightforward to implement within the limited storage resources available in our environment.
- High recovery speed because taking a daily full backup means any recovery event within the last week will require restoring a very recent full backup and applying a minimal amount of archive logs, keeping downtime low.

Recovery Test Protocol

Manually Checking Correctness of Backup Scripts

- Prior to scheduling RMAN backup scripts to automate the backup plan, run the RMAN scripts manually to ensure they properly create backups as instructed.

Scripts to Test Existing Backups

These scripts are manually run by the DBA to test the backup and recovery process. These should be run once a month to ensure the recovery process is intact.

Location of scripts: /u01/app/oracle/admin/orcl/scripts/test_scripts/backup_recovery_tests

Script	Purpose
simulate_pitr_restore.sh	Simulate large data loss or corruption, and restore from backups kept by the 7 day recovery window.
simulate_restore_from_monthly.sh	Simulate large data loss or corruption, and restore from the most recent monthly backup.

simulate_restore_from_yearly.sh	Simulate large data loss or corruption, and restore from the most recent yearly backup.

Securing the Database Structure: Dedicated Recovery Metadata

In our virtual machine environment, we provisioned a small, separate disk (/dev/xvdb1) specifically to hold critical disaster recovery metadata.

The physical volume we allocated has a tight capacity limit (approximately 10MB). Therefore, our strategy is to store only a single, compressed backup of the current control file in this location, ensuring that we can quickly recover the database structure should the main disk suffer damage.

Since our core application data is sourced directly from Financial Modelling Prep and is easily repopulated, and there is a lack of storage on the secondary disk, we will not put a full backup on that disk. Our primary recovery concern is protecting the essential structural files (the control file) which defines the database layout and history.

We first mounted the filesystem and established the access directory at /mnt/backups. Gemini [2] was used to explain the process of partitioning a disk and adding a file system, as well as provide commands to accomplish this. The successful mount is confirmed below:

```
[root@cosc-ora ~]# df -h /mnt/backups
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvdb1      7.6M   14K  7.0M   1% /mnt/backups
```

We then executed an RMAN command to create a compressed backup set containing only the current control file, which is small enough to fit the volume. The script used for this targeted operation is available for reference in **Appendix I: RMAN Script for Structural Backup**.

The final result below confirms the successful creation of the control file backup piece in the designated recovery location:

```
[oracle@cosc-ora ~]$ ls /mnt/backups
CONTROL_FILE_TEST_2m49mgfj_1_1
```

Monitoring and Tuning Procedures

Performance Monitoring and Tuning

Performance Monitoring and Tuning is fully automated through a suite of Oracle Scheduler jobs owned by the MONITORING user (Alex Anthony & Emilio Iturbide Gonzalez). These jobs address key performance areas in the shared teaching environment:

- Resource management – automatic termination of idle or long-running personal sessions

    - MON_KILL_HOGS – Create idle (>8 h) and long-running (>2 h) test sessions; the job terminates them cleanly
- Index maintenance – daily rebuild of any unusable indexes
    - MON_INVALID_FIX – Manually mark an index unusable; the job rebuilds it (STATUS changes from UNUSABLE to VALID in USER_INDEXES)
- Optimizer statistics – weekly refresh of schema statistics
    - MON_GATHER_STATS – Execute manually; LAST_ANALYZED timestamps are updated in USER_TABLES
- Space management – daily alert if any tablespace exceeds 85% usage
    - MON_TS_ALERT – Simulate high tablespace usage; job fails with ORA-20001 containing correct alert text in USER_SCHEDULER_JOB_RUN_DETAILS
- Query analysis – nightly snapshot of the top-20 most expensive SQL statements
    - MON_TOP_SQL – Execute heavy queries; job captures top-20 statements in monitoring.top_sql_log table

All jobs are pure PL/SQL (type STORED_PROCEDURE). Jobs were created based on materials presented in the Lesson 13: Performance Management [7] PowerPoint slides from the course with support using Gemini AI [2].

Memory Management
Oracle Database 19c uses Automatic Memory Management (AMM) by default. The parameters MEMORY_TARGET and MEMORY_MAX_TARGET are configured by the DBA team, enabling the database to dynamically allocate memory between the System Global Area (SGA) and Program Global Area (PGA) based on workload demands. No manual tuning of individual components is required.

- Verification (run as MONITORING user on 22-Nov-2025):
    - Confirm AMM is active

```
SELECT name, value FROM v$parameter
WHERE name IN
('memory_target','memory_max_target','sga_target','pga_aggre
gate_target');
```

A daily dashboard script (**morning_check.sql**) is executed each morning to provide a one-screen health summary.

For full details on the Scheduler jobs, supporting table, and daily dashboard script, refer to Appendix H: Performance Monitoring and Tuning Scripts.

For the decision workflow when alerts or failures occur, refer to Appendix H.4: Performance Monitoring Response Guide.

Capacity Planning
Based on ~29 million records (initial load), empirical growth is ~10,605,168 records per year (~1.86 GB/year combined data and indexes). Provision 15-33 GB over 5 years, with autoextend enabled and alerts at 75%/90% tablespace usage.

Risk Mitigation
Performance risks are mitigated through automated backups (refer to Appendix A: RMAN Backup Scripts), least-privilege user access, comprehensive logging, and Flashback Recovery for logical errors.

Security Policies

User and Role Management

List of DBA Users/Roles

Person	Account Username	Privileges/Roles
Keona Gagnier	kgag	DBA, SYSDBA
Alex Anthony	aant	DBA, SYSDBA, ADVISOR
Cody Jorgenson	cjor	DBA, SYSOPER
Emilio Iturbide Gonzalez	eitu	DBA, SYSOPER, ADVISOR
Cade Dempsey	cdem	DBA, SYSOPER

For the script used to create DBA users and grant their privileges and roles, refer to **Appendix C: User Creation**.

Privileges Granted to Users

User	Purpose	Privileges/Quotas
APP_READONLY	Provides read-only access for dashboards or APIs. Useful for visualization tools like	CREATE SESSION, SELECT, No quota (read only)

	PowerBI.	
ML_ANALYST	Performs queries, aggregation, and statistical exploration	CREATE SESSION, SELECT ANY TABLE, CREATE MATERIALIZED VIEW, CREATE SYNONYM,
ML_DEVELOPER	Pulls stock and derived data into Python notebooks or model pipelines. Represents developers doing model training and testing.	CREATE SESSION, SELECT on relevant views/tables, CREATE TABLE and CREATE VIEW for experimentation
STOCK_USER	Store the STOCKS table in their tablespace	CREATE SESSION, CREATE TABLE, CREATE SEQUENCE

For the script used to create non-system users and grant their privileges and roles, refer to **Appendix C: User Creation**.

Access Control

Grant/Revoke Statements (Include the actual GRANT statements used to allow DBA users SELECT access on the stocks table.)
Principle of least privilege enforcement.

Auditing

To enable standard auditing and ensure audit records are stored in the data dictionary, the initialization parameter audit_trail must be set, followed by a database restart.

ALTER SYSTEM SET audit_trail = DB, extended SCOPE=SPFILE;

Configuration Setting: audit_trail is set to DB, EXTENDED. This ensures audit records are stored in the data dictionary, and the full sql statements are recorded, accessible via the DBA_AUDIT_TRAIL view in the SYS schema.

Auditing is enabled for the use of critical system privileges by DBAs and developers, configured BY ACCESS to ensure administrative actions are recorded for accountability

Key Privileges Audited:

- AUDIT CREATE ANY TABLE BY ACCESS;
- AUDIT DROP ANY TABLE BY ACCESS;
- AUDIT ALTER SYSTEM BY ACCESS;
- AUDIT CREATE USER BY ACCESS;
- AUDIT GRANT ANY PRIVILEGE BY ACCESS;

Since the stocks table is the core of the database, DML activity (Data Manipulation Language) performed on it is audited:

- Activity Audited: SELECT, INSERT, UPDATE, and DELETE operations on the stocks table are audited BY ACCESS

- Command Required: AUDIT SELECT, INSERT, UPDATE, DELETE ON stock_user.stocks BY ACCESS; This captures every access or modification of the stock data, especially by non-system users

Logging and Session Auditing:

Session-level auditing is configured to track who connects to the database and when, including unsuccessful login attempts

- Command Required:
    - AUDIT SESSION;

Review and Maintenance

- Audit data will be reviewed periodically using queries on dba_audit_trail.
- To manage space, older audit records (e.g., those older than 30 days) are archived and purged regularly using a DBMS SCHEDULER JOB.

Refer to **Appendix F: Auditing Scripts** to see the SQL used to archive and purge audit logs.

Data Loading and Migration

Source of Data

Data was originally sourced from Financial Modelling Prep API, however for importing data into this database, a folder of CSV files from the Algorithmic Trading GitHub was used [1]. The folder holds 503 stocks, each with its own file, containing FMP data from July 1 2023 to October 24 2025.

Loading Method

Originally, the data was actually loaded into the DRI database, using a bash script that would go through each file and load the data into the table, which you can see in **Appendix D: Loading Method Scripts, section D.1**.

After this, we decided to use the COSC server to host our database, so we exported a CSV file of the data from the DRI database.

SQL*Loader was used to load the CSV file from the DRI database into our COSC server database. The control file used to do can be seen in **Appendix D: Loading Method Scripts, section D.2.**

The SQL*Loader operation was executed with the following command:

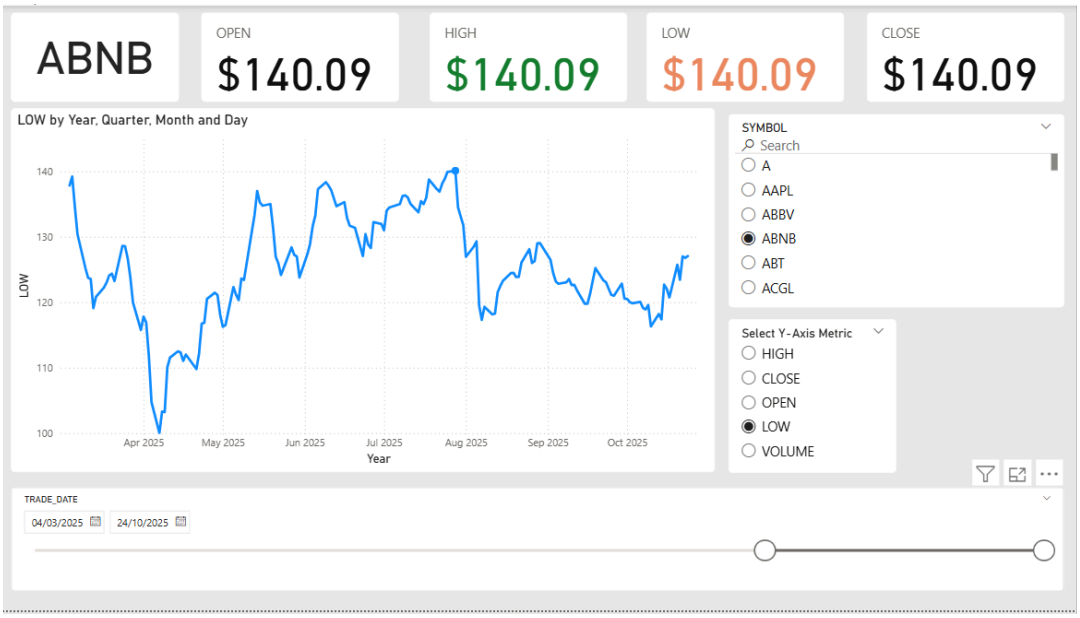
```
sqlldr stock_user/pass@ORCLPDB control=stocks.ctl log=stocks.log
```

Verification

The unique constraint for (symbol, trade_date) which exists on the stocks table was useful is ensuring no duplicate rows were inserted into the table (no more than one row exists for one stock at one time).

After the data was loaded, a SELECT COUNT(*) query was issued to see the number of rows inserted into the table. The result was 29,381,155 rows of data, which follows the estimate of approximately 10 million rows per year for 503 stocks with data collected at 5 minute intervals.

Additionally, data was inspected through visualization in BI tools:



The above dashboard allows the user to select a stock, the y-axis metric, as well as the time frame of interest to modify the line chart as needed. The values shown up top reflect the values for the chosen data point in the graph.

Testing and Evaluation

Performance Tuning

The performance tuning strategy and health-check procedures implemented in this project were designed based off the concepts and tools presented in Lesson 13: Performance Management [7] PowerPoint slides from the course and debugged using Gemini AI [2].

Health Checks and Performance Baselines
The Automatic Workload Repository (AWR) and Automatic Database Diagnostic Monitor (ADDM) are used to collect and analyze performance data. Baselines are established using DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE for comparison across environments

Statistics are reviewed at system-wide (V$SYSSTAT, V$SYSTEM_EVENT), session-specific (V$SESSTAT, V$SESSION_EVENT), and service-specific (V$SERVICE_STATS, V$SERVICE_EVENT) levels.

Alert logs and trace files are reviewed weekly for ORA- errors, I/O failures, inconsistencies, or corruptions. Retention policy: Alert logs retained for 90 days.

Bi-weekly Drill

1. Load one full quarter of data using the script in Appendix D.1 (or SQL*Loader control file in D.2)
2. Execute a sort-heavy analytical query: SELECT symbol, AVG(close) AS avg_close, SUM(volume) AS total_volume FROM stocks PARTITION (STOCKS_Q1_2025) -- change partition as needed GROUP BY symbol ORDER BY SUM(volume) DESC;
3. Verify performance:
    a. Buffer cache hit ratio > 95% (query V$SYSSTAT)
    b. No temporary segment disk spill (query V$PGASTAT and V$TEMPSEG_USAGE)
    c. PGA memory remains within acceptable limits (query V$SESSION)

Verify performance queries:

- Check buffer cache hit ratio (> 95% = good)

```
SELECT ROUND(100 * (1 - (pr.value / (dbg.value + cg.value))), 2)
|| '%' AS cache_hit_ratio
FROM v$sysstat pr,v$sysstat dbg,v$sysstat cg
WHERE pr.name  = 'physical reads'
      AND dbg.name = 'db block gets'
      AND cg.name  = 'consistent gets';
```

- Check no temp disk spill (temp_spill should be 0 or very low)

```
SELECT name, value
FROM   v$pgastat
WHERE  name LIKE '%temp%' OR name LIKE ‘%physical%direct%’;
```

- Check PGA memory usage

```
SELECT s.sid,s.username,ROUND(ss.value/1024/1024,2) AS pga_mb
FROM   v$session s
JOIN   v$sesstat ss ON s.sid = ss.sid
JOIN   v$statname sn ON ss.statistic# = sn.statistic#
WHERE  sn.name = 'session pga memory'
  AND  s.username = 'STOCK_USER'
ORDER  BY pga_mb DESC;
```

Refer to Appendix H: Performance Monitoring and Tuning Scripts for creation scripts and test output.

Backup and Recovery Strategy Testing

Testing of the backup and recovery strategy is done through scripts that simulate data loss and then use the backups on hand to perform restoration and recovery.

Test Scripts

You can view the full scripts in **Appendix G: Backup & Recovery Test Scripts**.
The scripts are stored at /u01/app/oracle/admin/orcl/scripts/test_scripts/backup_recovery_tests

simulate_pitr_restore.sh

- This script deletes 100,000 rows of data from the stocks table, and then uses the most recent backup to recover them.
- It checks recovery by comparing the count of rows before deletions, and after restoration.
- The following is example output from a successful run of this test script:

```
================================================================
Starting Data Loss Simulation and PITR at Wed Nov 19 22:31:44 EST
2025
Target PDB: ORCLPDB
================================================================
1. Initializing: Ensuring PDB ORCLPDB is open for READ WRITE.
2. Checking initial row count in ORCLPDB...
Initial Rows in STOCKS: 29381155
3. Simulating disaster: Deleting 100,000 rows from STOCKS in
ORCLPDB.
Checking post-disaster row count...
POST-DISASTER Rows in STOCKS: 29281155
Disaster confirmed: Data deleted. Proceeding with recovery.
4. Closing PDB ORCLPDB before RMAN restore.

5. Starting RMAN PITR process...
Recovery Target Time: 2025-11-19 22:31:47
RMAN> RMAN> 2> 3> 4> 5> 6> 7> 8> 9> 10> 11> 12> 13> 14> 15> 16>
17>

RMAN> RMAN PITR completed successfully.
6. Checking final row count after recovery...
RECOVERED Rows in STOCKS: 29381155
SUCCESS: Recovery confirmed. Initial count (29381155) matches
recovered count (29381155).
================================================================
Simulation finished at Wed Nov 19 22:46:38 EST 2025
================================================================
```

simulate_restore_from_monthly.sh and **simulate_restore_from_yearly.sh**

- These scripts delete 10,000 rows from the stocks table and use the most recent monthly backup or most recent yearly backup, respectively, to perform restoration and recovery.
- The scripts check recovery by comparing the count of rows before deletions, and after restoration.
- The following is example output from a successful run of the monthly test script:

```
==========================================================================
Starting Monthly Backup Restore Simulation at Thu Nov 20 00:06:45
EST 2025
Target PDB: ORCLPDB | Rows to Delete: 10000
Using Backup Tag: STABLE_MONTHLY_BU
==========================================================================
1. Initializing: Ensuring PDB ORCLPDB is open for READ WRITE.
2. Checking initial row count in ORCLPDB...
Initial Rows in STOCKS: 29381155
3. Simulating disaster: Deleting 10000 rows from STOCKS.
Checking post-disaster row count...
POST-DISASTER Rows in STOCKS: 29371155
Disaster confirmed: Data deleted. Proceeding with full CDB
recovery.
4. Shutting down the CDB for full restore operation.
5. Starting RMAN FULL CDB RESTORE using tag STABLE_MONTHLY_BU...
Recovery Target Time: 2025-11-20 00:06:47
RMAN> RMAN> 2> 3> 4> 5> 6> 7> 8> 9> 10> 11> 12> 13> 14> 15> 16>
17> 18> 19> 20> 21> 22> 23> 24> 25> 26>

RMAN> RMAN CDB Restore/Recovery completed successfully.
6. Checking final row count after recovery...
RECOVERED Rows in STOCKS: 29381155
SUCCESS: Recovery confirmed. Initial count (29381155) matches
recovered count (29381155).
==========================================================================
Simulation finished at Thu Nov 20 00:18:46 EST 2025
==========================================================================
```

Scripts and Configuration Appendix

This appendix contains the various scripts used in construction of the database and testing. You may also view this code in the GitHub Repository for this project.

Appendix A: RMAN Backup Scripts

This section contains the complete shell scripts used to implement the database backup strategy.

These scripts were created with the help of Gemini AI [2] for debugging, and were built using information from the Lesson 5: Using RMAN to Create Backups [5] powerpoint slides shown in class.

A.1 Primary Daily Full Backup Script (rman_daily_cold_fullbu_cdb1.sh)

This script handles the routine daily backup and runs DELETE OBSOLETE to clean up the previous backups as needed. This script is located at $ORACLE_BASE/admin/orcl/scripts/rman.

```
#!/bin/bash
#-------------------------------------------------------------------------
# Filename: rman_daily_cold_fullbu_cdb1.sh
# Purpose: Executes a daily COLD (offline) full database backup for cdb1
#          container database (CDB). This makes a consistent backup.
# Execution: Scheduled via cron daily.
# Requires downtime during the backup process.
#-------------------------------------------------------------------------

# --- Environment Setup ---
export ORACLE_SID=cdb1
export ORAENV_ASK=NO
export ORACLE_BASE=/u01/app/oracle

# Determine ORACLE_HOME by reading the system /etc/oratab file.
export ORACLE_HOME=$(cat /etc/oratab | grep -E "^$ORACLE_SID:" | cut -d
':' -f 2)

# Fallback to a common path if oratab reading fails
if [ -z "$ORACLE_HOME" ]; then
    export ORACLE_HOME="/u01/app/oracle/product/19.0.0/dbhome_1"
fi
export PATH=$ORACLE_HOME/bin:$PATH
export NLS_DATE_FORMAT='DD-MON-YYYY HH24:MI:SS'
```

```
# --- Logging Configuration ---
LOG_DIR="$ORACLE_BASE/admin/cdb1/logs/rman"
LOG_FILE="$LOG_DIR/daily_coldbu_$(date +\%Y\%m\%d).log"

# Create log directory if it doesn't exist
mkdir -p $LOG_DIR

echo
"======================================================================"
| tee -a $LOG_FILE
echo "Starting Daily Cold (Offline) Database Backup at $(date)" | tee -a
$LOG_FILE
echo "Estimated Downtime: Database will be down for SHUTDOWN, BACKUP,
and STARTUP." | tee -a $LOG_FILE
echo
"======================================================================"
| tee -a $LOG_FILE

# --- 1. SHUTDOWN IMMEDIATE ---
echo "1. Attempting to SHUTDOWN IMMEDIATE..." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
SHUTDOWN IMMEDIATE;
EXIT;
EOF

# Check if shutdown was successful (status must be 0)
if [ $? -ne 0 ]; then
    echo "ERROR: Database SHUTDOWN failed. Check logs for details.
Exiting backup." | tee -a $LOG_FILE
    exit 1
fi
echo "Database successfully shut down." | tee -a $LOG_FILE

# --- 2. STARTUP MOUNT ---
echo "2. Attempting to STARTUP MOUNT..." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
STARTUP MOUNT;
EXIT;
EOF

if [ $? -ne 0 ]; then
```

```
    echo "ERROR: Database STARTUP MOUNT failed. Check logs for details.
Exiting backup." | tee -a $LOG_FILE
    exit 1
fi
echo "Database successfully mounted." | tee -a $LOG_FILE

# --- 3. RMAN Execution (COLD Backup and Cleanup) ---
echo "3. Starting RMAN COLD Backup and Cleanup..." | tee -a $LOG_FILE

rman target / log=$LOG_FILE append << RMAN_EOF
set echo on;
RUN {
    # Take the COLD Full Backup while the database is mounted.
    # This backup is guaranteed to be consistent, so no archivelogs are
needed.
    BACKUP
      AS BACKUPSET
      DATABASE;

    # Cleanup: Delete all backups (fulls and archivelogs) that are no
longer
    # required to satisfy the configured 7-day RECOVERY WINDOW.
    DELETE NOPROMPT OBSOLETE;
}
EXIT;
RMAN_EOF

RMAN_STATUS=$?

# --- 4. OPEN DATABASE (Restore service) ---
echo "4. Attempting to ALTER DATABASE OPEN..." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
ALTER DATABASE OPEN;
EXIT;
EOF

# Check the status of the RMAN command (RMAN_STATUS is set above)
if [ $RMAN_STATUS -eq 0 ]; then
    echo "Daily COLD Backup and Cleanup completed successfully at
$(date)" | tee -a $LOG_FILE
else
```

```
    echo "ERROR: Daily COLD Backup failed (RMAN Status: $RMAN_STATUS).
Database was restarted. Check logs for details." | tee -a $LOG_FILE
fi

echo
"========================================================================="
| tee -a $LOG_FILE
# --- Completion ---
exit $RMAN_STATUS
```

A.2 Stable Monthly Redundant Backup Script (rman_monthly_stable_cold_fullbu_cdb1.sh)

This script creates the long-term archival copy. It dynamically calculates an expiry date one month into the future using the KEEP UNTIL TIME clause. The subsequent DELETE OBSOLETE command cleans up the previous month's backup, which is now expired.

Since the KEEP clause is not allowed in the FRA when the recovery window retention policy is being used, these backups are stored at /u02/rman/cdb1/stable_archives.

```
#!/bin/bash
#-------------------------------------------------------------------------
# Filename: rman_monthly_stable_cold_fullbu_cdb1.sh
# Purpose: Executes a stable monthly COLD (offline) full database backup
#          for the cdb1 container database (CDB), protected by the KEEP
clause.
# Execution: Scheduled via cron monthly (e.g., on the 1st day).
# Requires downtime during the backup process.
#-------------------------------------------------------------------------

# --- Environment Setup (CRON-Ready Fixes Retained) ---
export ORACLE_SID=cdb1
export ORAENV_ASK=NO
export ORACLE_BASE=/u01/app/oracle

# Determine ORACLE_HOME by reading the system /etc/oratab file.
export ORACLE_HOME=$(cat /etc/oratab | grep -E "^$ORACLE_SID:" | cut -d
':' -f 2)

# Fallback to a common path if oratab reading fails or is inaccurate
if [ -z "$ORACLE_HOME" ]; then
```

```
    export ORACLE_HOME="/u01/app/oracle/product/19.0.0/dbhome_1"
fi
export PATH=$ORACLE_HOME/bin:$PATH
export NLS_DATE_FORMAT='DD-MON-YYYY HH24:MI:SS'


# --- Configuration for Monthly Backup ---
# Calculate the date 35 days from now for the KEEP UNTIL TIME clause.
KEEP_DATE=$(date -d "+35 days" +\%Y-\%m-\%d)
BACKUP_TAG='STABLE_MONTHLY_BU'
STABLE_BACKUP_DEST='/u02/rman/cdb1/stable_archives'

# --- Logging Configuration ---
LOG_DIR="$ORACLE_BASE/admin/cdb1/logs/rman"
LOG_FILE="$LOG_DIR/monthly_stable_coldbu_$(date +\%Y\%m\%d).log"

# Create log directory if it doesn't exist
mkdir -p $LOG_DIR

echo
"============================================================================="
| tee -a $LOG_FILE
echo "Starting Stable Monthly COLD (Offline) Database Backup at $(date)"
| tee -a $LOG_FILE
echo "New backup will be kept until: ${KEEP_DATE}" | tee -a $LOG_FILE
echo
"============================================================================="
| tee -a $LOG_FILE

# --- 1. SHUTDOWN IMMEDIATE (Stop all activity to ensure consistency)
---
echo "1. Attempting to SHUTDOWN IMMEDIATE..." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
SHUTDOWN IMMEDIATE;
EXIT;
EOF

if [ $? -ne 0 ]; then
    echo "ERROR: Database SHUTDOWN failed. Check logs for details.
Exiting backup." | tee -a $LOG_FILE
    exit 1
fi
```

```
echo "Database successfully shut down." | tee -a $LOG_FILE

# --- 2. STARTUP MOUNT (Database must be mounted for RMAN) ---
echo "2. Attempting to STARTUP MOUNT..." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
STARTUP MOUNT;
EXIT;
EOF

if [ $? -ne 0 ]; then
    echo "ERROR: Database STARTUP MOUNT failed. Check logs for details.
Exiting backup." | tee -a $LOG_FILE
    exit 1
fi
echo "Database successfully mounted." | tee -a $LOG_FILE

# --- 3. RMAN Execution (COLD Backup and Cleanup) ---
echo "3. Starting RMAN COLD Backup and Cleanup..." | tee -a $LOG_FILE

rman target / log=$LOG_FILE append << RMAN_EOF
set echo on;
RUN {

    # Create the new stable monthly COLD backup.
    # KEEP UNTIL TIME: Protects this backup from the daily DELETE
OBSOLETE.
    # Since it is a COLD backup (consistent), LOGS is unnecessary.
    BACKUP
      AS BACKUPSET
      DATABASE
      FORMAT '${STABLE_BACKUP_DEST}/monthly_stable_bu_%U'
      TAG '${BACKUP_TAG}'
      KEEP UNTIL TIME "TO_DATE('${KEEP_DATE}','YYYY-MM-DD')";
}
EXIT;
RMAN_EOF

RMAN_STATUS=$?

# --- 4. OPEN DATABASE (Restore service) ---
echo "4. Attempting to ALTER DATABASE OPEN..." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
```

```
ALTER DATABASE OPEN;
EXIT;
EOF

# Check the status of the RMAN command (RMAN_STATUS is set above)
if [ $RMAN_STATUS -eq 0 ]; then
    echo "Stable Monthly COLD Backup finished successfully at $(date)" |
tee -a $LOG_FILE
else
    echo "ERROR: Monthly COLD Backup failed (RMAN Status: $RMAN_STATUS).
Database was restarted. Check logs for details." | tee -a $LOG_FILE
fi

echo
"============================================================================="
| tee -a $LOG_FILE
# --- Completion ---
exit $RMAN_STATUS
```

A.3 Stable Yearly Redundant Backup Script (rman_yearly_stable_cold_fullbu_cdb1.sh)

```
#!/bin/bash
#-----------------------------------------------------------------------------
# Filename: rman_yearly_stable_cold_fullbu_cdb1.sh
# Purpose: Executes a stable YEARLY COLD (offline) full compressed
database backup
#          for the cdb1 container database (CDB), protected by the KEEP
clause.
# Execution: Scheduled via cron yearly (e.g., on Jan 1st).
# Requires downtime during the backup process.
#-----------------------------------------------------------------------------

# --- Environment Setup ---
export ORACLE_SID=cdb1
export ORAENV_ASK=NO
export ORACLE_BASE=/u01/app/oracle

# Determine ORACLE_HOME by reading the system /etc/oratab file.
export ORACLE_HOME=$(cat /etc/oratab | grep -E "^$ORACLE_SID:" | cut -d
':' -f 2)
```

```
# Fallback to a common path if oratab reading fails or is inaccurate
if [ -z "$ORACLE_HOME" ]; then
    export ORACLE_HOME="/u01/app/oracle/product/19.0.0/dbhome_1"
fi
export PATH=$ORACLE_HOME/bin:$PATH
export NLS_DATE_FORMAT='DD-MON-YYYY HH24:MI:SS'


# --- Configuration for Yearly Backup ---
# Calculate the date 2 years from now for the KEEP UNTIL TIME clause.
YEARS_TO_KEEP=2
# Using 365.25 days/year for a slightly more accurate date
DAYS_TO_KEEP=$((YEARS_TO_KEEP * 365 + YEARS_TO_KEEP / 4))
KEEP_DATE=$(date -d "+$DAYS_TO_KEEP days" +\%Y-\%m-\%d)

BACKUP_TAG='STABLE_YEARLY_BU'
# New destination path for yearly archives
STABLE_BACKUP_DEST='/u02/rman/cdb1/stable_archives/yearly'

# --- Logging Configuration ---
LOG_DIR="$ORACLE_BASE/admin/cdb1/logs/rman"
LOG_FILE="$LOG_DIR/yearly_stable_coldbu_$(date +\%Y\%m\%d).log"

# Create log directory and the yearly destination if they don't exist
mkdir -p $LOG_DIR
mkdir -p $STABLE_BACKUP_DEST

echo
"============================================================================="
| tee -a $LOG_FILE
echo "Starting Stable YEARLY COLD (Offline) COMPRESSED Database Backup
at $(date)" | tee -a $LOG_FILE
echo "New backup will be kept for ${YEARS_TO_KEEP} years, until:
${KEEP_DATE}" | tee -a $LOG_FILE
echo
"============================================================================="
| tee -a $LOG_FILE

# --- 1. SHUTDOWN IMMEDIATE (Stop all activity to ensure consistency)
---
echo "1. Attempting to SHUTDOWN IMMEDIATE..." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
```

```
SHUTDOWN IMMEDIATE;
EXIT;
EOF

if [ $? -ne 0 ]; then
    echo "ERROR: Database SHUTDOWN failed. Check logs for details.
Exiting backup." | tee -a $LOG_FILE
    exit 1
fi
echo "Database successfully shut down." | tee -a $LOG_FILE

# --- 2. STARTUP MOUNT (Database must be mounted for RMAN) ---
echo "2. Attempting to STARTUP MOUNT..." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
STARTUP MOUNT;
EXIT;
EOF

if [ $? -ne 0 ]; then
    echo "ERROR: Database STARTUP MOUNT failed. Check logs for details.
Exiting backup." | tee -a $LOG_FILE
    exit 1
fi
echo "Database successfully mounted." | tee -a $LOG_FILE

# --- 3. RMAN Execution (COLD Backup and Cleanup) ---
echo "3. Starting RMAN COLD Backup and Cleanup..." | tee -a $LOG_FILE

rman target / log=$LOG_FILE append << RMAN_EOF
set echo on;
RUN {
      AS COMPRESSED BACKUPSET
      DATABASE
      PLUS ARCHIVELOG
      FORMAT '${STABLE_BACKUP_DEST}/yearly_stable_bu_%U'
      TAG '${BACKUP_TAG}'
      KEEP UNTIL TIME "TO_DATE('${KEEP_DATE}','YYYY-MM-DD')";
}
EXIT;
RMAN_EOF

RMAN_STATUS=$?
```

```
# --- 4. OPEN DATABASE (Restore service) ---
echo "4. Attempting to ALTER DATABASE OPEN..." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
ALTER DATABASE OPEN;
EXIT;
EOF

if [ $? -ne 0 ]; then
    echo "ERROR: Database ALTER DATABASE OPEN failed. Service may still
be down." | tee -a $LOG_FILE
fi

if [ $RMAN_STATUS -ne 0 ]; then
    echo "ERROR: RMAN backup failed with status $RMAN_STATUS. Check log
file: $LOG_FILE" | tee -a $LOG_FILE
    exit 1
else
    echo "SUCCESS: Stable yearly cold compressed backup completed
successfully and the database is OPEN." | tee -a $LOG_FILE
fi

exit 0
```

A.4 Backup Logs Cleanup Script (rman_log_cleanup.sh)

This script runs every day and deletes the logs generated by the cron job for the backups that are older than 7 days.

```
#!/bin/bash
# ----------------------------------------------------------------
# Script: rman_log_cleanup.sh
# Purpose: Deletes RMAN log files older than 7 days
#          to prevent excessive log accumulation. Designed for silent
cron execution.
# Frequency: Daily via cron.
# ----------------------------------------------------------------

LOG_DIR="/u01/app/oracle/admin/orcl/logs/rman"
LOG_RETENTION_DAYS=7

# On success this script produces no output.
```

```
# -type f : look for regular files (not directories)
# -name "*.log" : only target the files which end in .log
# -mtime +${LOG_RETENTION_DAYS} : Find the files modified more than 7
days ago
find ${LOG_DIR} -type f -name "*.log" -mtime +${LOG_RETENTION_DAYS}
-delete
```

A.5 Crontab Scheduling of Backups and Log Cleaning Jobs

Gemini AI [2] was used for debugging this crontab file.
The following crontab entries schedule the above scripts to run daily and monthly, respectively:

```
# Crontab Entry for the ORACLE user

# Primary Daily Backup (Runs at 11:00 PM every day)
# Minute(0) Hour(23) DayMonth(*) Month(*) DayWeek(*)
0 23 * * * /bin/bash -c
'/u01/app/oracle/admin/orcl/scripts/rman/rman_daily_cold_fullbu_cdb1.sh
>> /u01/app/oracle/admin/orcl/logs/rman/daily_backup_$(date
+\%Y\%m\%d).log 2>&1'

# Stable Monthly Redundant Backup (Runs at 12:00 AM on the 1st of every
month)
# Minute(0) Hour(0) DayMonth(1) Month(*) DayWeek(*)
# NOT CURRENTLY ACTIVE
#0 0 1 * *
/u01/app/oracle/admin/orcl/scripts/rman/rman_monthly_stable_cold_fullbu_
cdb1.sh >> /u01/app/oracle/admin/orcl/logs/rman/monthly_backup.log 2>&1

# Yearly Redundant Backup (Runs at 1:00 AM on January 1st)
# Minute(0) Hour(1) DayMonth(1) Month(1) DayWeek(*)
0 1 1 1 * /bin/bash -c
'/u01/app/oracle/admin/orcl/scripts/rman/rman_yearly_stable_cold_fullbu_
cdb1.sh >> /u01/app/oracle/admin/orcl/logs/rman/yearly_shell_$(date
+\%Y\%m\%d).log 2>&1'

# Daily Log Cleanup (Runs at 6:00 AM every day)
# Minute(0) Hour(6) DayMonth(*) Month(*) DayWeek(*)
0 6 * * * /u01/app/oracle/admin/orcl/scripts/rman/rman_log_cleanup.sh >
/dev/null 2>&1
```

Appendix B: Tablespace Creation

The following SQL was used to create the two tablespaces in our database:

```
CREATE TABLESPACE STOCKS_DATA
DATAFILE 'stocks_data01.dbf'
SIZE 4G
AUTOEXTEND ON NEXT 1G MAXSIZE 20G;

CREATE TABLESPACE STOCKS_INDEX
DATAFILE 'stocks_index01.dbf'
SIZE 5G
AUTOEXTEND ON NEXT 1G MAXSIZE 10G;
```

Appendix C: User Creation

C.1 Create DBA Users

This script was used to create the DBA users, and grant their privileges and roles.

```
-- Ensure password file is enabled (should already be EXCLUSIVE)
SHOW PARAMETER REMOTE_LOGIN_PASSWORDFILE;

-- Create users
CREATE USER C##KGAG IDENTIFIED BY "pass" CONTAINER=ALL;
CREATE USER C##AANT IDENTIFIED BY "pass" CONTAINER=ALL;
CREATE USER C##CJOR IDENTIFIED BY "pass" CONTAINER=ALL;
CREATE USER C##EITU IDENTIFIED BY "pass" CONTAINER=ALL;
CREATE USER C##CDEM IDENTIFIED BY "pass" CONTAINER=ALL;

-- Allow them to log in
GRANT CREATE SESSION TO C##KGAG CONTAINER=ALL;
GRANT CREATE SESSION TO C##AANT CONTAINER=ALL;
GRANT CREATE SESSION TO C##CJOR CONTAINER=ALL;
GRANT CREATE SESSION TO C##EITU CONTAINER=ALL;
GRANT CREATE SESSION TO C##CDEM CONTAINER=ALL;

-- Grant DBA roles
GRANT DBA TO C##KGAG CONTAINER=ALL;
GRANT DBA TO C##AANT CONTAINER=ALL;
GRANT DBA TO C##CJOR CONTAINER=ALL;
```

```
GRANT DBA TO C##EITU CONTAINER=ALL;
GRANT DBA TO C##CDEM CONTAINER=ALL;

-- Grant SYSDBA / SYSOPER privileges
-- Only possible for common users
GRANT SYSDBA TO C##KGAG CONTAINER=ALL;
GRANT SYSDBA TO C##AANT CONTAINER=ALL;

GRANT SYSOPER TO C##CJOR CONTAINER=ALL;
GRANT SYSOPER TO C##EITU CONTAINER=ALL;
GRANT SYSOPER TO C##CDEM CONTAINER=ALL;

-- Additional roles
GRANT ADVISOR TO C##AANT CONTAINER=ALL;
GRANT ADVISOR TO C##EITU CONTAINER=ALL;

GRANT ALTER SESSION TO C##KGAG CONTAINER=ALL;
GRANT ALTER SESSION TO C##AANT CONTAINER=ALL;
GRANT ALTER SESSION TO C##CJOR CONTAINER=ALL;
GRANT ALTER SESSION TO C##EITU CONTAINER=ALL;
GRANT ALTER SESSION TO C##CDEM CONTAINER=ALL;
```

C.2 Create Non-System Users

```
-- Switch into PDB
ALTER SESSION SET CONTAINER = ORCLPDB;

-- Create Users
CREATE USER APP_READONLY IDENTIFIED BY "pass";
CREATE USER ML_ANALYST IDENTIFIED BY "pass";
CREATE USER ML_DEVELOPER IDENTIFIED BY "pass";
CREATE USER STOCK_USER IDENTIFIED BY "pass";

-- Allow sessions
GRANT CREATE SESSION TO APP_READONLY;
GRANT CREATE SESSION TO ML_ANALYST;
GRANT CREATE SESSION TO ML_DEVELOPER;
GRANT CREATE SESSION TO STOCK_USER;

-- APP_READONLY ROLE: Dashboards / Power BI / APIs
-- Read-only access: only SELECT allowed
```

```
GRANT SELECT ANY TABLE TO APP_READONLY;

-- No quotas since user should not write to any tablespace
ALTER USER APP_READONLY QUOTA 0 ON USERS;

-- prevent object creation entirely
GRANT READ ANY TABLE TO APP_READONLY;

-- Give stock user unlimited quota on stocks data tablespace
ALTER USER STOCK_USER QUOTA UNLIMITED ON STOCKS_DATA;

-- ML_ANALYST ROLE: Querying, aggregation, materialized views
GRANT SELECT ANY TABLE TO ML_ANALYST;
GRANT CREATE MATERIALIZED VIEW TO ML_ANALYST;
GRANT CREATE SYNONYM TO ML_ANALYST;

-- No table creation required, so set quota to 0
ALTER USER ML_ANALYST QUOTA 0 ON USERS;

-- ML_DEVELOPER ROLE: Exploratory modeling & experimentation
GRANT SELECT ANY TABLE TO ML_DEVELOPER;

-- Allow experimentation (local scratch tables)
GRANT CREATE TABLE TO ML_DEVELOPER;
GRANT CREATE VIEW TO ML_DEVELOPER;

-- Allow them to save experimental objects in USERS tablespace
ALTER USER ML_DEVELOPER QUOTA UNLIMITED ON USERS;
```

Appendix D: Loading Method Scripts

D.1 Loading data into the DRI database

This script gathers CSV data from the Algorithmic Trading GitHub and loads it into the DRI PostgreSQL database. Gemini AI [2] was used for debugging.

```
#!/bin/bash
module load postgresql/16.0

DB_NAME="keona_db"
DB_USER="keona"
```

```
DB_HOST="cedar-pgsql-vm"
DB_PORT="5432"

DATA_DIR="/home/keona/stocks-data/AlgorithmicTrading/Data/July 1, 2023
to October 24, 2025/FMP API/503 Stocks"

for file in "$DATA_DIR"/*.csv; do
    symbol=$(basename "$file" .csv)
    echo "Loading file: $file for symbol $symbol ..."

    psql "host=$DB_HOST dbname=$DB_NAME user=$DB_USER port=$DB_PORT"
<<EOF
CREATE TEMP TABLE temp_stocks(
    date TIMESTAMP,
    open NUMERIC(10,4),
    low NUMERIC(10,4),
    high NUMERIC(10,4),
    close NUMERIC(10,4),
    volume BIGINT
);

\copy temp_stocks FROM '$file' WITH (FORMAT csv, HEADER true);

INSERT INTO stocks(symbol, date, open, low, high, close, volume)
SELECT '$symbol', date, open, low, high, close, volume FROM temp_stocks;
EOF

done
```

D.2 SQL*Loader Control File

This control file loads data from a single CSV file into the COSC server Oracle database.

```
LOAD DATA
INFILE '/u01/app/oracle/oradata/ORCL/csv_data/stocks_data_utf.csv'
INTO TABLE stock_user.stocks
INSERT
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(
  dummy_id FILLER,              -- ignore ID column bc we will recreate
  symbol,
```

```
 trade_date "TO_TIMESTAMP(:trade_date,'YYYY-MM-DD HH24:MI:SS')",
 open,
 low,
 high,
 close,
 volume
)
```

Appendix E: Partitioning Tables

E.1 Modify Table Partitions

This script alters the current stocks table to add quarterly partitions set by the trade_date column. Gemini AI [2] was used to debug this script.

```
ALTER SESSION SET CONTAINER = orclpdb;
ALTER TABLE STOCK_USER.STOCKS
      MODIFY PARTITION BY RANGE (trade_date)
      INTERVAL (NUMTOYMINTERVAL(3, 'MONTH'))
      (
            PARTITION STOCKS_BEFORE_2025 VALUES LESS THAN (TIMESTAMP
'2025-01-01 00:00:00') TABLESPACE stocks_data,
                  PARTITION STOCKS_Q1_2025 VALUES LESS THAN (TIMESTAMP
'2025-04-01 00:00:00') TABLESPACE stocks_data,
                  PARTITION STOCKS_Q2_2025 VALUES LESS THAN (TIMESTAMP
'2025-07-01 00:00:00') TABLESPACE stocks_data,
                  PARTITION STOCKS_Q3_2025 VALUES LESS THAN (TIMESTAMP
'2025-10-01 00:00:00') TABLESPACE stocks_data,
                  PARTITION STOCKS_Q4_2025 VALUES LESS THAN (TIMESTAMP
'2026-01-01 00:00:00') TABLESPACE stocks_data
      )
      ONLINE
      UPDATE INDEXES;
```

E.2 Testing Stocks Table Partitions

This script tests the stocks table partitions to ensure that the previous script was executed correctly. Gemini AI [2] was used to debug the script.

```
CONNECT / AS SYSDBA;
ALTER SESSION SET CONTAINER = orclpdb;
DECLARE
    v_plan_start VARCHAR2(255);
```

```
    v_plan_stop VARCHAR2(255);
    v_partitioned_status VARCHAR2(5);
    v_owner CONSTANT VARCHAR2(30) := 'STOCK_USER';
    v_table CONSTANT VARCHAR2(30) := 'STOCKS';
    v_test_date TIMESTAMP := TIMESTAMP '2025-05-15 00:00:00'; -- A date
clearly within Q2
BEGIN
    DBMS_OUTPUT.PUT_LINE('--- Starting Partition Verification Script for
' || v_owner || '.' || v_table || ' ---');

    -- Test 1: Verify Table Partitioning Status
    SELECT partitioned
    INTO v_partitioned_status
    FROM dba_tables
    WHERE owner = v_owner AND table_name = v_table;

    IF v_partitioned_status = 'YES' THEN
        DBMS_OUTPUT.PUT_LINE('TEST 1 SUCCESS: Table is confirmed as
partitioned.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('TEST 1 FAILURE: Table is not partitioned.
Aborting script.');
        RETURN;
    END IF;

    -- Test 2: Verify Partition Pruning via EXPLAIN PLAN

    -- Clear previous plan entry for this statement
    EXECUTE IMMEDIATE 'DELETE FROM PLAN_TABLE WHERE STATEMENT_ID =
''PART_TEST_01''';
    DBMS_OUTPUT.PUT_LINE('Test 2: Cleared previous plan entries.');

    -- Explain a query that should prune to a single partition
    EXECUTE IMMEDIATE '
        EXPLAIN PLAN SET STATEMENT_ID = ''PART_TEST_01'' FOR
        SELECT * FROM ' || v_owner || '.' || v_table || '
        WHERE trade_date = :1'
    USING v_test_date;
    DBMS_OUTPUT.PUT_LINE('Test 2: Executed EXPLAIN PLAN.');

    -- Retrieve the partition access details from the standard
PLAN_TABLE
```

```
    -- We are looking for any step that shows a specific Pstart/Pstop
    SELECT partition_start, partition_stop
    INTO v_plan_start, v_plan_stop
    FROM PLAN_TABLE
    WHERE STATEMENT_ID = 'PART_TEST_01'
      AND (
            (operation = 'TABLE ACCESS' AND options LIKE '%BY RANGE%')
OR
            (operation = 'TABLE ACCESS' AND options LIKE '%BY LOCAL
INDEX%') OR
            (operation LIKE '%INDEX%' AND options LIKE '%BY RANGE%')
          )
      AND partition_start IS NOT NULL
      AND ROWNUM = 1; -- Just take the first relevant row

    -- Analyze the plan output for evidence of single partition access
    IF v_plan_start = v_plan_stop AND v_plan_start <> 'ROWID' THEN
        DBMS_OUTPUT.PUT_LINE('TEST 2 SUCCESS: Query on specific date
shows evidence of exact partition pruning.');
        DBMS_OUTPUT.PUT_LINE('   (Pstart: ' || v_plan_start || ', Pstop:
' || v_plan_stop || ')');
    ELSE
        DBMS_OUTPUT.PUT_LINE('TEST 2 FAILURE: Query on specific date did
not prune effectively.');
        DBMS_OUTPUT.PUT_LINE('   (Pstart: ' || v_plan_start || ', Pstop:
' || v_plan_stop || ')');
    END IF;

    -- Cleanup the plan table entry for Test 2
    EXECUTE IMMEDIATE 'DELETE FROM PLAN_TABLE WHERE STATEMENT_ID =
''PART_TEST_01''';

    -- Test 3: Verify Data Distribution for a specific partition
(Optional but useful)
    DECLARE
        v_row_count NUMBER;
    BEGIN
        -- This uses the syntax to force access to a specific partition
for a row count
        EXECUTE IMMEDIATE '
            SELECT COUNT(*) FROM ' || v_owner || '.' || v_table || '
PARTITION (STOCKS_Q2_2025)'
```

```
        INTO v_row_count;
        DBMS_OUTPUT.PUT_LINE('TEST 3 RESULT: Partition STOCKS_Q2_2025
contains ' || v_row_count || ' rows.');
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('TEST 3 FAILURE: Could not access
partition STOCKS_Q2_2025 directly. Check partition names.');
    END;

    DBMS_OUTPUT.PUT_LINE('--- Verification Script Finished ---');

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: NO_DATA_FOUND in Test 2
or 3. The optimizer might have chosen an unexpected path.');
        DBMS_OUTPUT.PUT_LINE('Check alert log for errors or use the
debug script provided previously.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' ||
SQLERRM);
END;
/
```

Appendix F: Auditing Scripts

F.1 Archive and Purge Audit Logs

This Script will archive audit records and compress them in the aud$_archive table and purge them from the sys.aud$ table automatically every 30 days.

```
-- ============================================================
-- 1. Create compressed archive table for audit rows
-- ============================================================

BEGIN
    -- Create table only if not exists
    EXECUTE IMMEDIATE '
        CREATE TABLE aud$_archive
        COMPRESS FOR ARCHIVE HIGH
        AS SELECT * FROM sys.aud$ WHERE 1=0
```

```
    ';
EXCEPTION
    WHEN OTHERS THEN
        IF SQLCODE = -955 THEN
            NULL; -- table already exists
        ELSE
            RAISE;
        END IF;
END;
/

-- Create index only if not exists
BEGIN
    EXECUTE IMMEDIATE '
        CREATE INDEX aud$_archive_ts_idx
        ON aud$_archive (timestamp#)
    ';
EXCEPTION
    WHEN OTHERS THEN
        IF SQLCODE = -955 THEN
            NULL; -- index already exists
        ELSE
            RAISE;
        END IF;
END;
/

-- ============================================================
-- 2. Create monthly scheduler job (runs 1st of every month)
-- ============================================================

BEGIN
    DBMS_SCHEDULER.CREATE_JOB (
        job_name        => 'ARCHIVE_PURGE_AUDIT_JOB',
        job_type        => 'PLSQL_BLOCK',
        job_action      => q'[
            DECLARE
                l_cutoff DATE := TRUNC(SYSDATE) - 30;
            BEGIN
                -- Archive rows
                INSERT /*+ APPEND */ INTO aud$_archive
                SELECT *
```

```
                FROM sys.aud$
                WHERE timestamp# < l_cutoff;

                -- Purge rows
                DELETE FROM sys.aud$
                WHERE timestamp# < l_cutoff;

                COMMIT;
            END;
        ]',
        start_date      => SYSTIMESTAMP,
        repeat_interval =>
'FREQ=MONTHLY;BYMONTHDAY=1;BYHOUR=02;BYMINUTE=00;BYSECOND=00',
        enabled         => TRUE,
        comments        => 'Monthly archive & purge of SYS.AUD$ older
than 30 days'
    );
END;
/
```

Appendix G: Backup & Recovery Test Scripts

Gemini AI [2] was used for debugging the recovery test scripts, and Lesson 7: Using RMAN to Perform Recovery [6] was referred to throughout writing the scripts.

G.1 Simulate PITR using most recent daily backup (simulate_pitr_restore.sh)

This script deletes 100,000 rows from the stocks table, and then uses the most recent daily backup to recover and restore the table.

```
#!/bin/bash
#--------------------------------------------------------------------------
# Script: simulate_pitr_restore.sh
# Purpose: Simulates a user disaster (mass DELETE) in ORCLPDB and
restores
#          the PDB to a Point-in-Time (PITR) before the disaster
occurred.
# Execution: Run manually by the Oracle user.
#--------------------------------------------------------------------------
```

```
# --- Environment Setup ---
export ORACLE_SID=cdb1
export ORAENV_ASK=NO
export ORACLE_BASE=/u01/app/oracle

# PDB specific configuration
TARGET_PDB='ORCLPDB'
USER_TO_IMPERSONATE='STOCK_USER'
TABLE_NAME='STOCKS'

# Determine ORACLE_HOME
export ORACLE_HOME=$(cat /etc/oratab | grep -E "^$ORACLE_SID:" | cut -d
':' -f 2)
if [ -z "$ORACLE_HOME" ]; then
    export ORACLE_HOME="/u01/app/oracle/product/19.0.0/dbhome_1"
fi
export PATH=$ORACLE_HOME/bin:$PATH
export NLS_DATE_FORMAT='DD-MON-YYYY HH24:MI:SS'


# --- Logging Configuration ---
LOG_DIR="$ORACLE_BASE/admin/cdb1/logs/rman"
LOG_FILE="$LOG_DIR/pitr_simulation_$(date +\%Y\%m\%d_\%H\%M\%S).log"
mkdir -p $LOG_DIR

echo
"========================================================================="
| tee -a $LOG_FILE
echo "Starting Data Loss Simulation and PITR at $(date)" | tee -a
$LOG_FILE
echo "Target PDB: ${TARGET_PDB}" | tee -a $LOG_FILE
echo
"========================================================================="
| tee -a $LOG_FILE

# --- 1. INITIALIZATION: Ensure PDB is open for read and write  ---
echo "1. Initializing: Ensuring PDB ${TARGET_PDB} is open for READ
WRITE." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
-- Open the PDB if it's not already open. This is essential for all
queries.
ALTER PLUGGABLE DATABASE ${TARGET_PDB} OPEN;
```

```
EXIT;
EOF

# --- 2. PRE-CHECK: Get initial row count ---
echo "2. Checking initial row count in ${TARGET_PDB}..." | tee -a
$LOG_FILE
INITIAL_ROWS=$(sqlplus -s / as sysdba << EOF
SET HEAD OFF FEED OFF;
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
SELECT COUNT(*) FROM ${USER_TO_IMPERSONATE}.${TABLE_NAME};
EXIT;
EOF
)
# Use tr to clean up whitespace for the comparison later
INITIAL_ROWS=$(echo "$INITIAL_ROWS" | tr -d '[:space:]')
echo "Initial Rows in ${TABLE_NAME}: ${INITIAL_ROWS}" | tee -a $LOG_FILE

# --- 3. DELETE THE ROWS FROM STOCKS ---
# Capture the current timestamp BEFORE the DELETE for our recovery
point.
RECOVERY_UNTIL_TIME=$(date +"%Y-%m-%d %H:%M:%S")

echo "3. Simulating disaster: Deleting 100,000 rows from ${TABLE_NAME}
in ${TARGET_PDB}." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
-- 1. Switch the container to the PDB
ALTER SESSION SET CONTAINER = ${TARGET_PDB};

-- 2. Temporarily change the current schema to the target user
ALTER SESSION SET CURRENT_SCHEMA = ${USER_TO_IMPERSONATE};

-- Run the delete command
DELETE FROM ${TABLE_NAME} WHERE ROWNUM <= 100000;

-- Commit the deletion to make it permanent
COMMIT;
EXIT;
EOF

# Check post-disaster row count
echo "Checking post-disaster row count..." | tee -a $LOG_FILE
DELETED_ROWS=$(sqlplus -s / as sysdba << EOF
```

```
SET HEAD OFF FEED OFF;
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
SELECT COUNT(*) FROM ${USER_TO_IMPERSONATE}.${TABLE_NAME};
EXIT;
EOF
)
DELETED_ROWS=$(echo "$DELETED_ROWS" | tr -d '[:space:]')
echo "POST-DISASTER Rows in ${TABLE_NAME}: ${DELETED_ROWS}" | tee -a
$LOG_FILE

# Check if the row count is less than the initial count (i.e., data was
deleted)
if [ "$DELETED_ROWS" -lt "$INITIAL_ROWS" ]; then
    echo "Disaster confirmed: Data deleted. Proceeding with recovery." |
tee -a $LOG_FILE
else
    echo "ERROR: Data deletion failed or no rows were deleted. Cannot
proceed with recovery simulation. Exiting." | tee -a $LOG_FILE
    exit 1
fi

# --- 4. PDB CLOSURE ---
echo "4. Closing PDB ${TARGET_PDB} before RMAN restore." | tee -a
$LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
-- The PDB must be closed before RMAN can perform a RESTORE/RECOVER.
ALTER PLUGGABLE DATABASE ${TARGET_PDB} CLOSE IMMEDIATE;
EXIT;
EOF


# --- 5. POINT-IN-TIME RECOVERY ---
echo "5. Starting RMAN PITR process..." | tee -a $LOG_FILE
echo "Recovery Target Time: ${RECOVERY_UNTIL_TIME}" | tee -a $LOG_FILE

rman target / log=$LOG_FILE append << RMAN_EOF
set echo on;
RUN {
    # 1. Ensure the CDB is in MOUNT state
    SHUTDOWN IMMEDIATE;
    STARTUP MOUNT;
```

```
    # 2. Restore the PDB datafiles from the last available backup
    RESTORE PLUGGABLE DATABASE ${TARGET_PDB} UNTIL TIME
"TO_DATE('${RECOVERY_UNTIL_TIME}','YYYY-MM-DD HH24:MI:SS')";

    # 3. FIX: Open the CDB in READ WRITE mode before PDB recovery
    ALTER DATABASE OPEN;

    # 4. Apply recovery (archive logs) up to the specified UNTIL TIME
    RECOVER PLUGGABLE DATABASE ${TARGET_PDB} UNTIL TIME
"TO_DATE('${RECOVERY_UNTIL_TIME}','YYYY-MM-DD HH24:MI:SS')";

    # 5. Open the PDB with RESETLOGS (required after PITR)
    ALTER PLUGGABLE DATABASE ${TARGET_PDB} OPEN RESETLOGS;
}
EXIT;
RMAN_EOF

RMAN_STATUS=$?

if [ $RMAN_STATUS -eq 0 ]; then
    echo "RMAN PITR completed successfully." | tee -a $LOG_FILE
else
    echo "ERROR: RMAN PITR failed (Status: $RMAN_STATUS). Check the RMAN
log in $LOG_FILE." | tee -a $LOG_FILE
    # Attempt to open the CDB root, but the PDB state may be unknown
    sqlplus -s / as sysdba << EOF >> $LOG_FILE
    -- Attempt to open the CDB if it's still mounted
    ALTER DATABASE OPEN;
    EXIT;
EOF
    exit 1
fi

# --- 6. VERIFICATION ---
echo "6. Checking final row count after recovery..." | tee -a $LOG_FILE
RECOVERED_ROWS=$(sqlplus -s / as sysdba << EOF
SET HEAD OFF FEED OFF;
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
SELECT COUNT(*) FROM ${USER_TO_IMPERSONATE}.${TABLE_NAME};
EXIT;
EOF
)
```

```
RECOVERED_ROWS=$(echo "$RECOVERED_ROWS" | tr -d '[:space:]')

echo "RECOVERED Rows in ${TABLE_NAME}: ${RECOVERED_ROWS}" | tee -a
$LOG_FILE

if [ "$RECOVERED_ROWS" -eq "$INITIAL_ROWS" ]; then
    echo "SUCCESS: Recovery confirmed. Initial count ($INITIAL_ROWS)
matches recovered count ($RECOVERED_ROWS)." | tee -a $LOG_FILE
else
    echo "FAILURE: Recovery may have partially failed. Initial count
($INITIAL_ROWS) does not match recovered count ($RECOVERED_ROWS)." | tee
-a $LOG_FILE
fi

echo
"========================================================================="
| tee -a $LOG_FILE
echo "Simulation finished at $(date)" | tee -a $LOG_FILE
echo
"========================================================================="
| tee -a $LOG_FILE

exit 0
```

G.2 Simulate Restoration and Recovery from Monthly Backup (simulate_restore_from_monthly.sh)

This script deletes 10,000 rows from the stocks table, and then restores them using the monthly backup. Gemini AI [2] was used for debugging this script.

```
#!/bin/bash
#-------------------------------------------------------------------------
# Script: simulate_restore_from_monthly.sh
# Purpose: Simulates a data loss event and recovers the entire CDB
#          (and ORCLPDB) using the RMAN monthly COLD backup tagged
#          'STABLE_MONTHLY_BU'.
# Execution: Run manually by the Oracle user.
#-------------------------------------------------------------------------

# --- Environment Setup ---
export ORACLE_SID=cdb1
export ORAENV_ASK=NO
```

```
export ORACLE_BASE=/u01/app/oracle

# PDB and Backup configuration
TARGET_PDB='ORCLPDB'
USER_TO_IMPERSONATE='STOCK_USER'
TABLE_NAME='STOCKS'
DELETE_ROWS=10000
BACKUP_TAG='STABLE_MONTHLY_BU' # Tag used in your monthly backup script

# Determine ORACLE_HOME
export ORACLE_HOME=$(cat /etc/oratab | grep -E "^$ORACLE_SID:" | cut -d
':' -f 2)
if [ -z "$ORACLE_HOME" ]; then
    export ORACLE_HOME="/u01/app/oracle/product/19.0.0/dbhome_1"
fi
export PATH=$ORACLE_HOME/bin:$PATH
export NLS_DATE_FORMAT='DD-MON-YYYY HH24:MI:SS'


# --- Logging Configuration ---
LOG_DIR="$ORACLE_BASE/admin/cdb1/logs/rman"
LOG_FILE="$LOG_DIR/monthly_restore_sim_$(date +\%Y\%m\%d_\%H\%M\%S).log"
mkdir -p $LOG_DIR

echo
"======================================================================="
| tee -a $LOG_FILE
echo "Starting Monthly Backup Restore Simulation at $(date)" | tee -a
$LOG_FILE
echo "Target PDB: ${TARGET_PDB} | Rows to Delete: ${DELETE_ROWS}" | tee
-a $LOG_FILE
echo "Using Backup Tag: ${BACKUP_TAG}" | tee -a $LOG_FILE
echo
"======================================================================="
| tee -a $LOG_FILE

# --- 1. INITIALIZATION: Ensure PDB is open for read and write ---
echo "1. Initializing: Ensuring PDB ${TARGET_PDB} is open for READ
WRITE." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
ALTER PLUGGABLE DATABASE ${TARGET_PDB} OPEN;
EXIT;
```

```
EOF


# --- 2. PRE-CHECK: Get the initial row count ---
echo "2. Checking initial row count in ${TARGET_PDB}..." | tee -a
$LOG_FILE
INITIAL_ROWS=$(sqlplus -s / as sysdba << EOF
SET HEAD OFF FEED OFF;
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
SELECT COUNT(*) FROM ${USER_TO_IMPERSONATE}.${TABLE_NAME};
EXIT;
EOF
)
INITIAL_ROWS=$(echo "$INITIAL_ROWS" | tr -d '[:space:]')
echo "Initial Rows in ${TABLE_NAME}: ${INITIAL_ROWS}" | tee -a $LOG_FILE

# --- 3. Delete the rows ---
# Capture the current timestamp BEFORE the DELETE for our recovery
point.
RECOVERY_UNTIL_TIME=$(date +"%Y-%m-%d %H:%M:%S")

echo "3. Simulating disaster: Deleting ${DELETE_ROWS} rows from
${TABLE_NAME}." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
-- Switch the container and user schema
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
ALTER SESSION SET CURRENT_SCHEMA = ${USER_TO_IMPERSONATE};

-- Run the delete command
DELETE FROM ${TABLE_NAME} WHERE ROWNUM <= ${DELETE_ROWS};

-- Commit the deletion to make it permanent
COMMIT;
EXIT;
EOF

# Check post-disaster row count
echo "Checking post-disaster row count..." | tee -a $LOG_FILE
DELETED_ROWS=$(sqlplus -s / as sysdba << EOF
SET HEAD OFF FEED OFF;
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
SELECT COUNT(*) FROM ${USER_TO_IMPERSONATE}.${TABLE_NAME};
```

```
EXIT;
EOF
)
DELETED_ROWS=$(echo "$DELETED_ROWS" | tr -d '[:space:]')
echo "POST-DISASTER Rows in ${TABLE_NAME}: ${DELETED_ROWS}" | tee -a
$LOG_FILE

# Validation check
if [ "$DELETED_ROWS" -lt "$INITIAL_ROWS" ]; then
    echo "Disaster confirmed: Data deleted. Proceeding with full CDB
recovery." | tee -a $LOG_FILE
else
    echo "ERROR: Data deletion failed or no rows were deleted. Cannot
proceed. Exiting." | tee -a $LOG_FILE
    exit 1
fi

# --- 4. CDB SHUTDOWN (REQUIRED FOR FULL CDB RESTORE) ---
echo "4. Shutting down the CDB for full restore operation." | tee -a
$LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
SHUTDOWN IMMEDIATE;
EXIT;
EOF


# --- 5. RMAN STUFF: FULL CDB RESTORE AND RECOVERY ---
echo "5. Starting RMAN FULL CDB RESTORE using tag ${BACKUP_TAG}..." |
tee -a $LOG_FILE
echo "Recovery Target Time: ${RECOVERY_UNTIL_TIME}" | tee -a $LOG_FILE

rman target / log=$LOG_FILE append << RMAN_EOF
set echo on;
RUN {
    # 1. Start the CDB in MOUNT state
    STARTUP MOUNT;

    # Allocate channels for parallel I/O
    ALLOCATE CHANNEL c1 DEVICE TYPE DISK;
    ALLOCATE CHANNEL c2 DEVICE TYPE DISK;
    ALLOCATE CHANNEL c3 DEVICE TYPE DISK;
    ALLOCATE CHANNEL c4 DEVICE TYPE DISK;
```

```
    # 2. Restore the full database using monthly backup TAG.
    RESTORE DATABASE FROM TAG '${BACKUP_TAG}' UNTIL TIME
"TO_DATE('${RECOVERY_UNTIL_TIME}','YYYY-MM-DD HH24:MI:SS')";

    # 3. Recover the database by applying archive logs up to the UNTIL
TIME.
    RECOVER DATABASE UNTIL TIME
"TO_DATE('${RECOVERY_UNTIL_TIME}','YYYY-MM-DD HH24:MI:SS')";

    # 4. Open the CDB with RESETLOGS since we are not
    #    bringing the db back to the exact state it was in.
    ALTER DATABASE OPEN RESETLOGS;

    # Release the channels
    RELEASE CHANNEL c1;
    RELEASE CHANNEL c2;
    RELEASE CHANNEL c3;
    RELEASE CHANNEL c4;
}
EXIT;
RMAN_EOF

RMAN_STATUS=$?

if [ $RMAN_STATUS -eq 0 ]; then
    echo "RMAN CDB Restore/Recovery completed successfully." | tee -a
$LOG_FILE
else
    echo "ERROR: RMAN CDB Recovery failed (Status: $RMAN_STATUS). Check
the RMAN log in $LOG_FILE." | tee -a $LOG_FILE
    exit 1
fi

# --- 6. CHECK IF RECOVERY WORKED ---
echo "6. Checking final row count after recovery..." | tee -a $LOG_FILE
RECOVERED_ROWS=$(sqlplus -s / as sysdba << EOF
SET HEAD OFF FEED OFF;
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
-- The CDB OPEN RESETLOGS command implicitly opens the PDBs, so no need
to explicitly open it here.
SELECT COUNT(*) FROM ${USER_TO_IMPERSONATE}.${TABLE_NAME};
```

```
EXIT;
EOF
)
RECOVERED_ROWS=$(echo "$RECOVERED_ROWS" | tr -d '[:space:]')

echo "RECOVERED Rows in ${TABLE_NAME}: ${RECOVERED_ROWS}" | tee -a
$LOG_FILE

if [ "$RECOVERED_ROWS" -eq "$INITIAL_ROWS" ]; then
    echo "SUCCESS: Recovery confirmed. Initial count ($INITIAL_ROWS)
matches recovered count ($RECOVERED_ROWS)." | tee -a $LOG_FILE
else
    echo "FAILURE: Recovery may have partially failed. Initial count
($INITIAL_ROWS) does not match recovered count ($RECOVERED_ROWS)." tee
-a $LOG_FILE
fi

echo
"========================================================================="
| tee -a $LOG_FILE
echo "Simulation finished at $(date)" | tee -a $LOG_FILE
echo
"========================================================================="
| tee -a $LOG_FILE

exit 0
```

G.3 Simulate Restoration and Recovery from Most Recent Yearly Backup (simulate_restore_from_yearly.sh)

This script deletes 10,000 rows from the stocks table, and then restores them using the most recent yearly backup. Gemini AI [2] was used for debugging this script.

```
#!/bin/bash
#-------------------------------------------------------------------------
# Script: simulate_restore_from_yearly.sh
# Purpose: Simulates a data loss event and recovers the entire CDB
#          (and ORCLPDB) using the RMAN YEARLY COLD backup tagged
#          'STABLE_YEARLY_BU'.
# Execution: Run manually by the Oracle user.
#-------------------------------------------------------------------------
```

```
# --- Environment Setup ---
export ORACLE_SID=cdb1
export ORAENV_ASK=NO
export ORACLE_BASE=/u01/app/oracle

# PDB and Backup configuration
TARGET_PDB='ORCLPDB'
USER_TO_IMPERSONATE='STOCK_USER'
TABLE_NAME='STOCKS'
DELETE_ROWS=10000
BACKUP_TAG='STABLE_YEARLY_BU'

# Determine ORACLE_HOME
export ORACLE_HOME=$(cat /etc/oratab | grep -E "^$ORACLE_SID:" | cut -d
':' -f 2)
if [ -z "$ORACLE_HOME" ]; then
    export ORACLE_HOME="/u01/app/oracle/product/19.0.0/dbhome_1"
fi
export PATH=$ORACLE_HOME/bin:$PATH
export NLS_DATE_FORMAT='DD-MON-YYYY HH24:MI:SS'


# --- Logging Configuration ---
LOG_DIR="$ORACLE_BASE/admin/cdb1/logs/rman"
LOG_FILE="$LOG_DIR/yearly_restore_sim_$(date +\%Y\%m\%d_\%H\%M\%S).log"
mkdir -p $LOG_DIR

echo
"======================================================================="
| tee -a $LOG_FILE
echo "Starting YEARLY Backup Restore Simulation at $(date)" | tee -a
$LOG_FILE
echo "Target PDB: ${TARGET_PDB} | Rows to Delete: ${DELETE_ROWS}" | tee
-a $LOG_FILE
echo "Using Backup Tag: ${BACKUP_TAG}" | tee -a $LOG_FILE
echo
"======================================================================="
| tee -a $LOG_FILE

# --- 1. INITIALIZATION: Ensure PDB is open for read and write ---
```

```
echo "1. Initializing: Ensuring PDB ${TARGET_PDB} is open for READ
WRITE." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
ALTER PLUGGABLE DATABASE ${TARGET_PDB} OPEN;
EXIT;
EOF

# --- 2. PRE-CHECK: Get initial row count ---
echo "2. Checking initial row count in ${TARGET_PDB}..." | tee -a
$LOG_FILE
INITIAL_ROWS=$(sqlplus -s / as sysdba << EOF
SET HEAD OFF FEED OFF;
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
SELECT COUNT(*) FROM ${USER_TO_IMPERSONATE}.${TABLE_NAME};
EXIT;
EOF
)
INITIAL_ROWS=$(echo "$INITIAL_ROWS" | tr -d '[:space:]')
echo "Initial Rows in ${TABLE_NAME}: ${INITIAL_ROWS}" | tee -a $LOG_FILE

# --- 3. DISASTER SIMULATION ---
# Capture the current timestamp BEFORE the DELETE for our recovery
point.
RECOVERY_UNTIL_TIME=$(date +"%Y-%m-%d %H:%M:%S")

echo "3. Simulating disaster: Deleting ${DELETE_ROWS} rows from
${TABLE_NAME}." | tee -a $LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
-- Switch the container and user schema
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
ALTER SESSION SET CURRENT_SCHEMA = ${USER_TO_IMPERSONATE};

-- Run the delete command
DELETE FROM ${TABLE_NAME} WHERE ROWNUM <= ${DELETE_ROWS};

-- Commit the deletion to make it permanent
COMMIT;
EXIT;
EOF

# Check post-disaster row count
echo "Checking post-disaster row count..." | tee -a $LOG_FILE
```

```
DELETED_ROWS=$(sqlplus -s / as sysdba << EOF
SET HEAD OFF FEED OFF;
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
SELECT COUNT(*) FROM ${USER_TO_IMPERSONATE}.${TABLE_NAME};
EXIT;
EOF
)
DELETED_ROWS=$(echo "$DELETED_ROWS" | tr -d '[:space:]')
echo "POST-DISASTER Rows in ${TABLE_NAME}: ${DELETED_ROWS}" | tee -a
$LOG_FILE

# Validation check
if [ "$DELETED_ROWS" -lt "$INITIAL_ROWS" ]; then
    echo "Disaster confirmed: Data deleted. Proceeding with full CDB
recovery." | tee -a $LOG_FILE
else
    echo "ERROR: Data deletion failed or no rows were deleted. Cannot
proceed. Exiting." | tee -a $LOG_FILE
    exit 1
fi

# --- 4. CDB SHUTDOWN ---
echo "4. Shutting down the CDB for full restore operation." | tee -a
$LOG_FILE
sqlplus -s / as sysdba << EOF >> $LOG_FILE
SHUTDOWN IMMEDIATE;
EXIT;
EOF

# --- 5. FULL CDB RESTORE AND RECOVERY ---
echo "5. Starting RMAN FULL CDB RESTORE using tag ${BACKUP_TAG}..." |
tee -a $LOG_FILE
echo "Recovery Target Time: ${RECOVERY_UNTIL_TIME}" | tee -a $LOG_FILE

rman target / log=$LOG_FILE append << RMAN_EOF
set echo on;
RUN {
    # 1. Start the CDB in MOUNT state
    STARTUP MOUNT;

    # Allocate channels for parallel I/O
    ALLOCATE CHANNEL c1 DEVICE TYPE DISK;
```

```
    ALLOCATE CHANNEL c2 DEVICE TYPE DISK;
    ALLOCATE CHANNEL c3 DEVICE TYPE DISK;
    ALLOCATE CHANNEL c4 DEVICE TYPE DISK;

    # 2. Restore the full database using yearly backup TAG.
    # We use 'FROM TAG' for robust RMAN parsing.
    RESTORE DATABASE FROM TAG '${BACKUP_TAG}' UNTIL TIME
"TO_DATE('${RECOVERY_UNTIL_TIME}','YYYY-MM-DD HH24:MI:SS')";

    # 3. apply archive logs up to the UNTIL TIME.
    RECOVER DATABASE UNTIL TIME
"TO_DATE('${RECOVERY_UNTIL_TIME}','YYYY-MM-DD HH24:MI:SS')";

    # 4. Open the CDB with RESETLOGS
    ALTER DATABASE OPEN RESETLOGS;

    # Release the channels
    RELEASE CHANNEL c1;
    RELEASE CHANNEL c2;
    RELEASE CHANNEL c3;
    RELEASE CHANNEL c4;
}
EXIT;
RMAN_EOF

RMAN_STATUS=$?

if [ $RMAN_STATUS -eq 0 ]; then
    echo "RMAN CDB Restore/Recovery completed successfully." | tee -a
$LOG_FILE
else
    echo "ERROR: RMAN CDB Recovery failed (Status: $RMAN_STATUS). Check
the RMAN log in $LOG_FILE." | tee -a $LOG_FILE
    exit 1
fi

# --- 6. CHECK IF RECOVERY WORKED ---
echo "6. Checking the final row count after recovery..." | tee -a
$LOG_FILE
RECOVERED_ROWS=$(sqlplus -s / as sysdba << EOF
SET HEAD OFF FEED OFF;
ALTER SESSION SET CONTAINER = ${TARGET_PDB};
```

```
-- The CDB OPEN RESETLOGS command implicitly opens the PDBs, so no need
to explicitly open it here.
SELECT COUNT(*) FROM ${USER_TO_IMPERSONATE}.${TABLE_NAME};
EXIT;
EOF
)
RECOVERED_ROWS=$(echo "$RECOVERED_ROWS" | tr -d '[:space:]')

echo "RECOVERED Rows in ${TABLE_NAME}: ${RECOVERED_ROWS}" | tee -a
$LOG_FILE

if [ "$RECOVERED_ROWS" -eq "$INITIAL_ROWS" ]; then
    echo "SUCCESS: Recovery confirmed. Initial count ($INITIAL_ROWS)
matches recovered count ($RECOVERED_ROWS)." | tee -a $LOG_FILE
else
    echo "FAILURE: Recovery may have partially failed. Initial count
($INITIAL_ROWS) does not match recovered count ($RECOVERED_ROWS)." tee
-a $LOG_FILE
fi

echo
"========================================================================="
| tee -a $LOG_FILE
echo "Simulation finished at $(date)" | tee -a $LOG_FILE
echo
"========================================================================="
| tee -a $LOG_FILE

exit 0
```

Appendix H – Performance Monitoring and Tuning Scripts

H.1 Scheduler Job Creation Scripts

These scripts were created with the help of Gemini AI [2] for debugging, and were built using information from the Lesson 13: Performance Management [7] powerpoint slides shown in class.
Location: All jobs are created under the MONITORING user schema.

```
# --- 1. Daily fix of unusable indexes – 02:00 AM –
      EXEC DBMS_SCHEDULER.CREATE_JOB('MON_INVALID_FIX',
      job_type=>'PLSQL_BLOCK', job_action=>'BEGIN FOR i IN (SELECT
      index_name FROM user_indexes WHERE status=''UNUSABLE'') LOOP
      EXECUTE IMMEDIATE ''ALTER INDEX ''||i.index_name||'' REBUILD
```

```
ONLINE''; END LOOP; END;', repeat_interval=>'FREQ=DAILY;
BYHOUR=2', enabled=>TRUE, comments=>'Daily fix of unusable indexes
only');

# --- 2. Kill my own zombie sessions – 01:00 AM –
EXEC DBMS_SCHEDULER.CREATE_JOB('MON_KILL_HOGS',
job_type=>'PLSQL_BLOCK', job_action=>q'[BEGIN FOR r IN (SELECT
sid, serial# FROM v$session WHERE username=USER AND
((status='INACTIVE' AND last_call_et>28800) OR (status='ACTIVE'
AND last_call_et>7200)) AND type='USER') LOOP EXECUTE IMMEDIATE
'ALTER SYSTEM KILL SESSION '''||r.sid||','||r.serial#||'''
IMMEDIATE'; END LOOP; END;]', repeat_interval=>'FREQ=DAILY;
BYHOUR=1', enabled=>TRUE, comments=>'Kill only MY idle >8h or
running >2h sessions');

# --- 3. Weekly schema statistics – Saturday 03:00 AM –
EXEC DBMS_SCHEDULER.CREATE_JOB('MON_GATHER_STATS',
job_type=>'PLSQL_BLOCK', job_action=>q'[BEGIN
DBMS_STATS.GATHER_SCHEMA_STATS(ownname=>USER,
estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE, cascade=>TRUE,
degree=>4, options=>'GATHER AUTO'); END;]',
repeat_interval=>'FREQ=WEEKLY; BYDAY=SAT; BYHOUR=3',
enabled=>TRUE, comments=>'Weekly schema statistics - Saturday
03:00');


# --- 4. Tablespace >=85% alert – 07:00 AM –
EXEC DBMS_SCHEDULER.CREATE_JOB('MON_TS_ALERT',
job_type=>'PLSQL_BLOCK', job_action=>q'[DECLARE v VARCHAR2(1000);
BEGIN SELECT LISTAGG(tablespace_name||'
'||ROUND(used_percent,1)||'%',', ') INTO v FROM
dba_tablespace_usage_metrics WHERE used_percent>=85; IF v IS NOT
NULL THEN RAISE_APPLICATION_ERROR(-20001,''TS Alert: ''||v); END
IF; END;]', repeat_interval=>'FREQ=DAILY; BYHOUR=7',
enabled=>TRUE, comments=>'All tablespaces >=85% alert');


# --- 5. Nightly top-20 SQL snapshot – 23:00 –
EXEC DBMS_SCHEDULER.CREATE_JOB('MON_TOP_SQL',
job_type=>'PLSQL_BLOCK', job_action=>q'[BEGIN EXECUTE IMMEDIATE
''TRUNCATE TABLE monitoring.top_sql_log''; INSERT INTO
monitoring.top_sql_log(sql_id,plan_hash,execs,disk_reads,lio,cpu_s
```

```
ec,ela_sec) SELECT
sql_id,plan_hash_value,executions,disk_reads,buffer_gets,ROUND(cpu
_time/1000000,2),ROUND(elapsed_time/1000000,2) FROM v$sql WHERE
parsing_schema_name=USER AND executions>0 AND ROWNUM<=20 ORDER BY
cpu_time DESC; COMMIT; END;]', repeat_interval=>'FREQ=DAILY;
BYHOUR=23', enabled=>TRUE, comments=>'Nightly top-20 SQL
snapshot');
```

H.2 Supporting Table – monitoring.top_sql_log

- Location: MONITORING user schema
  - Create once as a GLOBAL TEMPORARY TABLE
- Data is preserved across commits but truncated nightly by MON_TOP_SQL
- Exists only for the lifetime of the session or until explicitly dropped

```
CREATE GLOBAL TEMPORARY TABLE monitoring.top_sql_log (
snap_time DATE DEFAULT SYSDATE, sql_id VARCHAR2(13), plan_hash
NUMBER, execs NUMBER, disk_reads NUMBER, lio NUMBER, cpu_sec
NUMBER, ela_sec NUMBER) ON COMMIT PRESERVE ROWS;
```

H.3 Daily Dashboard Script – morning_check.sql

Location: $ORACLE_BASE/admin/orcl/scripts/morning_check.sql

```
-- Run with: @morning_check.sql
SET PAGESIZE 100
SET LINESIZE 200
COL job_name     FORMAT A30
COL enabled      FORMAT A8
COL next_run     FORMAT A20
COL time         FORMAT A19
COL status       FORMAT A8
COL msg          FORMAT A100
COL used         FORMAT A10

SELECT '=== TODAY''S SCHEDULE ===' FROM dual;

SELECT job_name,CASE WHEN enabled = 'TRUE' THEN 'YES' ELSE 'NO' END AS
enabled,TO_CHAR(next_run_date, 'DY HH24:MI') AS next_run, failure_count
FROM user_scheduler_jobs
ORDER BY next_run_date;

SELECT '=== FAILURES LAST 24H ===' FROM dual;
```

```
SELECT TO_CHAR(log_date,'YYYY-MM-DD HH24:MI') AS time,job_name, status,
SUBSTR(additional_info,1,100) AS msg
FROM   user_scheduler_job_run_details
WHERE  log_date >= SYSDATE-1
AND  status = 'FAILED'
ORDER BY log_date DESC;

SELECT '=== TABLESPACES >= 80% ===' FROM dual;
SELECT tablespace_name,ROUND(used_percent,1) || '%' AS used
FROM   dba_tablespace_usage_metrics
WHERE  used_percent >= 80
ORDER BY used_percent DESC;

SELECT '=== YESTERDAY''S TOP-5 SQL ===' FROM dual;

SELECT sql_id, execs, ROUND(cpu_sec,2) AS cpu_sec, ROUND(ela_sec,2) AS
ela_sec, ROUND(lio/execs) AS lio_per_exec
FROM   monitoring.top_sql_log
WHERE  TRUNC(snap_time) = TRUNC(SYSDATE-1)
ORDER BY cpu_sec DESC
FETCH FIRST 5 ROWS ONLY;

SELECT '=== SUMMARY ===',
'Jobs defined       : ' || COUNT()
FROM user_scheduler_jobs
UNION ALL

SELECT '=== SUMMARY ===',
'Failed runs 24h    : ' || COUNT()
FROM user_scheduler_job_run_details WHERE log_date >= SYSDATE-1 AND
status='FAILED'
UNION ALL

SELECT '=== SUMMARY ===',
'Tablespaces >=80%  : ' || COUNT()
FROM dba_tablespace_usage_metrics WHERE used_percent >= 80
UNION ALL

SELECT '=== SUMMARY ===',
'Top-SQL rows today : ' || COUNT()
FROM monitoring.top_sql_log WHERE TRUNC(snap_time)=TRUNC(SYSDATE);
```

```
# --- Buffer Cache Advisor – is current size good? –
      SELECT '=== BUFFER CACHE ADVICE ===', 'Current DB_CACHE_SIZE ≈ '
      || ROUND(SUM(current_size)/1024/1024) || ' MB' AS info
      FROM   v$memory_dynamic_components
      WHERE  component LIKE '%buffer cache%'
      UNION ALL

      SELECT 'Size (MB): ' || size_for_estimate,
      'Est Physical Read Factor: ' || estd_physical_read_factor
      FROM   v$db_cache_advice
      WHERE  estd_physical_read_factor > 1
      OR  size_for_estimate = (SELECT current_size/1024/1024/8 FROM
      v$buffer_pool WHERE name='DEFAULT')
      ORDER BY size_for_estimate;

# --- Key system statistics (compare day-to-day) –
      SELECT '=== KEY SYSTEM STATS ===',
      name || ': ' || TO_CHAR(value, '999,999,999,990') AS stat
      FROM   v$sysstat
      WHERE  name IN ('physical reads',
      'physical writes',
      'db block gets',
      'consistent gets',
      'session logical reads',
      'table scan rows gotten',
      'table scans (short tables)',
      'table scans (long tables)')
      ORDER BY name;
```

H.4 Performance Monitoring Response Guide

Run morning_check.sql every morning.
When all jobs healthy:

- No rows in “Failures last 24h”
- No tablespaces listed under “Tablespaces ≥80%”
- Yesterday’s top-5 SQL populated
- All summary numbers = 0

MON_INVALID_FIX – Daily unusable index fix (02:00 AM)

- Expected when OK: Runs silently

- If listed as FAILED: One or more indexes were unusable — the job already rebuilt them (this is successful auto-repair, not an error)
- Action: None required — verify with:

```
SELECT owner, index_name, status
FROM dba_indexes
WHERE status = 'UNUSABLE';
```

MON_KILL_HOGS – Daily zombie session killer (01:00 AM)

- Expected when OK: Runs silently
- If FAILED: Could not kill a session
- Action: Check ADDITIONAL_INFO → manually kill the session:

```
SELECT sid, serial#, username, status, last_call_et/3600 AS hours
FROM   v$session
WHERE  username = 'MONITORING'
AND  (status = 'INACTIVE' AND last_call_et > 28800
OR status = 'ACTIVE'   AND last_call_et > 7200);
```

- Kill example

```
ALTER SYSTEM KILL SESSION '123,45678' IMMEDIATE;
```

MON_GATHER_STATS – Weekly statistics refresh (Saturday 03:00 AM)

- Expected when OK: Runs silently
- If FAILED: Statistics gathering failed
- Action: Rerun manually:

```
BEGIN
DBMS_STATS.GATHER_SCHEMA_STATS(
ownname          => 'MONITORING',
estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE,
cascade          => TRUE,
degree           => 4,
options          => 'GATHER AUTO'
);
END;
/
```

MON_TS_ALERT – Daily tablespace alert (07:00 AM)

- Expected when OK: Runs silently
- If FAILED: One or more tablespaces ≥85% full
- Action: ADDITIONAL_INFO contains exact list → add datafile or resize:

```
SELECT tablespace_name,
ROUND(used_percent,1) || '%' AS used
FROM   dba_tablespace_usage_metrics
```

```
ORDER BY used_percent DESC;
```

- Example: Add 5GB datafile to STOCKS_DATA

```
ALTER TABLESPACE STOCKS_DATA ADD DATAFILE '+DATA' SIZE 5G
AUTOEXTEND ON;
```

MON_TOP_SQL – Nightly top-20 SQL snapshot (23:00)

- Expected when OK: Runs silently and populates monitoring.top_sql_log
- If FAILED: Insert/truncate failed
- Action: Verify table and rerun manually:

```
SELECT COUNT(*) FROM monitoring.top_sql_log;
```

- Manual rerun

```
BEGIN
      EXECUTE IMMEDIATE 'TRUNCATE TABLE monitoring.top_sql_log';
      INSERT INTO
      monitoring.top_sql_log(sql_id,plan_hash,execs,disk_reads,lio
      ,cpu_sec,ela_sec)
      SELECT
      sql_id,plan_hash_value,executions,disk_reads,buffer_gets,
      ROUND(cpu_time/1000000,2),ROUND(elapsed_time/1000000,2)
      FROM v$sql
      WHERE parsing_schema_name='MONITORING' AND executions>0 AND
      ROWNUM<=20
      ORDER BY cpu_time DESC;
      COMMIT;
END;
/
```

Appendix I: RMAN Script for Structural Backup

The following script was created with the use of information from Lesson 5: Using RMAN to Create Backups [5], and was debugged with the help of Gemini AI [2]:

```
RUN {
  ALLOCATE CHANNEL disk1 DEVICE TYPE DISK FORMAT
'/mnt/backups/CONTROL_FILE_TEST_%U';

  BACKUP
    AS COMPRESSED BACKUPSET
    CURRENT CONTROLFILE
    ;
```

```
  RELEASE CHANNEL disk1;
}
```

References

[1] Y. Khmelevsky, *AlgorithmicTrading: FMP API / 503 Stocks folder*, GitHub repository, accessed Nov. 14, 2025. [Online]. Available: https://github.com/youry/AlgorithmicTrading/tree/main/Data/July%201%2C%202023%20to%20October%2024%2C%2025/FMP%20API/503%20Stocks

[2] Google, “Gemini,” Google LLC, accessed Nov. 17-24, 2025. [Online]. Available: https://gemini.google.com/

[3] Oracle, “Backup and Recovery Concepts,” presented to Class, Okanagan College, Kelowna, BC, Canada. October, 2025. [Powerpoint slides]. Available: https://mymoodle.okanagan.bc.ca/pluginfile.php/4636644/mod_resource/content/0/Less14_BR_Concepts.pdf

[4] Oracle, “Configuring for Recoverability,” presented to Class, Okanagan College, Kelowna, BC, Canada. November, 2025. [Powerpoint slides]. Available: https://mymoodle.okanagan.bc.ca/pluginfile.php/4636662/mod_resource/content/0/les_02_config_rec.pdf

[5] Oracle, “Using RMAN to Create Backups,” presented to Class, Okanagan College, Kelowna, BC, Canada. November, 2025. [Powerpoint slides]. Available: https://mymoodle.okanagan.bc.ca/pluginfile.php/4636579/mod_resource/content/0/les_05_create_bu.pdf

[6] Oracle, “Using RMAN to Perform Recovery,” presented to Class, Okanagan College, Kelowna, BC, Canada. November, 2025. [Powerpoint slides]. Available: https://mymoodle.okanagan.bc.ca/pluginfile.php/4636586/mod_resource/content/0/les_07_RMAN_rec.pdf

[7] Oracle, “Performance Management,” presented to Class, Okanagan College, Kelowna, BC, Canada. November, 2025. [Powerpoint slides]. Available:

https://mymoodle.okanagan.bc.ca/pluginfile.php/4636639/mod_resource/content/0/Less13_Performance24Feb2020.ppt