Elaboration

COSC 434 Database Administration

Okanagan College
Computer Science Department
Kelowna, BC

November 10 2025

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

Table of Contents

System Overview

Purpose of Database

The database stores historical stock market data to support machine learning models for financial forecasting and predictive analysis. In a production environment, the database would receive real-time updates at 5 minute intervals to maintain current market information.

Key Users and Applications

Key users include machine learning developers who query the database to extract historical stock data for model training and analysis via Python applications. The five database administrators also serve as key users, responsible for database maintenance and optimization.

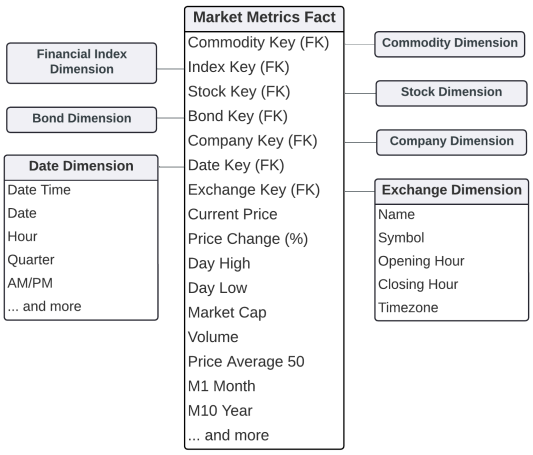
Database Design

The database structure is based on the best structure for machine learning models. This could take the form of a data warehouse with a star schema, or rather a single flat table with all data in it.

Since the database is hosted on the Digital Research Alliance of Canada's PostgreSQL infrastructure rather than Oracle, the design cannot be implemented exactly as specified. However, for instructional purposes, the database is planned and documented as if it were deployed in an Oracle environment.

Logical Design

A potential star schema for this, informed by work in the previous data warehousing course and research [1], could be:



However, such a design may be overcomplicated for the purposes of this project. Additionally, the client has seemed to plan for one single flat table.

So, ultimately, the database will simply be a flat table like so:

stocks	
id	INTEGER
symbol	VARCHAR(10)
date	TIMESTAMP
open	NUMERIC(10,4)
low	NUMERIC(10,4)
high	NUMERIC(10,4)
close	NUMERIC(10,4)
volume	BIGINT

Using the CSV files provided in the GitHub for the Algorithmic Trading project, this table would have 29,381,155 rows of data.

SQL to create this table:

```
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;
```

The stocks table uses data types chosen for precision, scalability, and compatibility with PostgreSQL. The id column uses the SERIAL type, which automatically generates unique identifiers for each record. The symbol column is defined as VARCHAR(10) to store stock ticker symbols of varying lengths. The date column uses the TIMESTAMP type to record the exact time of each data point. The price-related columns (open, low, high, close) use the NUMERIC(10,4) type to ensure high precision and prevent rounding errors common with floating-point types, an essential consideration for financial data. Finally, the volume column is defined as BIGINT to accommodate large trading volumes that may exceed the limits of a standard integer type.

Here is a sample of the kind of data that would be stored in this table:

date, open, low, high, close, volume, symbol
2023-07-11 15:45:00,187.47,187.46,188.27,188.06,4729821,AAPL
2023-07-11 15:30:00,187.13,186.97,187.47,187.46,1534358,AAPL
2023-07-11 15:15:00,187.215,187.0601,187.36,187.1372,1238542,AAPL
2023-07-11 15:00:00,187.35,187.155,187.395,187.21,996502,AAPL

Physical Design

Tablespace Design

Primary Data Tablespace

- STOCKS_DATA - Main tablespace for the stocks table
    - Purpose: Store all stock market data with appropriate storage parameters
    - Initial Size: 4GB for 29,381,155 rows of data.
    - Autoextend: Enabled at 1-100GB increments
    - Block Size: 8KB (default)
    - Justification: Separating application data from system tablespaces improves manageability and backup flexibility

Index Tablespace

- STOCKS_INDEX - Dedicated tablespace for indexes
    - Purpose: Store all indexes separately from table data
    - Initial Size: 5GB
    - Autoextend: Enabled at 1-50GB increments
    - Justification: Separating indexes from table data reduces I/O contention and simplifies performance tuning.

Temporary Tablespace

- TEMP - For sort operations during queries
    - Purpose: Handle sorting and temporary operations for ML data extraction queries
    - Initial Size: 2GB
    - Autoextend: Enabled at 1-20GB increments
    - Justification: ML developers may run complex analytical queries requiring substantial temporary space

SQL to Create Tablespaces

```
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;

CREATE TEMPORARY TABLESPACE TEMP
TEMPFILE 'temp01.dbf'
SIZE 2G
AUTOEXTEND ON NEXT 1G MAXSIZE 10G;
```

Datafile Strategy

STOCKS_DATA Tablespace Datafiles

- Multiple datafiles across different physical disks (if available)
    - Example:
        - stocks_data01.dbf - 2GB on Disk 1
        - stocks_data02.dbf - 2GB on Disk 2
- Justification: Distributing datafiles across physical storage improves I/O parallelism

STOCKS_INDEX Tablespace Datafiles

- Multiple index datafiles across different physical disks:
    - stocks_idx01.dbf - 2.5GB
    - stocks_idx02.db - 2.5GB
- Justification: Helps spread I/O, which improves index query performance

Control Files

Minimum of 3 copies of the control file, stored on separate physical locations:

- /u01/oradata/stocks/control01.ctl
- /u02/oradata/stocks/control02.ctl
- /u03/oradata/stocks/control03.ctl

Redo Log Configuration

Minimum of 3 redo log groups with 2 members each (multiplexed)

- Group 1: /u01/oradata/stocks/redo01a.log, /u02/oradata/stocks/redo01b.log
- Group 2: /u01/oradata/stocks/redo02a.log, /u02/oradata/stocks/redo02b.log
- Group 3: /u01/oradata/stocks/redo03a.log, /u02/oradata/stocks/redo03b.log
  Each 500MB

ARCHIVELOGMODE will be enabled.

Index Design

Primary Indexes

a. Symbol Index (STOCKS_SYMBOL_IDX)
   - Refer to the Performance & Tuning section for detailed description.

b. Date Index (STOCKS_DATE_IDX)
   - Refer to the Performance & Tuning section for detailed description.

Composite Index

c. Symbol-Date Composite Index (STOCKS_SYMBOL_DATE_IDX)
   - Refer to the Performance & Tuning section for detailed description.

Partitioning Strategy

Range Partitioning by Date

- Partition the stocks table by date ranges (quarterly)
- PARTITION BY RANGE (date)
  - STOCKS_Q1_2025: Jan 1 - Mar 31, 2025
  - STOCKS_Q2_2025: Apr 1 - Jun 30, 2025
  - STOCKS_Q3_2025: Jul 1 - Sep 30, 2025
  - STOCKS_Q4_2025: Oct 1 - Dec 31, 2025

  This will improve query performance and allow us to backup and recover individual partitions independently.

Storage Parameters

Feature	**Configuration**	**Benefit**
Extent Management	LOCAL AUTOALLOCATE	Automatic extent sizing, no fragmentation
Segment Space Management	AUTO (ASSM)	Automatic free space management, reduced contention
Undo Management	AUTO	Automatic undo segment management
Block Size	8 KB (default)	Optimal for mixed workload
Statistics Gathering	Automatic (Oracle 19c default)	Keeps optimizer statistics current
Memory Management	Automatic (if configured)	Self-tuning memory allocation

Backup and Recovery Strategy

Backup Plan

Backup and Recovery Strategy
We will use Recovery Manager (RMAN) for all database backups, rather than user-managed backups, to leverage RMAN's automation, optimization, and recovery features.

Backup Responsibility:
Keona and Alex will be responsible for managing backup and recovery procedures. They will configure the flash recovery area (FRA) by setting the DB_RECOVERY_FILE_DEST parameter to a location which is on a separate disk from the working set of database files to avoid a single point of failure.

Backup Frequency and Method:
Backups will be performed daily using the Oracle-Suggested Backup Strategy, which provides an efficient balance between performance and recoverability. The process operates as follows:

- Perform a one-time, whole-database online incremental level 0 backup (the baseline backup).
- Schedule incremental level 1 backups for each subsequent day.
- Beginning on the third day, RMAN automatically applies the previous day's level 1 backup to the level 0 baseline, effectively updating it. Afterward, a new level 1 backup is taken for the current day.
- This cycle continues indefinitely, maintaining an up-to-date level 0 baseline and reducing the volume of backup data required each day.

Flash Recovery Area (FRA):
The FRA will be sized according to Oracle's recommendation - at least twice the size of the database - to accommodate one complete backup and all required archived redo log files.

Control File Backups:
Following any structural change to the database (such as adding or dropping datafiles or tablespaces), the control file will be backed up to the trace file using:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

This trace file will then be stored in the USER_DUMP_DEST location for additional protection.

Parallelism:

Backup parallelism will be configured to match the number of disks over which the backup destination is striped, ensuring optimal I/O performance during backup operations.

Retention Policy:
The RMAN retention policy will be configured using the CONFIGURE command with a recovery window-based policy, which is the best practice for most Oracle environments.
A 7-day recovery window will be maintained, allowing point-in-time recovery to any moment within the past week. This policy ensures the ability to recover from operational failures such as data corruption or accidental deletions.

Because the database stores continuously appended time-series financial data, older records are rarely modified. Therefore, while the 7-day RMAN recovery window is sufficient for operational recovery, long-term data retention will be managed separately through data archival.

Archived backups (e.g., weekly or monthly full backups) will be stored for 3-12 months to preserve historical market data and support analytical workloads. This dual approach balances short-term recoverability with efficient storage use.

Backup Type	**Frequency**	**Retention**	**Purpose**
Incremental Backups	Daily	7 days	Operational recovery
Full backups	Weekly	3-6 months	Historical recovery
Archived/long-term	Monthly	1-3 years	Compliance / analytics

Recovery Scenarios

To recover a table that has been purged or accidentally deleted, we will use Point-in-Time Recovery (PITR). PITR enables restoration of the database or specific objects to a precise moment prior to the data loss event, minimizing downtime and data inconsistency.

In addition, Oracle Flashback Technology provides fast and flexible methods for examining and restoring past states of data. These features allow administrators and users to view historical data, trace the causes of errors, and recover from them efficiently.

Error Analysis Tools:

Flashback Query (SELECT ... AS OF ...) - View data as it existed at a specific point in time.

Flashback Version Query (SELECT ... VERSIONS BETWEEN ...) - Retrieve multiple historical versions of data for change tracking.

Flashback Transaction Query - Inspect all changes made at the transaction level, useful for detailed error investigation.

Error Recovery Tools:

Flashback Transaction Backout - Rolls back a specific transaction and all dependent transactions while preserving database integrity.

Flashback Table - Restores one or more tables to their previous state at a chosen time, without affecting other database objects.

Flashback Drop - Recovers dropped tables from the recycle bin, along with their dependent objects such as indexes and triggers.

Minimizing Mean Time to Recover (MTTR):

To ensure rapid recovery from instance failures, we will configure an MTTR target and optimize redo log sizing.
The MTTR advisor will be enabled by setting the FAST_START_MTTR_TARGET parameter to a non-zero value representing the desired recovery time.
Given that the algorithmic trading data refreshes every 5 minutes, an MTTR target of 3-5 minutes will be appropriate to meet operational recovery objectives.

Maximizing Mean Time Between Failures (MTBF):

We will improve system reliability and extend MTBF by tuning instance recovery parameters and minimizing the distance between checkpoint positions and the end of the redo logs.

The database will operate in ARCHIVELOG mode, with archived redo logs directed to the flash recovery area.
To protect against redo log loss, redo log multiplexing will be implemented, ensuring that each log group maintains multiple members on separate disks.

As shown in Lesson 14, Backup and Recovery Concepts [2], here are some common scenarios requiring recovery, and how to handle them:

Typical Problems	Possible solutions
Statement Failure	
Attempts to enter invalid data into a table	Work with users to validate and correct data. Have triggers which validate data before it is entered into the table.

Attempts to perform operations with insufficient privileges	Provide appropriate object or system privileges
Attempts to allocate space that fail	- Enable resumeable space allocation - Increase owner quota - Add space to tablespace
Logic errors in applications	Work with developers to correct program errors
User Process Failure	
A user performs an abnormal disconnect	A DBA's action is not usually needed to resolve user process failures. Instance background processes roll back uncommitted changes and release locks. Watch for trends.
A user's session is abnormally terminated	
A user experiences a program error that terminates the session	
Network Failure	
Listener fails	Configure a backup listener and connect time failover
Network Interface Card (NIC) fails	Configure multiple network cards
Network connection fails	Configure a backup network connection
User Error	
User inadvertently deletes or modifies data	Roll back transaction and dependent transactions or rewind table
User drops a table	Recover table from recycle bin
Instance Failure	
Power Outage	Restart the instance by using the STARTUP command. Recovering from instance failure is automatic, including rolling forward changes in the redo logs and then rolling back any uncommitted transactions. Investigate the causes of failure by using the alert log, trace files, and Enterprise Manager.
Hardware failure	
Failure of one of the critical background processes	
Emergency shutdown procedures	
Media Failure	
Failure of disk drive	1. Restore the affected file from backup. 2. Inform the database about a new file location (if necessary).
Failure of disk controller	

Deletion or corruption of database file	3. Recover the file by applying redo information (if necessary).

Testing Backups

We will perform simulated recovery exercises involving the loss of key database components (such as the entire database, control files, or redo log files) to validate our backup and recovery procedures and ensure that the recovery methods described above function as intended.

Security Design

User Management

Each DBA will have their own user account to ensure it is clear who has performed what tasks (especially in audit files). NO account sharing will be allowed to ensure separation of responsibilities.

SYSDBA is required for RMAN operations, database startup/shutdown, and recovery, so those DBA's responsible for backup and recovery will need to be granted the role.

The ADVISOR role lets you access the advisor framework, which is useful for performance management (SQL Tuning Advisor, SQL Access Advisor, Memory Advisors).

The DBA accounts will be as follows:

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

We will create database non-system user accounts and give them tablespaces, quotas, and SetuSedifferent system and object privileges based on the needs of users gathering data for machine learning purposes.

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

Auditing and Compliance

Purpose

The goal of auditing in this project is to track user activity, detect unauthorized access, and maintain accountability among DBA and non-system users. Because our database is small and contains a single table (stocks), our focus will be on monitoring key actions such as logins, privilege use, and DML operations on that table.

Enable Standard Auditing

We will use standard database auditing at the system level, which allows auditing of statements, privileges, and schema objects using the AUDIT command.

To enable standard auditing, we will set the following initialization parameter and restart the database:

```
ALTER SYSTEM SET audit_trail = DB SCOPE=SPFILE;
SHUTDOWN IMMEDIATE;
STARTUP;
```

This setting (DB) stores audit records in the data dictionary, accessible via the DBA_AUDIT_TRAIL view in the SYS schema.

System Privilege Auditing
We will enable auditing for the use of critical system privileges by DBAs and developers. Auditing will be configured BY ACCESS (meaning each use is recorded individually).

Examples:

```
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;
```

These statements ensure that any administrative actions are recorded for accountability.

Object Auditing (Table Level)
Since the stocks table is the core of our database, we will audit DML activity performed on it, including SELECT, INSERT, UPDATE, and DELETE operations.

```
AUDIT SELECT, INSERT, UPDATE, DELETE
ON stocks
BY ACCESS;
```

This ensures that every access or modification to stock data is captured, especially by non-system users.

Login and Session Auditing
To track who connects to the database and when, we will enable session-level auditing:

```
AUDIT SESSION;
```

This records each user's login and logout, including unsuccessful login attempts.

Reviewing and Maintaining Audit Records
Audit data will be reviewed periodically through queries such as:

```
SELECT username, action_name, obj_name, timestamp
FROM dba_audit_trail
ORDER BY timestamp DESC;
```

To manage space and performance, older audit records will be archived and purged regularly:

```
DELETE FROM sys.aud$ WHERE timestamp# < SYSDATE - 30;
```

```
COMMIT;
```

Summary of Audit Focus

Category	Audit Type	Purpose
System privileges	AUDIT CREATE/DROP/ALTER SYSTEM	Monitor DBA actions
Object access	AUDIT SELECT/INSERT/UPDATE/DELETE ON stocks;	Track table activity
Logins/logouts	AUDIT SESSION	Track user connections
Privilege grants	AUDIT GRANT ANY PRIVILEGE	Monitor changes to user rights

Performance and Tuning Design

Memory Configuration

Memory Architecture & Optimization Goals

The System Global Area (SGA) holds shared structures:

- Buffer Cache (data blocks from STOCKS_DATA and STOCKS_INDEX)
- Shared Pool (SQL for symbol/date queries)
- Large Pool, Java Pool, Streams Pool

The Program Global Area (PGA) manages private session memory:

- Sort areas (for quarterly aggregations on volume, close)
- Hash areas (for analytics on open, high, low)

Performance Goal:

- Keep buffer cache hit ratio > 95% for STOCKS_DATA blocks (50MB initial).
- Avoid PGA sort-to-disk during temp segment usage in TEMP (100MB).
- Minimize physical I/O across multiple datafiles (stocks_data01.dbf, stocks_data02.dbf)

Recommended Strategy: Automatic Memory Management (AMM)

We will enable AMM to allow Oracle to dynamically shift memory between SGA and PGA based on query patterns. This is Oracle's recommended approach and eliminates manual tuning errors.

MEMORY_TARGET = 3G:

- Covers **50MB data** + **2GB indexes** + **100MB temp** + overhead.
- Allows **full caching of STOCKS_DATA** and **hot index blocks** in buffer cache.
- Supports **PGA for 100MB temp sorts** without disk spill.

Reasoning:

- **Data + Index = ~2.05GB** → fits in buffer cache with room for shared pool.
- **TEMP = 100MB** → PGA must handle full sort in memory.
- **Redo = 500MB x 3 groups** → not in SGA, but AMM leaves OS room.

MEMORY_MAX_TARGET = 4G:

- Headroom for quarterly partition growth.
- Restart required only if increasing max.

Buffer Cache Sizing (Within AMM)

We will not manually set DB_CACHE_SIZE. AMM + ASMM will auto-allocate based on access patterns

Expected Buffer Cache Behavior

Component	Size	Expected Cache Residency
STOCKS_DATA	50MB	100% cached (8KB blocks × ~6,250)
STOCKS_INDEX	2GB	Hot partitions (current quarter) cached
TEMP	100MB	Not in SGA - handled by PGA

Weekly Buffer Cache Advisor Check

1. EM → Server → Memory Advisors → Advice (Buffer Cache)
2. If estd_physical_read_factor > 1.0 at current allocation: increase MEMORY_TARGET by 500MB.

PGA Sizing (Within AMM)

PGA is critical for:

- Sorting quarterly data on volume, close
- Hash joins on open, high, low

- TEMP tablespace = 100MB → PGA must support in-memory sorts.
- AMM will increase PGA during high-activity queries.
- Monitor via EM → Top Sessions → PGA Memory.

Monitoring & Maintenance

Task	**Frequency**	**Tool**
Review Memory Advisors	Weekly	EM → Server → Memory Advisors
Run V$DB_CACHE_ADVICE	After MEMORY_TARGET change	SQL*Plus
Check PGA vs TEMP usage	Daily	V$SESSION + V$TEMPSEG_USAGE

Testing & Validation

Bi-weekly Drill:

1. Load one full quarter (~14,239 rows) into STOCKS_Q1_2025.
2. Run sort-heavy query using our table columns:

```
SELECT symbol, AVG(close) AS avg_close, SUM(volume) AS total_volume

FROM stocks PARTITION (STOCKS_Q1_2025)

GROUP BY symbol

ORDER BY SUM(volume) DESC;
```

3. Confirm:
    - Buffer cache hit > 95% (V$SYSSTAT)
    - No temp disk spill (V$PGASTAT, V$TEMPSEG_USAGE)
    - PGA < 500MB (EM Top Sessions)
4. Document in Performance Log.

Indexing Strategy

Primary Indexes

a. Symbol Index (STOCKS_SYMBOL_IDX)

- Type: B-tree index on symbol column
- Tablespace: STOCKS_INDEX
- Expected Use: `SELECT * FROM stocks WHERE symbol = 'AAPL';`

b. Date Index (STOCKS_DATE_IDX)
- Type: B-tree index on date column
- Tablespace: STOCKS_INDEX
- Expected Use:
```
SELECT * FROM stocks
WHERE trade_date BETWEEN DATE '2025-01-01'
AND DATE '2025-10-30';
```

Composite Index

c. Symbol-Date Composite Index (STOCKS_SYMBOL_DATE_IDX)
- Type: B-tree index on (symbol, date) columns
- Tablespace: STOCKS_INDEX
- Expected Use:
```
SELECT * FROM stocks
WHERE symbol = 'AAPL' AND
trade_date BETWEEN DATE '2025-01-01'
AND DATE '2025-10-30';
```
- Trade off: This will result in more storage and maintenance overhead but it will improve query performance a lot

Query Optimizer

Use of optimizer hints, statistics gathering.
Optimizer statistics are used by the query optimizer to find the best execution plan for an SQL statement.
Database statistics are used for performance monitoring.
System statistics can only be gathered using the DBMS_STATS package.

We will gather system statistics once at database creation and after any hardware change. This gives the optimizer CPU and I/O speed data for accurate cost calculations.

```
EXEC DBMS_STATS.GATHER_SYSTEM_STATS;
```

Daily Monitoring Plan

We will check query performance every day using Enterprise Manager to spot problems early.

1. EM → Performance Page
   - Look at Average Active Sessions graph.
   - Watch for spikes in System I/O or Concurrency.

2. Drill down on any spike → View Top Working SQL.
    - Watch for SQL with high Physical Read or Logical Reads.
3. EM → Top Consumers → Top Sessions
    - Sort by Logical Reads or Physical Read.

Weekly Statistics Gathering

We will run a scheduled job every Sunday at 01:00 using DBMS_SCHEDULER to gather schema statistics and keep the optimizer accurate.

```
BEGIN

  DBMS_STATS.GATHER_SCHEMA_STATS(

       ownname      => '[SCHEMA_NAME]',  -- will refer to our schema name
during Nov 3 class

       cascade      => TRUE             -- Includes indexes, this might be
default in Oracle

  );

END;

/
```

Weekly Testing

We will test optimizer improvement each Sunday.

1. Before stats, record baseline table scans

```
SELECT value FROM v$sysstat WHERE name = 'table scans (short tables)';
```

2. Run a select * query on one of the columns (look into a good one) → Note Physical Read in EM Top SQL.
3. Run stats job.
4. After stats: Re-run query → Physical Read should be lower.
5. Check table scans again and compare to baseline
6. Success Criteria:
    - Physical Read decreases
    - Table scans do not increase → Optimizer is now using index instead of full scan.

INVALID OBJECTS

Invalid PL/SQL: Recompiles on first use → delay + possible failure.

Unusable indexes: Ignored by optimizer → full table scans.

Daily Detection (Automated Job at 03:00)

We will run a scheduled job every day using DBMS_SCHEDULER to detect issues and send alerts to EM.

Job Strategy:

1. Needs to run the following queries.

   Invalid PL/SQL:

   ```
   SELECT owner, object_name, object_type

   FROM dba_objects

   WHERE status = 'INVALID'

     AND owner NOT IN ('SYS','SYSTEM');
   ```

   Unusable indexes:

   ```
   SELECT owner, index_name, table_name

   FROM dba_indexes

   WHERE status = 'UNUSABLE';
   ```

2. If any rows return, write ALERT to EM Alert Log.
3. We check EM every morning → See alert → repair same day. (need rule for who checks for/runs the repair, maybe split days up and keep like a sign-off sheet?)

   Repair queries:

   ```
   ALTER PROCEDURE [schema].[proc_name] COMPILE;

   ALTER PACKAGE [schema].[pkg_name] COMPILE;

   ALTER PACKAGE [schema].[pkg_name] COMPILE BODY;

   ALTER INDEX [schema].[index_name] REBUILD ONLINE;
   ```

If no alerts detected throughout week, run automated weekly repair job Sundays at 02:00

Weekly Testing (Only If Repair Ran)

If we repaired an unusable index, we will re-run a query that uses it and check EM Top SQL to confirm Physical Read is lower after rebuild.

Resource Management

To avoid slow queries and system waits, we will check resource usage every day using Enterprise Manager to spot problems early.

Daily tasks:

1. EM → Performance Page
    - View Average Active Sessions graph.
    - Watching for spikes in CPU, System I/O, or Concurrency.
2. Drill down on any spike → View Top Working SQL and Top Working Sessions.
    - Note sessions with high CPU, Physical Read, or Logical Reads.
3. EM → Top Consumers → Top Sessions
    - Sort by CPU, PGA Memory, Logical Reads, or Physical Read.

Weekly Action (Sunday 02:00)

We will run a scheduled job every Sunday at 02:00 using DBMS_SCHEDULER to kill high-resource sessions.

```
BEGIN

  FOR rec IN (

       SELECT sid, serial#

       FROM v$session

       WHERE status = 'ACTIVE'

       AND (cpu > 100000

       OR physical_reads > 100000

       OR pga_memory > 500000000

       OR hard_parses > 100

       OR sorts > 1000)

  ) LOOP

       EXECUTE IMMEDIATE 'ALTER SYSTEM KILL SESSION ''' || sess.sid || ',' ||
sess.serial# || ''' IMMEDIATE';

   END LOOP;

END;
```

/

Weekly Testing

We will test resource control each Sunday.

1. Run a resource-heavy query, noting CPU, Physical Read, PGA Memory in EM Top Sessions.
2. Run the weekly kill job.
3. After action - Re-check EM Top Sessions → CPU, Physical Read, PGA Memory should drop.

Monitoring and Maintenance Plan

Alert Logs and Trace Files

The Oracle alert log and trace files are critical diagnostic tools for database monitoring and maintenance. The alert log (alert_<SID>.log) records significant database events such as startup, shutdown, structural changes, and critical errors. It is located in the Automatic Diagnostic Repository (ADR) directory, as defined by the DIAGNOSTIC_DEST initialization parameter.

Trace files are also stored within the ADR.

- **Foreground trace files** (e.g., ops_ora_<thread_num>.trc) are generated by user processes that encounter errors or need diagnostic tracing.

- **Background trace files** (e.g., ops_<writer><thread_num>.trc) are created by background processes such as the log writer (LGWR), process monitor (PMON), or database writer (DBWR).

Monitoring and Review Procedures

The alert log will be reviewed weekly to detect and address early signs of database issues. The following types of failures should be investigated immediately if detected:

- **I/O failures** - such as exceeding the file descriptor limit or inaccessible storage channels.
- **Inconsistencies** - for example, when control files are out of sync with data files or redo logs.
- **Physical corruptions** - including block checksum errors or invalid block headers.
- **Logical corruptions** - such as inconsistent data dictionary entries or corrupt index/row pieces.
- **Inaccessible components** - missing or misconfigured data files, incorrect OS-level permissions, or offline tablespaces.

Critical alerts include the loss or corruption of data files associated with the SYSTEM or UNDO tablespaces, as these are essential for database operation and recovery. Any such events must be escalated and resolved immediately.

Setting up Automated Alert Monitoring via EM or a custom script to parse the alert log for ORA-errors and send us email notifications would be a good idea to reduce manual effort.

Retention Policy: We will retain alert logs for 90 days, and then have them deleted.

Repair and Recovery

In the event of a detected failure, RMAN (Recovery Manager) can be used to analyze and resolve the issue.

The following RMAN commands assist with automated diagnostics and recovery:

- ADVISE FAILURE lists detected failures and recommended actions.
- REPAIR FAILURE PREVIEW shows which repairs RMAN would perform.
- REPAIR FAILURE initiates those repairs automatically.

Health Checks and Performance Baselines

Database health checks and performance baselines are essential for identifying abnormal behavior, diagnosing degradation, and ensuring ongoing system reliability.

Performance baselines will vary depending on the execution environment. In this project, potential environments include:

- Personal machines (local testing)
- Okanagan College virtual machines (controlled instructional environment)
- Digital Research Alliance infrastructure (research-grade hardware and networking)

Because performance characteristics (such as CPU throughput, I/O latency, and memory availability) can differ significantly between these platforms, each environment should have its own baseline measurements. These baselines will serve as reference points for evaluating performance trends over time.

Automated Performance Data Collection

The Automatic Workload Repository (AWR) and Automatic Database Diagnostic Monitor (ADDM) will be used to automatically collect and analyze performance data.

The AWR stores a series of snapshots that capture key performance statistics at regular intervals. These snapshots can be grouped into baselines using the DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE procedure, enabling direct comparison of system performance across different periods.

Insights derived from AWR and ADDM reports will guide decisions about routine maintenance tasks, which are managed through Oracle's Automated Maintenance Tasks Infrastructure.

Key Performance Indicators
Key areas to monitor for performance and stability include:

- Memory allocation issues
- I/O device contention
- Application or SQL inefficiencies
- Resource contention between sessions
- Network bottlenecks

Performance statistics will be reviewed at three levels:

- System-wide: V$SYSSTAT, V$SYSTEM_EVENT
- Session-specific: V$SESSTAT, V$SESSION_EVENT
- Service-specific: V$SERVICE_STATS, V$SERVICE_EVENT

This layered monitoring ensures that both global and localized performance problems can be identified and addressed promptly.

Alert Management
Server-generated alerts are another key diagnostic mechanism. These include both threshold-based alerts (metric-driven, such as space usage or response time) and non-threshold alerts (event-driven, such as data block corruption or job failures).

When database metrics deviate from baseline values and exceed predefined thresholds, Oracle automatically issues an alert. During testing, we will configure sample thresholds, trigger controlled test cases, and verify that alerts are raised as expected.

Once an alert is received, we will:

1. Gather additional diagnostic input from AWR, ADDM, and alert logs.
2. Investigate and confirm the root cause.
3. Apply corrective actions or adjustments as necessary.

Active alerts can be queried in the DBA_OUTSTANDING_ALERTS view, while resolved alerts are recorded in DBA_ALERT_HISTORY.

Health Monitor and Automated Diagnostics
Oracle's built-in Health Monitor automatically performs integrity checks and logs findings in the Automatic Diagnostic Repository (ADR). These checks include verification of:

- Control file and data file consistency
- Undo and redo integrity
- Transactional and dictionary corruption detection

Failures and their associated symptoms are recorded as diagnostic findings within the ADR.

The results of these checks will be reviewed weekly as part of the routine maintenance process to identify and resolve emerging issues early.

Capacity Planning

Overview

The stocks database is designed to store time-series financial market data received from modeling or data feed processes. Each record represents a price snapshot for a stock symbol captured approximately every 5 minutes.

Currently, the database holds around 29,381,155 records, imported from historical CSV data from July 1 2023 to October 24 2025. This serves as the initial dataset for testing and development. In production, new data would be inserted continuously throughout the trading day.

Data Growth Estimation

Assuming there are ~7 trading hours per day, 12 samples per hour (every 5 minutes), that brings us to 84 records per stock per day.

Currently, the CSV file to populate the database has information on 501 distinct stocks, so that's 501 tracked stock symbols.

84 * 501 = 42,084 new records per day

There are 252 trading days per year, so

~= 10,605,168 records per year

Observed baseline: after loading 2 years of historical data the database on-disk usage is:

- stocks_data01.dbf (table data): 2.59 GB
- stocks_index01.dbf (indexes): 1.11 GB
- Combined used on disk: 3.70 GB

Empirical annual growth (measured):

- Data: 2.59 GB / 2 years = ~1.30 GB/year
- Indexes: 1.11 GB / 2 years = ~0.56 GB/year
- Combined: ~1.86 GB/year

Scenarios:

- Linear growth (add ~1.86 GB/year): total ≈ 13.0 GB in 5 years.
- Mild exponential (10% CAGR): total ≈ 6.0 GB in 5 years.
- Aggressive exponential (25% CAGR): total ≈ 11.3 GB in 5 years.

Provisioning recommendation: allocate 2.5× the estimated primary total to cover working headroom, backups, and archive log storage. Under this approach, provision ~28–33 GB for

worst-case alignment with aggressive growth over 5 years, or ~15 GB if you expect only mild exponential growth.

Operational controls: enable autoextend with a max limit, implement partitioning by trade_date, enable compression for historical partitions where possible, and set monitoring/alerts at 75% and 90% tablespace usage. Regularly capture DBA_SEGMENTS and datafile sizes and recalibrate projections.

Risk Mitigation

To minimize the risk of data loss, corruption, or unauthorized access, several protective measures are in place:

- **Multiple Backups:** Regular backups are maintained to ensure recoverability in case of hardware failure, corruption, or accidental deletion.
- **Data Duplication:** Critical data is mirrored across separate partitions, reducing the likelihood of total data loss from a single point of failure.
- **Limited User Privileges:** Principle of least privilege is enforced-users are granted only the permissions necessary for their roles, minimizing the risk of accidental or malicious changes.
- **Comprehensive Logging:** All database activities, configuration changes, and access attempts are logged to support auditing and incident analysis.
- **Flashback Recovery:** Flashback features provide the ability to quickly restore data to a previous state, allowing for rapid recovery from logical errors or unintended modifications.

Together, these measures provide a layered defense strategy that safeguards data integrity, availability, and security throughout the project lifecycle.

Collaboration with Stakeholders

Communicating with Albert Wong, Youry Khmelevsky, and Gaetan Hains to ensure proper implementation and meeting of expectations.

Tools and Utilities

Oracle Enterprise Manager

We will use Enterprise Manager to monitor real-time and historical performance diagnostics along with handling alerts, incident management and resource management.

RMAN (Recovery Manager)

We will use Recovery Manager for backups of data files, control files, archived logs, and server parameter files to disk. We will also use RMAN for the recovery of such files.

ASM (Automatic Storage Management)

We will use ASM for a high-performance cluster file system for the database files. This will spread data across disks for automatic load balancing.

AMM (Automatic Memory Management)

We will use AMM to dynamically shift memory between the SGA and PGA based on query patterns to improve developer productivity.

Data Pump / SQL*Loader

We will use Data Pump if we need to move schemas, tablespaces, or the full database.
We will use SQL*Loader to import or export data files to the database (import the data from the csv file stored on Github into the Oracle database).

AWR (Automatic Workload Repository) & ADDM (Automatic Database Diagnostic Monitor)

Used to collect snapshots for performance baselines and analyze performance data.

Citations

[1] K. Cormier et al., "Data Extraction, Transformation, and Loading (ETL) Process Automation and Data Warehouse Implementation for Algorithmic Trading Machine Learning Modelling*," 2025 IEEE International systems Conference (SysCon), Montreal, QC, Canada, 2025, pp. 1-8, doi: 10.1109/SysCon64521.2025.11014812. keywords: {Automation;Prototypes;Transforms;Predictive models;Data warehouses;Data models;Real-time systems;Data mining;Stock markets;Load modeling}

[2] 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