

Cloud Evolving Data



Oracle Performance Simplified

Handbook for DBAs & Architects

Nitin Mare

BLUEROSE PUBLISHERS
India | U.K.

Copyright © **Nitin Mare** 2026

All rights reserved by author. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the author.

Although every precaution has been taken to verify the accuracy of the information contained herein, the publisher assumes no responsibility for any errors or omissions. No liability is assumed for damages that may result from the use of information contained within.

BlueRose Publishers takes no responsibility for any damages, losses, or liabilities that may arise from the use or misuse of the information, products, or services provided in this publication.



BlueRoseONE[®]
S t o r i e s • M a t t e r
New Delhi • London

For permissions requests or inquiries regarding this publication,
please contact:

BLUEROSE PUBLISHERS
www.BlueRoseONE.com
info@bluerosepublishers.com
+91 8882 898 898
+4407342408967

ISBN: 978-93-7825-172-6

First Edition: March 2026

Contents at a Glance

1. DB & Application Design	1
2. SQL Tuning	9
3. Memory Usage	35
4. Oracle Data Access	66
5. Storage	85
6. Network	94
7. ASM, AWR & ADDM Reports	111



Contents

02

SQL Tuning

2.1 SQL Tuning	10
2.2 Index, Index Types	13
2.3 Explain Plan	18
2.4 Optimizer Purpose & CBO	22
2.5 High Load SQL	29
2.6 SQL Access Adv	32

04

Oracle Data Access

4.1 Oracle Data Access	67
4.2 Tools: Oracle Data Access	71
4.3 Chained Rows	73
4.4 Internet of Things (IoT)	76
4.5 Data Manipulation	80

06

Network

6.1 Mview and Benefits	96
6.2 Tune Mview SQL	99
6.3 Mview methods & Mlog	102
6.4 Mview Refresh & Mgmt	104
6.5 Stored Procedure	105

01

Database & Application Design

1.1 Business Logic	03
1.2 Data Design	04
1.3 Distribution of CPU	05
1.4 User access Path	06
1.6 Restrict Dynamic SQL	07
1.6 Minimize SQL Required	08

03

Memory Usage

3.1 SGA-System Global Area	37
3.2 PGA-Program Global Area	42
3.3 Database Buffer Cache	47
3.4 Redo Log Buffer	53
3.5 Large Pool	56
3.6 Shared Pool	60

05

Storage

5.1 Optimize Storage	87
5.2 Storage Performance	88
5.3 Storage Type	90
5.4 Logical & Physical	93

07

ASM, AWR & ADDM Reports

7.1 AWR	113
7.2 ASH	124
7.3 ADDM	132



Oracle Reports

07

ASH, AWR, ADDM

Network

06

Mview Benefits, Tune Mview,
Refresh Methods, Mlog,
Stored Proc

Storage

05

Optimize Storage Performance,
Types, Logical & Physical Storage

Data Access

04

Chained Rows, Internet of
Things

Memory Usage

03

SGA, PGA, Large & Shared
Pool, DB/ Redo Buffer

SQL Tuning

02

Index, Explain Plan, Optimizer,
High Loud SQL, SQL Access Adviser

DB & App Design

01

Business Logic, Data Design, Distribution,
User Access, Restruct Dynamic SQL Required

Database Performance Tuning

7-Step Framework



About The Author



Nitin Mare has spent 19 years in the trenches of database administration and architecture. He is an Oracle Certified Professional(OCP-DBA) & Oracle Certified Expert(OCE-RAC) has been at the forefront of Oracle database performance tuning and architecture. As an Oracle DBA Architect and Exadata Expert, he has been the person called in when performance goes critical. His career has been defined by solving the toughest database slowdowns, diagnosing RAC contention, optimizing Exadata environments for maximum throughput, and architecting cloud solutions that outperform on-premise benchmarks.

As a Multi-Cloud Database Specialist, He has helped enterprises break free from single-vendor lock-in, optimizing performance whether the workload sits on OCI, AWS, or Azure. In this book, He shares the practical scripts, methodologies, and architectural patterns he uses in the field to turn slow databases into high-speed engines.

Nitin Mare





About The Design & Reviewer



Arpita Bhagure

Arpita Bhagure, Designer- expert in design and gave artistic vision to the book layout, not just the practical requirements of book design, but the deeper goal: to create a cover that would speak to readers before they read a single word.



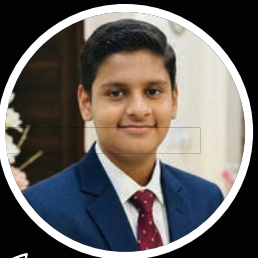
Harshada Mali

Harshada Mali, Reviewer- Data Scientist with expertise in multi-Database across multiple Cloud applications. This book benefited immensely from the rigorous scrutiny & her practical, battle-tested perspective for design reviews.



Shweta Bhuwan

Shweta Singh, Reviewer-Data Scientist with Database Development knowledge, applied her singular expertise. Her design review sharpened the book's focus on measurable outcomes and actionable advice.



Aryaman Mare

Aryaman Mare, Designer-Treats each pictures as a story chapter and pictographs as key plot points. Expertly uses color and composition to make the book a visual artifact. His approach combines to create covers that attract readers & serve the text.





Acknowledgement

Writing a book is rarely a solo journey. While the drafting happened in private, the content was shaped by a community of support.

First, I must thank my family. To my wife, Manisha, thank you for your unwavering patience during the countless late nights and weekends spent in front of the screen. To my children, Aniket & Aryaman, thank you for understanding when Daddy had to "tune one more query."

I am deeply grateful to my technical reviewers, Harshada Shweta & Arpita. Your forensic attention to detail caught my errors and strengthened my logic. You made this work technically robust, and I am forever in your debt.

To my colleagues and friends in the industry: our late-night debates and shared "war stories" from the trenches are the soul of this book. You have influenced my thinking more than you know.

"Oracle Performance Simplified" the methodology primarily focuses on a 7-step framework designed to drill down into database performance bottlenecks for efficient resolution.

Lastly, to the reader: thank you for your faith in this work. I hope these pages help you solve your toughest challenges and spark new ideas.



Foreword

In my 19 years working with the Oracle kernel, the landscape has shifted completely, moving from spinning disks to the boundless horizon of the cloud. Every new feature adds power, but it also adds complexity.

Until now, deep performance tuning was often guarded by "gurus" and passed down through cryptic scripts. Nitin Mare changes that. He provides a comprehensive guide that connects core principles to modern, hybrid realities.

Nitin accomplishes a rare feat here: he turns two decades of architectural expertise into a practical roadmap. He treats the database as an integrated part of a complex ecosystem, showing you exactly how to make the optimizer work for you in Exadata or cloud environments.

Nitin's background as a DBA Architect and multi-cloud specialist is exactly what the industry needs. This book is the bridge between theory and practice. If you are serious about performance, this book is for you.



Introduction

Welcome. By opening this book, you've recognized a fundamental truth: in the world of Oracle, performance isn't just a "feature" it is the foundation of reliability and trust.

For 19 years, my career as a DBA and architect has been defined by one question: "Why is it slow?" The answer is never simple. It has evolved from tracing issues in old legacy systems to managing massive Exadata racks and balancing costs across multiple clouds. While the tools change, the core discipline remains the same.

This book is built on one core belief: effective tuning is not just a collection of scripts; it is a proactive architectural discipline. It is a mindset. You must see the database not as a "black box," but as a complex system where one change can impact everything from storage latency to your monthly cloud bill.

Together, we will build that mindset. We will start with a solid mental model of the Oracle engine. Then, we will layer in the intelligence of Exadata and navigate the new frontier of multi-cloud environments. My goal isn't just to teach you how to "tighten a bolt," but to help you understand the entire engine.

1

Database & Application Design

**"A clever person solves a problem. A
wise person normalizes the database.
But an experienced developer just adds
another column and prays there are no
side effects."**

Taxonomy of Design



- **Oracle Database Design:**
- It is defined as the process of architecting a structured, AI-native information system using the Oracle Database ecosystem.
- It involves translating business requirements into a functional data model that leverages the multitenant architecture and advanced AI capabilities of Oracle AI Database 26ai.
- **Application Design:**
- In the Oracle ecosystem, application design is the process of defining how software interacts with the database to access, process, and present data.

1. Business Logic:

Business logic refers to the rules and procedures that govern a business.

2. Data Design:

DB design is a set of steps taken to implement a database that will manage data for business.

3. Distribution of CPU:

Use CPU threads without generating high load averages.

4. User Access Path:

Different users of the same application should query the DB in a very similar fashion.

5. Restrict Dynamic SQL:

You have the flexibility to create your SQL on the fly based on user input.

6. Minimize SQL Required:

App should minimize the number of times they request the data from the database.

1.1 Business/Application Logic



Business logic refers to the rules and procedures that govern a business, including things like pricing, discount, inventory levels, customer eligibility, etc. **Application logic**, on the other hand, is the code that implements those business rules within a specific application.

The key difference between business logic & application logic is all about the **data inputs** based on your business, while application logic is how the **user interacts** with the application.

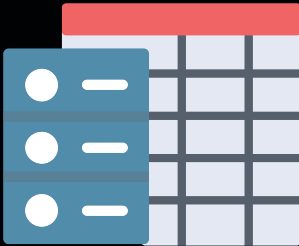
Business logic encodes real-world business rules that determine how users interact with application and how data should be **created, exchanged, & managed**.

Business logic should be written independently of the technology used to implement it. That way, if the technology ever needs to change, you won't have to rewrite your business logic. OR, if your business rules change, then alterations can quickly be made to the business logic.

Business Logic should be completely avoided before business build.

Example: Bank, RMAN Backup Schedule, OS, Network, Storage, Teams.

1.2 Data Design



- **What is Database Design?**
- A [Database Design](#) can be defined as a set of processes or steps taken to implement a database that will manage data for business. It is the organization of data from a database model. A good database design should follow certain rules like making sure all the information is up to date and no redundant information is present as this can cause many mistakes. Another rule will be to make sure inquiries can be answered accurately; the reliability of this data highly depends on the table structure of the design. These rules need to be followed as one mistake alone can be detrimental to a business's operation and function.

Methods to Design a Database:

Most designs use these two method when database designing is concerned.

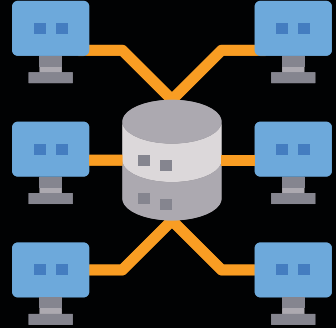
- [Entity relationship modeling](#): In this method, things that are interrelated in some interest will be dedicated to some domain. In other words, data are classified into entities with each entity having data that are related to each other in a specific manner which will be specified. This is done to signify real-life objects.
- [Normalization](#): A relational database is structured into tables and columns and are linked together based on their relationship. This is done so to reduce the redundancy of information and to increase the integrity of the data. (1NF, 2NF, 3NF).

1.3 Distribution of CPU

If the I/O burden of the system are reduced the CPU burden may increase you have several options for managing the CPU resource.

- **Schedule the CPU Load:**

Schedule the long running batch queries or update progress to run at off peak hours maintaining their normal priority level while scheduling the jobs appropriately will minimize locking, undo & CPU conflicts .



- **CPU movement:**

Take advantage of the opportunity to physically shift CPU requirements from one server to another. Where ever possible, isolate the database server from the application CPU requirement.

- **Parallel Query:**

Use parallel Query to distribute the processing requirement of SQL statement among the multiple CPU's. Parallelism can be used by every SQL, DML & DDL commands. The degree to which a transaction is parallelized depends on the defined degree of parallelism for the transaction .



1.4 User Access Path



- **User access path** ensures that users only have access to what they need. It involves multiple stakeholders, including IT, security, and compliance teams, to ensure the process is effective and aligns with Organizations and compliance objective.



- Different users of the same application should query the database in a very similar fashion. Consistent access paths increases the availability of information in SGA. If the queries are identical, parsed version of a query may already exist in the shared pool; reducing the amount of time needed to process the query. Cursor enhancements in the optimizer increases the likelihood of statement reuse within the shared pool but the application needs to be designed with statement reuse in mind.
- **Optimizer generate different access plan**, best plan, cost (Index, FT) Rewrite SQL.

1.5 Restruct Dynamic SQL

You should restruct the use of dynamic SQL;

- **Dynamic SQL:** by definition defined until run time, perform several full table scans of the order table the second time, and inadvertently perform a cartesian join the third time.
- Dynamic SQL is a double edged sword. You have the flexibility to create your SQL on the fly based on user input, but you open yourself upto SQL injection attacks for both of your in-house applicaions and your external web site application.



- **Dynamic code** is one of two methods for coding applications for SQL. The other is called stored procedures. The difference between the two is how the application calls information from the database using certain types of statements and processes.
- There are drawbacks to using dynamic SQL, however. Stored procedures execute a lot faster than dynamic SQL statements, since the application is only looking for commands. Using dynamic SQL statements requires the database application to generate the code then execute it as requested, which means it is processing more before it actually starts looking for results. It is then formatting the results on the fly as well, which further slows the application.
- Other drawbacks are security issues and potential errors. Increased flexibility means holes in the code are easier to find and exploit. Since there are no parameters, there is an increase in flexibility for large databases, and the code is not created until execution, testing for errors and conflicts is more difficult, if not impossible.

1.6 Minimize SQL Required



- **Application should minimize** the number of times they request the data from the database. By using sequences, PL-SQL blocks and the denormalization of tables, also MVIEW in distributed objects to reduce how many times DB queried
- You should minimize the number of times you open and close sessions in the DB. Because of the performance of SQL may be minor impact but session mgmt more
- **Oracle SQL best practices to help improve your SQL queries.**

1. Use ANSI Joins Instead of Oracle Joins
2. Avoid WHERE Clauses with Functions
3. Use CASE Instead of Multiple Unions
4. Minimise the Use of DISTINCT
5. Redesign Data Value Lists to Use Tables
6. UNION ALL instead of UNION
7. Use Table Aliases
8. Only Use HAVING on Aggregate Functions
9. Always Specify Columns in INSERT Statements
10. Avoid Object Names with Spaces



2

SQL Tuning

"Most of the tuning is done at application level, 20% of code take 80% of time. So rewrite 20% code to improve performance"

2.1 SQL Tuning

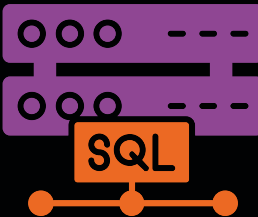
- **Introduction to SQL Tuning:**
- SQL tuning is the attempt to **diagnose & repair** SQL statements that fail to meet a performance standard.
- SQL tuning is the process of **improving the performance** of SQL statements in an Oracle database. The goal of SQL tuning is to make SQL queries run faster, which can help reduce the response time for users & the resources used.
- SQL tuning is essentially the process of optimizing a database for better performance.
- SQL tuning is the iterative process of improving SQL statement performance to meet specific, measurable, and achievable goals. SQL tuning implies fixing problems in deployed applications.



- **Objective of SQL Tuning:**
 - A SQL statement becomes a problem when it fails to perform according to a measurable standard. After you have identified the problem, a typical tuning session has one of the following goals:
- **Reduce user response time:** decreasing the time between when a user issues a statement and receives a response
- **Improve throughput:** using the least amount of resources necessary to process all rows accessed by a statement
- **Steps involved in SQL Tuning:**
 - **Identify High-Load SQL:** Use the ADDM or AWR to identify SQL statements that are poorly performing.
 - **Verify Execution Plans:** Check that the query optimizer's execution plans for high-load SQLs are reasonable.
 - **Implement Corrective Actions:** Generate better execution plans for poorly performing SQL statements.
 - **Some ways to improve SQL Performance include:** Using indexes to reduce the amount of resources used, Optimizing how SQL queries function, Reducing, Balancing, and Parallelizing the workload.



SQL Profile



- **SQL Tuning Tasks:**
- A tuning session depends on many factors, including whether you tune proactively or reactively.
- Proactive SQL tuning, you use SQL Tuning Advisor to determine whether to make SQL statements perform better.
- Reactive SQL tuning, you correct a SQL-related problem that a user has experienced.
- A typical SQL tuning session involves following tasks:
 - Identifying high-load SQL statements.
 - Gathering performance-related data.
 - Determining the causes of the problem.
 - Defining the scope of the problem.
 - Implementing corrective actions for suboptimally performing SQL statements.
 - Preventing SQL performance regressions.

SQL Profile:

1. Creating Tuning Task

```
DECLARE
  l_sql_tune_task_id VARCHAR2(100);
BEGIN
  l_sql_tune_task_id := DBMS_SQLTUNE.create_tuning_task (
    sql_id => 'd342t9u6r4jub', scope => DBMS_SQLTUNE.scope_comprehensive,
    task_name => 'd342t9u6r4jub_tuning_task',
    description => 'Tuning task for statement d342t9u6r4jub..');
  DBMS_OUTPUT.put_line('l_sql_tune_task_id: ' || l_sql_tune_task_id);
END;
```

2. Execute tuning task:

```
EXEC DBMS_SQLTUNE.execute_tuning_task(task_name => 'd342t9u6r4jub_tuning_task');
SELECT task_name, status FROM dba_advisor_log WHERE owner = 'SYS'
select owner, status, TASK_NAME from dba_advisor_log where TASK_NAME like '%tuning_task%' and TASK_NAME
like 'd342t9u6r4jub_tuning_task'
```

3. Display Recommendations

```
SET LONG 10000000 SET PAGESIZE 1000000 SET LINESIZE 20000
SELECT DBMS_SQLTUNE.report_tuning_task('d342t9u6r4jub_tuning_task') AS recommendations FROM dual;
execute dbms_sqltune.accept_sql_profile(task_name => '56s1yjh2s3dw6_tuning_task', task_owner => 'SYS',
replace => TRUE);
```

SQL Tuning Framework

1. Index

An index is a database structure that provides quick lookup of data in a column or columns of a table.

2. Explain Plan

An explain plan is a statement that shows the sequence of operations that an Oracle database performs to run a query.

3. Optimizer

The Optimizer is one of the most fascinating component of the Oracle Database, since it is essential to the processing of every SQL statement.

4. High Load SQL

High-load SQL are poorly-performing, resource-intensive SQL statements that impact the performance of an Oracle database.

5. SQL Access Advisor

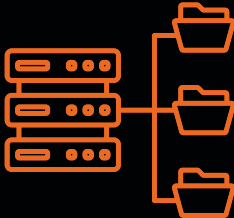
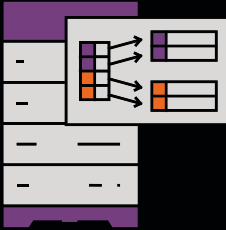
SQL Access Advisor is a diagnostic tool from Oracle that helps identify and fix performance issues with SQL statements.

Oracle SQL Tuning

??



2.2 Taxonomy of Index



- **Index types:**
 - Choose the appropriate index type based on query patterns, including:
 - **B-Tree Index:** The B-Tree index would be placed on columns that were frequently used in the predicate of a query and expect some small fraction of the data from the table to be returned. It is efficient for range queries and equality searches. High cardinality.
 - **Bitmap Indexes:** Bitmap indexes that store pointers to many rows with a single index key entry. There will be a very small number of index entries, each of which points to many rows. Suitable for large datasets with distinct values, efficient for filtering on multiple columns. Low cardinality.
 - **Function-Based Index:** A function-based index calculates the result of a function that involves one or more columns and stores that result in the index.
 - **Unique Index** is a database constraint that ensures that no two rows in a table have the same value for a given column or columns. Ensures no duplicate values in a column
-
- **Primary Key** is a column or group of columns in a relational database table that uniquely identifies each record in the table:
 - **Foreign Key** is a column or set of columns in a table that references the primary key of another table, which links the two tables together.
 - **Concatenated Index**, also known as a multi-column, composite, or combined index, is an index that spans multiple columns in a database table:
 - **Index Organized Tables (IOT)** have their primary key data and non-key column data stored within the same B*Tree structure. Effectively, the data is stored within the primary key index.
 - **Virtual Index** is a "fake" index whose definition exists in the data dictionary but has no associated index segment.

What is Index & How to Tune?



- **What is Index?** An index in Oracle is a structure that provides a shortcut to a table's rows, allowing for faster access to data.
- **Index cardinality:** Consider the number of distinct values in a column when creating an index, as a high cardinality can lead to better performance.
- **Avoiding over-indexing:** Creating too many indexes can negatively impact insert/update performance due to additional overhead, so carefully evaluate the need for each index.
- **How to tune indexes in Oracle:**
 - **Analyze query plans:**
 - Use the EXPLAIN PLAN command to analyze query execution plans and identify potential bottlenecks where indexes could improve performance.
 - **Gather Statistics:**
 - Regularly gather statistics on tables to ensure the optimizer has accurate information about data distribution for optimal index usage.
 - **Monitor Performance:**
 - Use tools like Oracle's Automatic Workload Repository (AWR) to identify poorly performing queries and analyze index usage.
 - **Re-evaluate Indexes:**
 - Regularly review indexes and drop or modify them as needed based on changing query patterns.
- **Important considerations:**
 - **Index Maintenance:**
 - Periodically rebuild or reorganize indexes to maintain optimal performance, especially after large data insertions or deletions.
 - **Index-related parameters:**
 - Adjust relevant database parameters like PCTFREE and PCTUSED to manage index space allocation effectively.



Scripts for monitoring, identifying & managing indexes

1. Identify Unused Indexes:

- Indexes consume space and slow down INSERT/UPDATE operations. Before dropping them, you must monitor them.
- Step A: Enable Monitoring
- ALTER INDEX schema_name.index_name MONITORING USAGE;

2. Step B: Check Usage:

- SELECT index_name, monitoring, used, start_monitoring, end_monitoring
- FROM v\$object_usage
- WHERE index_name = 'YOUR_INDEX_NAME';

3. Check Index Fragmentation (B-Tree Quality):

- In Oracle 19c, we rarely "rebuild" unless the index is truly sparse or the height is too high. This script helps you decide if a rebuild is necessary.
- ANALYZE INDEX schema_name.index_name VALIDATE STRUCTURE;
- SELECT name,
- height,
- lf_rows,
- del_lf_rows,
- (del_lf_rows/lf_rows)*100 AS ratio_deleted
- FROM index_stats;
- Tip: If ratio_deleted is > 20% or height > 3, consider an ALTER INDEX ... REBUILD ONLINE;.





4. Rebuild fragmented indexes:

- ALTER INDEX index_name REBUILD ONLINE;

5. Enable index monitoring:

- ALTER INDEX index_name MONITORING USAGE;

6. Check for high contention indexes:

- SELECT index_name, blevel, leaf_blocks
- FROM user_indexes
- WHERE leaf_blocks > 100000;

7. Identify indexes on frequently updated columns:

- SELECT index_name, table_name
- FROM user_ind_columns ic
- JOIN user_tab_columns tc ON ic.table_name = tc.table_name
- AND ic.column_name = tc.column_name
- WHERE tc.num_distinct < 10; -- Low cardinality

8. Find Duplicate Indexes:

- SELECT table_name,
- LISTAGG(index_name, ', ') WITHIN GROUP (ORDER BY index_name)
- indexes, column_list, COUNT(*) FROM (SELECT
- i.table_name, i.index_name,
- LISTAGG(c.column_name, ', ') WITHIN GROUP (ORDER BY
- c.column_position) column_list
- FROM user_indexes i
- JOIN user_ind_columns c ON i.index_name = c.index_name
- WHERE i.index_type IN ('NORMAL', 'FUNCTION-BASED NORMAL') GROUP BY
- i.table_name, i.index_name
-) GROUP BY table_name, column_list
- HAVING COUNT(*) > 1;





9. Check Index Fragmentation

- `SELECT index_name, blevel, leaf_blocks, distinct_keys,`
- `num_rows,ROUND((num_rows/NULLIF(distinct_keys,0)),2)`
`avg_rows_per_key,ROUND((clustering_factor/num_rows)*100,2)`
`clustering_pct FROM user_indexes`
- `WHERE table_name = 'YOUR_TABLE'ORDER BY blevel DESC;`
- High BLEVEL (>3) or clustering_pct (>20%) indicates fragmentation

10. Identify Missing Indexes (via V\$SQL_PLAN)

- `SELECT sql_text, object_owner, object_name`
- `FROM v$sql_plan`
- `WHERE operation = 'TABLE ACCESS'AND options = 'FULL'AND`
- `object_owner = USERAND executions > 100;`

11. Index Usage Statistics

- `SELECT i.index_name, i.table_name, t.rows_processed,`
- `t.executions,ROUND(t.rows_processed/NULLIF(t.executions,0),2)`
`rows_per_exec FROM v$sql_plan p`
- `JOIN v$sqlstats t ON p.sql_id = t.sql_id`
- `JOIN user_indexes i ON p.object_name = i.index_name`
- `WHERE p.operation LIKE '%INDEX%'AND t.executions > 1000ORDER`
`BY rows_per_exec DESC;`

12. Invalid Indexes

- `SELECT index_name, statusFROM user_indexes`
- `WHERE status NOT IN ('VALID', 'N/A');`

13. Large Index Size Check

- `SELECT`
- `segment_name AS index_name,ROUND(bytes/1024/1024,2)`
`size_mb,`
- `tablespace_name FROM user_segments`
- `WHERE segment_type = 'INDEX'AND bytes > 100*1024*1024 --`
`>100MBORDER BY bytes DESC;`



2.3 Explain Plan

- **Explain Plan:**
- An execution plan is a **step-by-step** explanation of how a database engine will execute the query. It outlines the sequence of operations, the indexes that will be used, and the methods for joining tables.
- The primary purpose of **an explain plan** is to give you insight into the query's performance. By analyzing the explain plan, you can see where the database might run efficiently or where it might encounter performance bottlenecks.
- **Taxonomy of Explain Plan:**
- **Execution Plan (Access Path):** The path followed by the database to execute a query.
- **Cost:** An estimate of the resources (like CPU and I/O) needed to perform a query.
- **Operation:** The specific database action, such as reading from a table or performing an index scan.
- **Rows:** The estimated number of rows that each operation will handle.
- **Filter:** Conditions that rows must meet to proceed to the next step in the query processing.
- **Join Method:** The type of join used, such as nested loop or hash join.
- **Bytes:** Provides an estimate of the amount of data that will be processed by each operation.



Explain Plan & Taxonomy

- **Running EXPLAIN PLAN:**

- EXPLAIN PLAN FOR
- SELECT last_name FROM employees;
-
- EXPLAIN PLAN
- SET STATEMENT_ID = 'st1' FOR
- SELECT last_name FROM employees;

- EXPLAIN PLAN
- INTO my_plan_table
- FOR
- SELECT last_name FROM employees;

- EXPLAIN PLAN
- SET STATEMENT_ID = 'st1'
- INTO my_plan_table
- FOR
- SELECT last_name FROM employees;

- **Displaying PLAN_TABLE Output:**

- SELECT PLAN_TABLE_OUTPUT FROM
TABLE(DBMS_XPLAN.DISPLAY());
- SELEC vT PLAN_TABLE_OUTPUT FROM
TABLE(DBMS_XPLAN.DISPLAY('MY_PLAN_TABLE',
'st1','TYPICAL'));
- SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);



Explain Plan: Purpose

- **Purpose:**
- Use the **EXPLAIN PLAN statement to determine** the execution plan Oracle Database follows to execute a specified SQL statement. This statement inserts a row describing each step of the execution plan into a specified table.
- This statement also determines **the cost of executing the statement**. If any domain indexes are defined on the table, then user-defined CPU and I/O costs will also be inserted.
- **The definition of a sample output table** PLAN_TABLE is available in a SQL script on your distribution media. Your output table must have the same column names and data types as this table. The common name of this script is UTLXPLAN.SQL.



Explain Plan: Views

- Oracle Database provides information on cached cursors through several dynamic performance views:
- For information on the work areas used by SQL cursors, query `V$SQL_WORKAREA`.
- For information on the execution plan for a cached cursor, query `V$SQL_PLAN`.
- For execution statistics at each step or operation of an execution plan of cached cursors (for example, number of produced rows, number of blocks read), query `V$SQL_PLAN_STATISTICS`.
- For a selective precomputed join of the preceding three views, query `V$SQL_PLAN_STATISTICS_ALL`.
- Execution statistics at each step or operation of an execution plan of cached cursors are displayed in `V$SQL_PLAN_MONITOR` if the statement execution is monitored. You can force monitoring using the `MONITOR` hint.
- **Difference between Explain Plan and Execution Plan:**
- To avoid any possible confusion, let us clearly define `EXPLAIN PLAN` vs `EXECUTION PLAN` in Oracle. Both are just sets of instructions for SQL statement that tell the database how to access and combine data to produce the required output.
- As mentioned earlier, `EXPLAIN PLAN` is basically what the Oracle optimizer is planning to do. However, the plan does not always work as expected. Sometimes you have to change and modify it on the fly so that it fits a particular situation. Here is when the `EXECUTION PLAN` comes in: it is an actual set of steps you end up taking. To sum up, `EXPLAIN PLAN` predicts how Oracle will process your query, and `EXECUTION PLAN` describes the steps it actually took.



2.4 Optimizer Purpose & CBO

- **Optimizer:**

- The Optimizer is one of the most fascinating components of the Oracle Database, since it is essential to the processing of every SQL statement. The Optimizer determines the most efficient execution plan for each SQL statement based on the structure of the query, the available statistical information about the underlying objects, and all the relevant optimizer and execution features.

- **Purpose of the Optimizer:**

- The optimizer attempts to generate the most optimal execution plan for a SQL statement.
- The optimizer choose the plan with the Database Buffer Cache among all considered candidate plans. The optimizer uses available statistics to calculate cost. For a specific query in a given environment, the cost computation accounts for factors of query execution such as I/O, CPU, and communication.

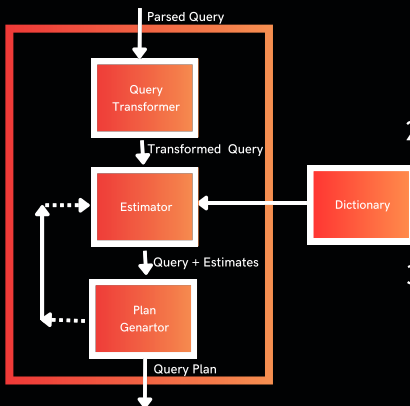


- **Cost-Based Optimization:**

- Query optimization is the overall process of choosing the most efficient means of executing a SQL statement. SQL is a nonprocedural language, so the optimizer is free to merge, reorganize, & process.
- The database optimizes each SQL statement based on statistics collected about the accessed data. The optimizer determines the optimal plan for a SQL statement by examining multiple access methods, such as full table scan or index scans, different join methods such as nested loops and hash joins, different join orders, and possible transformations.

Components of Optimizer

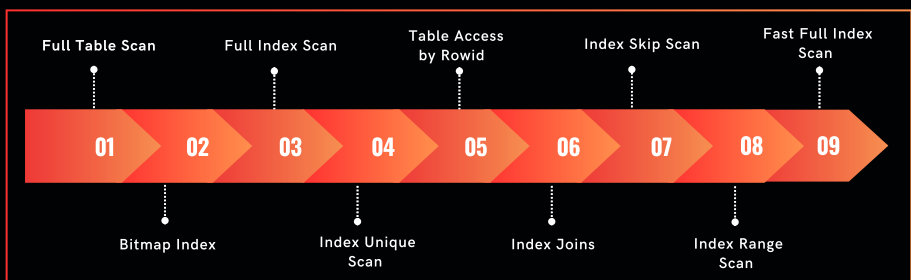
- For a given query and environment, the optimizer assigns a relative numerical cost to each step of a possible plan, and then factors these values together to generate an overall cost estimate for the plan. After calculating the costs of alternative plans, the optimizer chooses the plan with the lowest cost estimate. For this reason, the optimizer is sometimes called the [cost-based optimizer \(CBO\)](#) to contrast it with the legacy rule-based optimizer (RBO). The purpose of the Oracle Optimizer is to determine the most efficient execution plan for your queries. It makes these decisions based on the statistical information it has about your data and by leveraging Oracle database features such as hash joins, parallel query, and partitioning.
- The explain plan is by far the most useful tool at our disposal when it comes to investigating why the Optimizer makes the decisions it makes. By breaking down the explain plan and reviewing the four key elements of: cardinality estimations, access methods, join methods, and join orders; you can determine if the execution plan is the best available plan.



- Components of Optimizer:**
- After a query parse then it is handled by the optimizer. It has three components:
 - Query Transformer:** The query transformer transforms our query into a better one. The Oracle tries to find the best query that returns the same result but in a better performing way.
 - Estimator:** It optimizes our queries by their costs. The cost is a combination of the CPU usage, Memory Usage and the Disc Usage of our query.
 - Plan Generator:** The plan generator calculates various plans for our query by checking different access paths, join methods and join orders.
 - It generates a plan, handles it to the [estimator to estimate its cost](#), and creates another plan, and this goes on like that until all the possible execution plans are created.

How Many Access Paths are available to the Optimizer?

- Oracle supports 9 different Access Methods or Paths
- **Access Paths:** How Oracle gets data
- The optimizer generates the best execution plans for our queries.
- Optimizer verifies all available access paths by factoring in information based on statistics for the tables or indexes accessed by the SQL statement.



Scripts 1 : Explain Plan



- **Scripts for generating, viewing, and comparing plans, as well as managing the optimizer.**

1. **Generating & Viewing Execution Plans:** While EXPLAIN PLAN shows what the optimizer thinks will happen, DISPLAY_CURSOR shows what actually happened. A. The "Predictive" Plan (Estimated) Use this if you want to see a plan without actually running the query. `SQL> EXPLAIN PLAN FOR`

- `SELECT * FROM employees WHERE department_id = 10;`

2. Display the results:

- `SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);`
- B. The "Actual" Plan (from Cache) This retrieves the plan from the library cache for a query that has recently run. `SQL>-- Replace 'sql_id' with the actual ID from v$sql`
- `SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('your_sql_id', NULL, 'ALLSTATS LAST'));`
- C. The "History" Plan (from AWR) Use this if the query is no longer in memory but was captured in an AWR snapshot. `SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_AWR('your_sql_id'));`

3. Compare Plans (Oracle 19c New Feature):

one of the most powerful DBA tools in 19c is COMPARE_PLANS. It allows you

to find why a query's performance regressed by comparing two different execution plans.

```
SQL VARIABLE v_rep  
CLOB;
```

- `BEGIN`
- `:v_rep := DBMS_XPLAN.COMPARE_PLANS(`
- `reference_plan => cursor_cache_object('sql_id_1', 0),`
- `compare_plan_list => plan_object_list(cursor_cache_object('sql_id_2', 0)),`
- `type => 'TEXT', level => 'TYPICAL',`
- `section => 'ALL');`
- `END;`
- `/`
- `PRINT v_rep;`





1. Quick Explain Plan

- EXPLAIN PLAN FOR SELECT /*+ YOUR_HINT */ * FROM your_table WHERE conditions; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

2. Display with Advanced Options

- -- With execution stats if available
- SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(sql_id=>NULL, format=>'ALLSTATS LAST'));

3. Find SQL ID and Get Plan

- -- Get SQL ID from recent queries
- SELECT sql_id, sql_text
- FROM v\$sql WHERE sql_text LIKE '%your_table%' AND ROWNUM <= 5; -- Display plan for specific SQL_ID SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('&sql_id', format=>'TYPICAL'));

4. Compare Optimizer Versions

- ALTER SESSION SET optimizer_features_enable = '11.2.0.4'; EXPLAIN PLAN FOR your_query; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY); ALTER SESSION SET optimizer_features_enable = '19.1.0'; EXPLAIN PLAN FOR your_query; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

5. Check Optimizer Statistics

- SELECT table_name, num_rows,
- last_analyzed, stale_stats
- FROM user_tab_statistics
- WHERE table_name = 'YOUR_TABLE';

6. Identify SQL with Bad Plans

- SELECT sql_id, plan_hash_value,
- executions, buffer_gets/executions avg_buffers,
- elapsed_time/executions/1000 avg_ms
- FROM v\$sql WHERE executions > 100 AND buffer_gets/executions > 10000 ORDER BY avg_buffers DESC;



Scripts 3 : Explain Plan



7. View Optimizer Parameters

- -- Current session parameters
- SELECT name, value FROM v\$sqlparameter
- WHERE name LIKE '%optimizer%'; -- Differences from default SELECT name, value, default_value FROM v\$sqlparameter
- WHERE name LIKE '%optimizer%' AND value != default_value;

8. Force a Specific Plan (Baseline)

- -- Load plan from cursor
- DECLARE
- ret PLS_INTEGER; BEGIN
- ret := DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(
• sql_id => '&sql_id',
• plan_hash_value => &plan_hash); END; -- List baselines SELECT sql_handle,
plan_name, enabled, accepted
- FROM dba_sql_plan_baselines;

9. Quick Hints Guide

- /*+ ALL_ROWS */ -- Optimize for throughput /*+ FIRST_ROWS(n) */ -- Optimize for
n rows response /*+ INDEX(table index) */ -- Force index usage /*+ FULL(table) */ --
Force full table scan /*+ PARALLEL(table, n) */ -- Enable parallel degree n /*+
NO_PARALLEL(table) */ -- Disable parallel /*+ LEADING(table1 table2) */ -- Force
join order /*+ USE_HASH(table) */ -- Force hash join /*+ USE_NL(table) */ -- Force
nested loops

10. Bind Variable Peeking Check

- -- Check if bind variable peeking is enabled SELECT name, value FROM
v\$sqlparameter
- WHERE name = 'optimizer_peek_user_binds'; -- View bind values in cursor SELECT
sql_id, name, value_string, position
- FROM v\$sql_bind_capture WHERE sql_id = '&sql_id';

11. Adaptive Plans Detection

- SELECT sql_id, is_resolved_adaptive_plan, is_reoptimizable
- FROM v\$sql WHERE is_resolved_adaptive_plan = 'YES';

12. Cardinality Mismatch

- -- In 19c+ with DBMS_XPLAN.DISPLAY_CURSOR SELECT * FROM
TABLE(DBMS_XPLAN.DISPLAY_CURSOR(
• sql_id => '&sql_id',
• format => 'ADVANCED +ROWS +ALIAS +NOTE +PREDICATE')) -- Look for
"cardinality mismatch" notes



Scripts 4 : Explain Plan



- **Most Useful Quick Commands:**

1. Get immediate plan with costs:

- `EXPLAIN PLAN FOR your_query;SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());`

2. Get actual execution plan from cache:

- `SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(`
- `sql_id => NULL,`
- `format => 'ALLSTATS LAST ADVANCED'));`

3. Fix bad plan temporarily:

- `ALTER SESSION SET optimizer_features_enable = '12.1.0.2';--`
Run query`ALTER SESSION SET optimizer_features_enable = '19.1.0';`

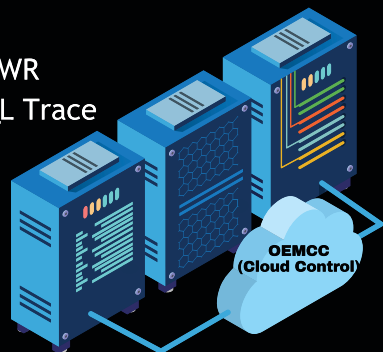
4. Check for statistics issues:

- `EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'YOUR_TABLE');`
- Quick diagnostic flow:
- Get `SQL_ID` from `v$sql`
- Display plan with `DBMS_XPLAN.DISPLAY_CURSOR`
- Check statistics freshness
- Test with hints or optimizer version changes
- Create baseline if good plan found



2.5 High Load SQLs

- **High-load SQL:** These are poorly-performing, resource-intensive SQL statements that impact the performance of an Oracle database.
- **Identifying High-Load SQL Statements:**
 - High-load SQL statements may consume a disproportionate amount of system resources. These SQL statements often greatly affect database performance and must be tuned to optimize their performance and resource consumption. Inefficient SQL can significantly degrade performance.
 - Identifying high-load SQL statements is an important SQL tuning activity that must be performed regularly. ADDM automates this task by proactively identifying potential high-load SQL statements. Additionally, you can use OEMCC (Cloud Control) to identify high-load SQL statements that require further investigation. You can use SQL Tuning Advisor & SQL Access Advisor to tune high load SQLs.
- **Tools/Ways to identify high-load SQL statements: (gathering data SQL statements)**
 - Identification of High-Load SQL Statements- Using ADDM Findings
 - Identifying High-Load SQL Statements- Using Top SQL
 - Automatic Database Diagnostic Monitor- ADDM
 - Automatic SQL tuning
 - Automatic Workload Repository- AWR
 - V\$SQL view, Custom Workload, SQL Trace



High Load SQLs & ADDM



- Let us deep dive to ADDM & TOP SQL:
- Identification of High-Load SQL Statements Using ADDM.
- By default, **ADDM runs proactively once every hour**. It analyzes key statistics gathered by the Automatic Workload Repository (AWR) over the last hour to identify any performance problems, including high-load SQL statements. When the system finds performance problems, it displays them as ADDM findings in the Automatic Database Diagnostic Monitor (ADDM) page.
- ADDM provides recommendations with each ADDM finding. When a high-load SQL statement is identified, ADDM gives recommendations, such as running SQL Tuning Advisor on the SQL statement.



- **Identifying High-Load SQL Statements Using Top SQL:**
- **ADDM automatically identifies high-load SQL** statements that may be causing system-wide performance degradation. Under normal circumstances, manual identification of high-load SQL statements is not necessary.
- The **Top SQL** section of the Top Activity page in Cloud Control enables you to identify **high-load SQL statements for any 5-minute interval**.
- SQL statements that are using the highest percentage of database activity are listed under the Top SQL section, and are displayed in 5-minute intervals.

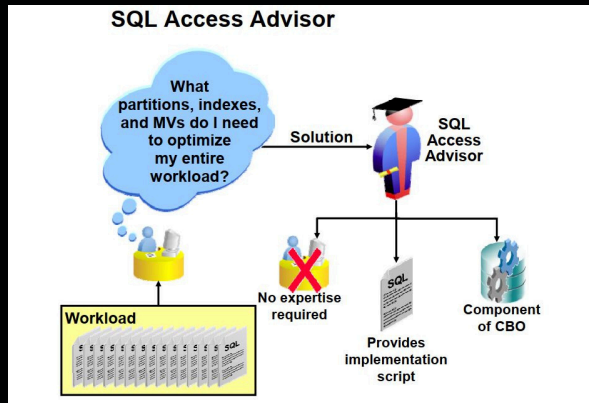
High Load SQLs & ADDM

- **Example:**

- **The AWR and ASH tables will give better information, but this could be a simple first step:**
 - select a.sql_id, a.ratio, a.executions
 - from (select sql_id, buffer_gets / executions ratio, executions
 - from v\$sql where executions > 10 order by 2 desc
 -) a where rownum <= 10
- **To find long running processes taking up your resources look at v\$session_long_ops :**
 - SELECT * FROM v\$session_longops
- **You can then take the sid to find the sql that is running:**
 - SELECT sql_text FROM v\$session s
 - LEFT JOIN v\$sqlarea sa ON s.sql_hash_value=sa.hash_value AND s.sql_address=sa.address
 - WHERE sid=&sid

2.6 SQL Access Advisor

- **SQL Access Advisor:**
- SQL Access Advisor is a **diagnostic tool that helps improve SQL query performance** by analyzing SQL workloads and recommending access structures. It can be used to analyze a single SQL statement or a group of statements.
- **How it works ??**
- Analyzes SQL workloads
- **Recommends indexes, materialized views, materialized view logs, or partitions** to create, drop, or retain
- Generates scripts to implement recommendations



- **Benefits:**
- Doesn't require expert knowledge
- Makes decisions based on **rules in the optimizer**
- Covers all aspects of SQL access in a single advisor
- Provides **user-friendly GUI** wizards
- **How to use??**
- Can accept **input from SQL tuning sets**
- Can take an **actual workload as input** or derive a hypothetical workload from a schema
- Can be used to tune an individual statement using the QUICK_TUNE procedure
- Can be used to **create tasks and templates**

SQL Access Advisor

- **Using SQL Access Advisor:**
- **Optimizing Access Paths with SQL Access Advisor:**
- SQL Access Advisor is **diagnostic software that identifies and helps resolve SQL performance problems** by recommending indexes, materialized views, materialized view logs, or partitions to create, drop, or retain.
- SQL Access Advisor takes an actual workload as input, or the Advisor can derive a hypothetical workload from the schema. It then recommends the access structures for faster execution paths.
- **Purpose of SQL Access Advisor:**
- SQL Access Advisor recommends the proper set of **materialized views, materialized view logs, partitions, and indexes** for a specified workload.
- Materialized views, partitions, and indexes are essential when tuning a database to achieve optimum performance for complex, data-intensive queries. SQL Access Advisor takes an actual workload as input, or derives a hypothetical workload from a schema. The advisor then recommends access structures for faster execution path. The advisor provides the following advantages:
- **SQL Access Advisor advantages:**
- Provides **simple, user-friendly GUI wizards in cloud control** & does not require you to have expert knowledge.
- Bases decision making on rules that actually reside in the cost-based optimizer.
- Is synchronized with the optimizer and Oracle database enhancements.
- Covers all aspects of SQL access methods in a single advisor
- Generates scripts for implementation of recommendations



SQL Tuning Advisor

- **SQL Tuning Advisor:**

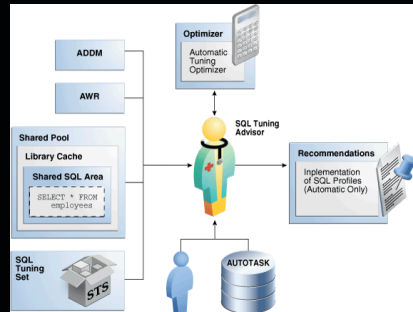
- In Oracle databases, the "SQL Tuning Advisor" is a tool that analyzes SQL statements and provides recommendations to improve their performance, suggesting actions like creating indexes, collecting statistics, rewriting the SQL statement, or creating SQL profiles to optimize query execution. Key points about SQL Tuning Advisor:
- **Function:** It helps identify and resolve issues with sub-optimally performing SQL statements by analyzing execution plans and suggesting tuning options based on the workload and database structure.
- **Access:** This feature is part of the Oracle Database Tuning Pack, requiring an appropriate license to use.
- **Input:** You can provide a single SQL statement or a set of statements (SQL Tuning Set) to the advisor for analysis.
- **Output:** The advisor generates recommendations with explanations and expected performance benefits, including suggestions for creating indexes, gathering statistics, or modifying the SQL statement itself.

- **Benefits:**

- **How to use SQL Tuning Advisor:**

- **Identify target SQL:**

- Find SQL statements with performance issues using tools like the Automatic Workload Characterization (AWC) or by analyzing the Active Session History (ASH).



- **Run the advisor:**

- Submit the SQL statement(s) to the SQL Tuning Advisor through the Oracle database management tools.

- **Review recommendations:**

- Analyze the provided tuning suggestions, including the rationale behind each recommendation and the potential performance gain.

- **Implement changes:**

- Apply the suggested modifications to your database schema or SQL code based on the advisor's recommendations.

3

Memory Usage

**"Set the target, and Oracle moves
memory where it is needed.
Manual tuning is optional;
automatic memory management is
the modern default."**

Oracle Memory Footprint



1. System Global Area

Identifying the basic idea or concept that will be applied to the product or project. This involves research, market analysis, and understanding user needs.

2. Program Global Area

Identifying the basic idea or concept that will be applied to the product or project. This involves research, market analysis, and understanding user needs.

3. Database & Redo Buffer

Identifying the basic idea or concept that will be applied to the product or project. This involves research, market analysis, and understanding user needs.

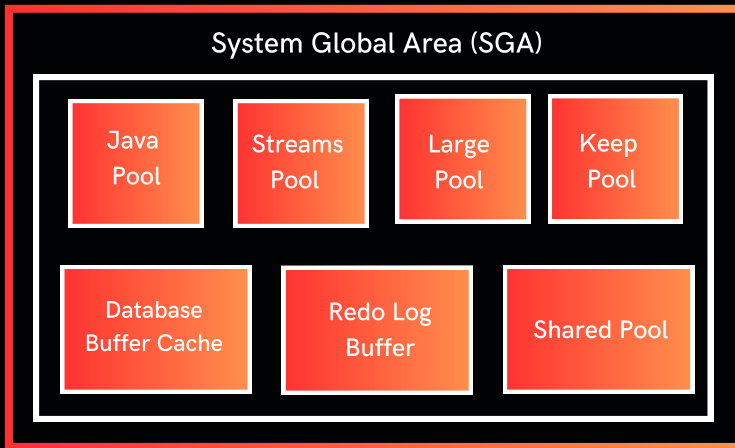
4. Large & Shared Pool

Identifying the basic idea or concept that will be applied to the product or project. This involves research, market analysis, and understanding user needs.

3.1 System Global Area

- **System Global Area (SGA):** It is a shared memory area in Oracle databases that contains data and control information for each database instance. It's also known as the shared global area.
- **How does the SGA work?**
- Oracle automatically allocates memory for the SGA when an instance starts
- The OS reclaims the memory when the instance shuts down. All users connected to the database can read the SGA, and some processes write to it.
- **Check following while configuring the sizes of these SGA components:**
- **SGA Sizing Unit:**
- Memory for the buffer cache, shared pool, large pool, and Java pool is allocated in units of granules. If the SGA size is less than 1 GB, then the granule size is 4MB. If the SGA size is greater than 1 GB, the granule size changes to 16MB. The granule size is calculated and fixed when the database instance starts up. The size does not change during the lifetime.
- **Application Considerations:**
- When configuring memory, size the memory caches appropriately based on the application's needs. Conversely, tuning the application's use of the memory caches can greatly reduce resource requirements. Efficient use of the memory caches also reduces the load on related resources, such as latches, CPU, and the I/O system.

System Global Area



- **What does the SGA contain?**
- **Database buffer cache:** Stores cached data blocks that are read from disk before being queried or modified
- **Redo log buffer:** A component of the SGA, which records all the changes made to database.
- **Shared pool:** Stores information that's shared among users, like SQL statements, user account data, and stored procedures
- **Large Pool:**
- It's used for memory-intensive operations like parallel query execution, backup and restore, and shared server processes.
- **Java Pool:** Used for all session-specific Java code and data in the Java Virtual Machine (JVM). Which holds session information about Java code and data.
- **Streams Pool:**
- Which holds Oracle Streams information like buffered queue messages, capture processes and apply processes.

How to tune SGA ??

- **How to tune the System Global Area (SGA) ?**
 - The **SGA is shared memory** area that **uses RAM**. It contains information needed to run instance.
 - If you are using **automatic memory management** to manage the database memory on your system, then there is no need to tune the SGA. SGA tuning in Oracle is the process of adjusting the size of the SGA to improve database performance.
 - **DBA can manually tune the SGA** to ensure it's the optimal size.
1. **Enable** automatic shared memory management by setting the following parameters:
 - Statistics_Level: Set To Typical Or All
 - Sga_Target: Set To A Non-Zero Value
 2. Set the **SGA_TARGET** to the amount of memory you want to allocate to the SGA. When DB infra is loaded increase SGA to improve performance.
 3. Monitor the SGA size using the command
 - **SQL> select from v\$sga_target_advice order by sga_size**
 4. **Disable** automatic shared memory management:
 - Set the value of the SGA_TARGET parameter dynamically to 0 at instance startup.
 - Here the current auto-tuned sizes will be used for each memory pool.
 5. **Query the \$SGASTAT** view to get information about SGA component.

SGA Views

- Following SGA Views will tell you how much SGA/PGA size you increase/decrease.
- V\$SGA_TARGET_ADVICE, V\$PGA_TARGET_ADVICE
- **Queries for SGA and PGA tuning:**
 - Select Sga_Size, Sga_Size_Factor, Estd_Db_Time_Factor
 - From V\$Sga_Target_Advice Order By Sga_Size Asc;
 - Use a SGA value where the estd_db_time_factor is optimally low
- Select Round (Pga_Target_For_Estimate/1024/1024) Target_Mb,
- Estd_Pga_Cache_Hit_Percentage Cache_Hit_Perc, Estd_Overalloc_Count
- From V\$Pga_Target_Advice;

• What to consider when tuning the SGA?

- Setting the **SGA_TARGET** parameter to too high a value can result in poor performance because too much physical memory is allocated to shared memory
- If the database is configured with Shared Server, you may need to increase the **SHARED_POOL_SIZE** parameter
- Enabling the **LARGE PAGE** feature can improve database performance and stability

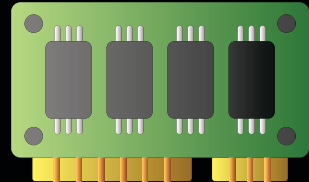
• Steps: How to increase SGA size in Oracle:

1. Check file location of spfile/pfile.
2. Take a backup of pfile before modification.
3. Modify in Pfile for SGA.
4. Bounce the database.
5. If pfile is on the default location then no need to pass the complete path.

SGA Configuration

- Check following while configuring the sizes of these SGA components:
- **SGA Sizing Unit:**
- Memory for the buffer cache, shared pool, large pool, and Java pool is allocated in units of granules. If the SGA size is less than 1 GB, then the granule size is 4MB. If the SGA size is greater than 1 GB, the granule size changes to 16MB. The granule size is calculated and fixed when the database instance starts up. The size does not change during the lifetime.
- **Application Considerations:**
- When configuring memory, size the memory caches appropriately based on the application's needs. Conversely, tuning the application's use of the memory caches can greatly reduce resource requirements. Efficient use of the memory caches also reduces the load on related resources, such as latches, CPU, and the I/O system.

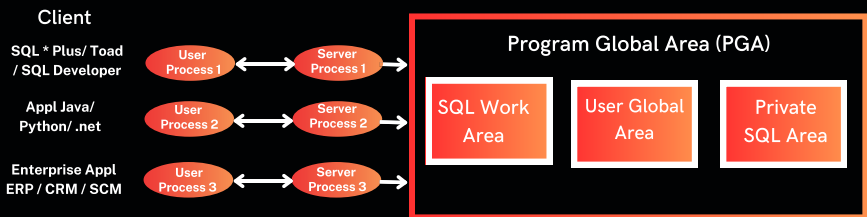
- **A granule is a unit of virtual memory**
- $SGA < 1\text{ GB}$, then granule size = 4MB
- $SGA > 1\text{ GB}$, then granule size = 16MB



- **Operating System Memory Use:**
- **Reduce Paging:**
- Fit the SGA into Main Memory
- Allow Adequate Memory to Individual Users
- **Reduce Paging:**
- Paging occurs when an operating system transfers memory-resident pages to disk solely to load new pages into memory. Many OS page to accommodate large amounts of information that do not fit into real memory.
- To determine whether significant paging is occurring on the host system, use OS utilities to examine the OS. If significant paging is occurring, then the total system memory may not be large enough to hold the memory caches for which memory is allocated. Consider either increasing the total memory on the system, or decreasing the amount of memory allocated.

3.2 Program Global Area

- **Program Global Area (PGA):**
- It is a private memory region that contains the data and control information for a server process. Only a server process can access the PGA. Oracle Database reads and writes information in the PGA on behalf of the server process. An example of such information is the run-time area of a cursor. Each time a cursor is executed, a new run-time area is created for that cursor in the PGA memory region of the server process executing that cursor.



- **Sizing the Program Global Area Using Automatic Memory Management:**
- Automatic PGA memory management simplifies and improves the way PGA memory is allocated. By default, PGA memory management is enabled. In this mode, Oracle Database automatically sizes the PGA by dynamically adjusting the portion of the PGA memory dedicated to work areas, based on 20% of the SGA memory size. The minimum value is 10MB.
- Note: For backward compatibility, automatic PGA memory management can be disabled by setting the value of the PGA_AGGREGATE_TARGET initialization parameter to 0.

Managing PGA & Sizing



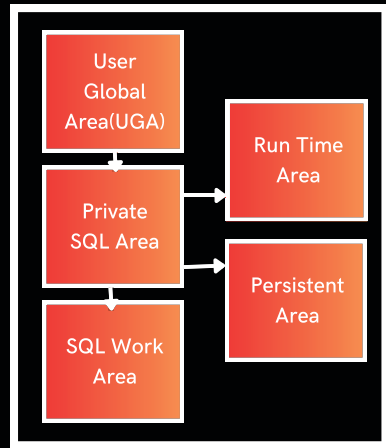
- **Program Global Area(PGA):**
 - RAM is used to allocate the SGA & PGA.
 - The SGA shared by all the sessions.
 - The PGA is private to each sessions.
 - A private memory region that contains data and control info for a single server or background process. Used for session private operations like sorting and private SQL cursors.
 - Allocated by Oracle when a user connects to an Oracle Database.
- **Managing PGA & SGA:**
 - You can use a initialization parameter to set a target maximum size fo the instance PGA.
 - Oracle can automatically manage individual cache sizes by setting a unified memory size parameter.
 - All Oracle memory parameters are set using the Alter System command
- **PGA Sizing (PGA_AGGREGATE_TARGET):**
 - Similar to SGA, PGA is also automatically tuned by the Oracle database if Automatic Memory Management is enabled. The following query can be used to monitor if PGA size is properly sized. The result is shown similar to the query results of v\$memory_target_size and v\$sga_target_size.
 - SQL> select * from v\$pga_target_advice order by pga_target_for_estimate;

Monitoring of PGA

- **Monitoring and tuning:**
- We can use several **dynamic performance views** to monitor the PGA status.
- **V\$PGA_TARGET_ADVICE** - This view provides information about how the statistics cache-hit percentage and over-allocation count are affected if we change the value of PGA_AGGREGATE_TARGET.
- **V\$PGA_TARGET_ADVICE_HISTOGRAM** - This view provides estimated information of a group of the performance statistics
- **V\$PGASTAT** - This view provides information about instance statistics on the PGA memory usage and the automatic PGA memory manager.
- **V\$PROCESS** - This view provides information for each process connected to the instance.
- **V\$PROCESS_MEMORY** - This view provides dynamic PGA memory usage by named component categories for each process.
- **V\$SQL_WORKAREA_HISTOGRAM** - This view shows the number of work areas executed with optimal memory size, one-pass/multi-pass size.
- **V\$SQL_WORKAREA_ACTIVE** - This view can be used to display the work areas that are active (or executing) and to determine whether these active work areas spill to a temporary segment.
- **V\$SQL_WORKAREA** - Contains cumulative work area statistics for each loaded cursor whose execution plan uses one or more work areas.

Components of PGA

- **Components of PGA are as follows:**
- **Private SQL Area:** Holds information about a parsed SQL statement and other session-specific information for processing. A private SQL area is further divided into the following areas



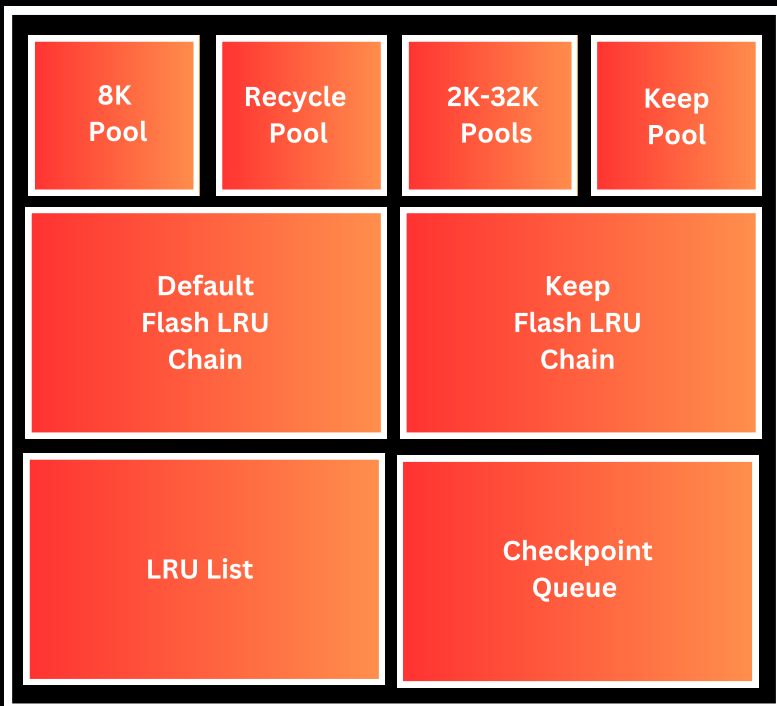
1. **Run time Area:** Contains query execution state information for example, the number of rows retrieved so far in a full table scan.
 2. **Persistent area:** Contains bind variable values.
- **SQL Work Areas:** Used for sorting, hash operations etc.
 - **User Global Area (UGA):** It essentially stores the session state. It contains session information such as logon information, buffers read as a result of direct path reads and other information required by a database Session.
 - **Memory intensive operations which make use of PGA are**
 - - Sort based operators (Order by, group by, rollup, Distinct etc.)
 - - Hash join
 - - Bitmap related operations
 - - Write buffers used by bulk load operations (Direct Path Load)

PGA Mgmt & Configuration

- **Configuring Automatic PGA Memory Management:**
- When running Oracle Database in automatic PGA memory management mode, sizing of work areas for all sessions is automatic, and the *_AREA_SIZE parameters are ignored by all sessions running in this mode. Oracle Database automatically derives the total amount of PGA memory available to active work areas from the PGA_AGGREGATE_TARGET initialization parameter. The amount of PGA memory is set to the value of PGA_AGGREGATE_TARGET minus the amount of PGA memory allocated to other components of the system. Oracle Database then assigns the resulting PGA memory to individual active work areas based on their specific memory requirements.
- With automatic PGA memory management, you set a target size only for the total instance PGA (PGA_AGGREGATE_TARGET). The database then tunes the size of the instance PGA to your target, and dynamically tunes the sizes of individual PGAs.
- **Setting the Initial Value for PGA_AGGREGATE_TARGET:**
- Set the initial value of the PGA_AGGREGATE_TARGET initialization parameter based on the amount of available memory for the Oracle database instance. This value can then be tuned and dynamically modified at the instance level. By default, Oracle Database uses 20% of the SGA size for this value. However, this setting may be too low for a large DSS system.
- **Monitoring Automatic PGA Memory Management:**
- Before starting the tuning process, run a representative workload on the database instance and monitor its performance. PGA statistics collected by Oracle Database enable you to determine if the maximum PGA size is under-configured or over-configured. Monitoring these statistics enables you to assess the performance of automatic PGA memory management and tune the value of the PGA_AGGREGATE_TARGET parameter accordingly.

3.3 Database Buffer Cache

- **Database buffer cache:** It is a memory area in the system global area (SGA) of the database instance. It stores copies of data blocks that are read from data files. All users that are concurrently connected to a database instance share access to the buffer cache. The goals of the buffer cache are to optimize physical I/O .
- **How to configure the database buffer cache:** If you are using automatic shared memory management to configure the SGA, there is no need to manually tune the database buffer cache.



Database Buffer Cache

- **Component of database buffer cache:**
- **The default pool** is the location where blocks are normally cached. The default block size is 8 KB.
- **The recycle pool** is for blocks that are used infrequently. A recycle pool prevents objects from consuming unnecessary space in the cache.
- **The keep pool** is for blocks that are expected to be accessed frequently. The purpose of the keep buffer pool is to retain specified objects in memory to avoid I/O operations. Tables are assigned to the keep pool.
- **Nondefault buffer pools** are for tablespaces that use the nonstandard block sizes of 2 KB, 4 KB, 16 KB, and 32 KB. Each block size has its own pool.
- **The least recently used (LRU) list** contains pointers to dirty and non-dirty buffers. The LRU list has a hot end and a cold end. A cold buffer is a buffer that has not been recently used. A hot buffer is frequently accessed and has been recently used.
- **The checkpoint queue** is a list of dirty buffers in the order they were changed based on the redo block address (RBA) order.
- **Database Smart Flash Cache** (flash cache) lets you use flash devices to increase the effective size of the buffer cache without adding more main memory.

DB Buffer & Queries

- **Hit ratio plays a major role in tuning memory using SQL queries.**
- **Data Dictionary Cache:**
- It is also called row cache. Data Dictionary Cache hit ratio must be >85%, if not increase the shared pool size.
- **Sample queries to calculate the hit ratio:**
- `select (sum(pinhits)/sum(pins))*100 as lchitratio from v$librarycache;`
- `select namespace, pins, pinhits, reloads from v$librarycache order by namespace;`
- To get the package of dbms_Shared_pool please run ``@$ORACLE_HOME/rdbms/admin/dbmspool.sql``
- `select namespace, kept, locks, executions from v$db_object_cache where type like '%PROC%';``
- **Data Dictionary Cache hit ratio:**
- `select (sum(gets-getmisses-fixed))/sun(gets) as "ddchitratio" from v$rowcache;``
- **Redo log Buffer Cache:**
- `Select name, value from `v$sysstat` where name like `redo%`;``
- I hope you find the above-discussed points useful and can now easily tune memory in the database.

DB Buffer & Queries



- **DB Buffer Cache:** The goal of `db_buffer_cache` is to increase the logical reads
- **Soft Parsing:** If we have an execution plan already available in Library Cache, it won't go to the disk, it will create a parse plan (parse code) from the existing execution plan. This is called Soft Parse.
- **Hard Parsing:** To create a parse code, it will check if any execution plan is available, if an execution plan is available then it goes with soft parse. If the execution plan is not available, then it performs a hard parse which means it goes to disk.
- **Shared Pool:** Shared Pool is a combination of Library Cache and Data Dictionary Cache.
- **Library Cache:** The goal of Library Cache is to increase the soft parsing
- **Methods to achieve goal:**
 1. Using bind variables,
 2. By writing stored procedures
 2. Hit ratio (Should be > 85% if not increase the shared pool size),
 4. `DBMS_Shared_pool` (package)

DB Buffer & Views

- **How to configure the Database Buffer Cache:**
- If you are using automatic shared memory management to configure the SGA, there is no need to manually tune the database buffer cache.
- **Using the V\$DB_CACHE_ADVICE View:**
- The V\$DB_CACHE_ADVICE view shows the simulated miss rates for a range of potential buffer cache sizes. This view assists in cache sizing by providing information that predicts the number of physical reads for each potential cache size.
- **To use the V\$DB_CACHE_ADVICE view:**
- Set the value of the DB_CACHE_ADVICE initialization parameter to ON.
- Run a representative workload on the database instance. Allow the workload to stabilize before querying the V\$DB_CACHE_ADVICE view.



DB Buffer & Views

- **Calculating the Buffer Cache Hit Ratio:**
- The buffer cache hit ratio calculates how often a requested block has been found in the buffer cache without requiring disk access. This ratio is computed using data selected from the V\$SYSSTAT performance view. Use the buffer cache hit ratio to verify the physical I/O as predicted by the V\$DB_CACHE_ADVICE
- **Querying the V\$SYSSTAT View:**
- SELECT name, value FROM V\$SYSSTAT
- WHERE name IN ('db block gets from cache', 'consistent gets from cache', 'physical reads cache');
- Using the values from the output of this query, calculate the hit ratio for the buffer cache using the following formula:
- $1 - ((\text{'physical reads cache'}) / (\text{'consistent gets from cache'} + \text{'db block gets from cache'}))$
- **Querying the V\$DB_CACHE_ADVICE View:**
- Query of this view that returns the predicted I/O requirement for the default buffer for various cache sizes.
- SELECT size_for_estimate, buffers_for_estimate, estd_physical_read_factor, estd_physical_reads FROM V\$DB_CACHE_ADVICE WHERE name = 'DEFAULT' AND
- block_size = (SELECT value FROM V\$PARAMETER
- WHERE name = 'db_block_size') AND advice_status = 'ON';

3.4 Redo Log Configuring

- **Redo Log Buffer:**
- Holds redo information (used for instance recovery) until it can be written to the physical redo log files stored on the disk. The Redo log buffer holds all changes made to the database. It records of all the DML and DDL statements fired by the user. It uses the circular mechanism and group of the buffers.
- **The Redo log** buffer plays very important role when DB instances crashes/fails(during recovery phase).
- **Configuring the Redo Log Buffer:**
- Server processes making changes to data blocks in the buffer cache generate redo data into the log buffer. The log writer process (LGWR) begins writing to copy entries from the redo log buffer to the online redo log if any of the following conditions are true:
 - a. **The redo log buffer becomes at least one-third full**
 - b. **LGWR is posted by a server process performing a COMMIT or ROLLBACK**
 - c. **A database writer process (DBWR) posts LGWR to do so.**
- On systems with fast processors and relatively slow disks, the processors might be filling the rest of the redo log buffer in the time it takes the redo log writer to move a portion of the redo log buffer to disk. In this situation, a larger redo log buffer can temporarily mask the effects of slower disks. Alternatively, consider either improving:
 - The checkpointing or archiving process
 - The performance of LGWR by moving all online logs to fast raw devices
- **To improve the performance of the redo log buffer, ensure that you are:**
- Batching commit operations for batch jobs, so that LGWR is able to write redo log entries efficiently
- Using NOLOGGING operations when loading large quantities of data

Redo Log Sizing

- **Sizing the Redo Log Buffer:**
- The default size of the redo log buffer is calculated as follows:
 - $\text{MAX}(0.5\text{M}, (128\text{K} * \text{number of cpus}))$
- On most systems, sizing the log buffer larger than 1 MB does not provide any performance benefit. However, increasing the size of the redo log buffer does not have any negative implications on performance; it merely uses more memory. More importantly, a modestly-sized redo log buffer can significantly enhance throughput on systems that perform many updates. Applications that insert, modify, or delete large volumes of data may require a change to the default size of the redo log buffer.
- To determine if the size of the redo log buffer is too small, monitor the redo log buffer statistics, as described in "[Using Redo Log Buffer Statistics](#)". You can also check if the log buffer space wait event is a significant factor in the wait time for the database instance. If it is not, then the log buffer size is most likely adequately-sized.



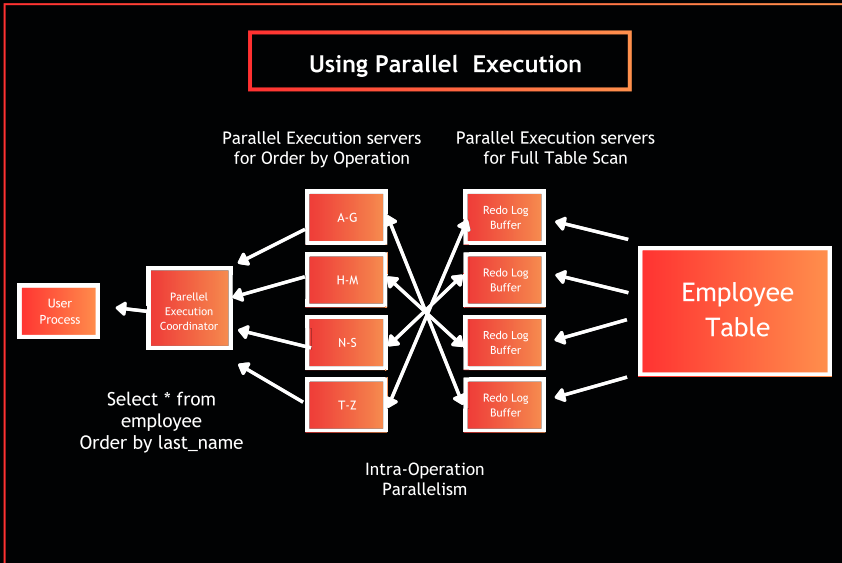
Redo Log Statistics

- **To size the redo log buffer:**
- Set the size of the redo log buffer by setting the value of the LOG_BUFFER initialization parameter to the required size.
- The value of this parameter is expressed in bytes.
- **Using Redo Log Buffer Statistics:**
- The REDO BUFFER ALLOCATION RETRIES statistic reflects the number of times a user process waits for space in the redo log buffer. This statistic can be queried using the V\$SYSSTAT performance view.
- You should monitor the redo buffer allocation retries statistic over a period while the application is running. The value of this statistic should be near zero over an interval. If this value increases consistently, then it means user processes had to wait for space in the redo log buffer to become available. The wait can be caused by the redo log buffer being too small or by checkpointing. In this case, consider one of the following options:
- Increase the size of the redo log buffer, as described in "[Sizing the Redo Log Buffer](#)"
- Improve the checkpointing or archiving process

- **Query of the V\$SYSSTAT view for this statistic:**
- `SELECT name, value`
- `FROM V$SYSSTAT`
- `WHERE name = 'redo buffer allocation retries';`

3.5 Large Pool

- **The large pool:**
- It is an optional memory area within the database instance and system global area (SGA). You can configure the large pool to provide large memory allocations for the following areas:



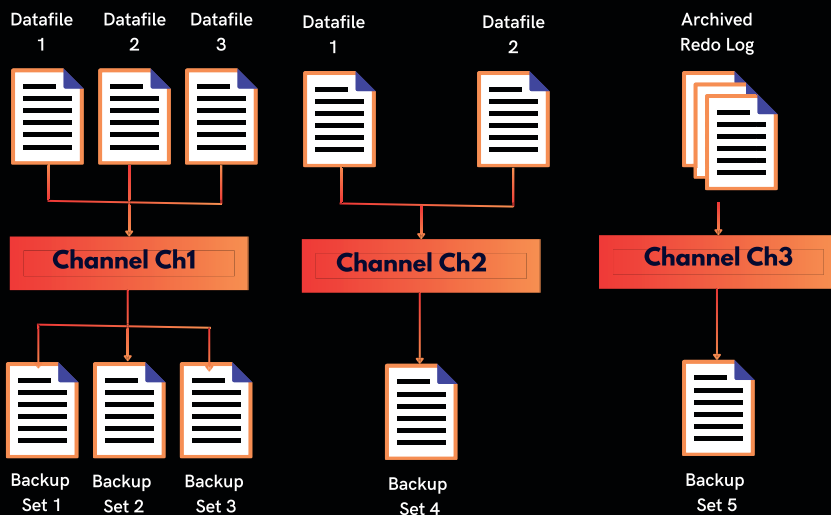
Large Pool & Components

- **Large pool to provide large memory allocations for the following areas:**
- The user global area (UGA) (session memory) for the shared server environment and the Oracle XA interface (which is used where transactions interact with multiple databases). In a dedicated server environment, the UGA is stored in the program global area (PGA).
- The I/O **buffer area includes** I/O server processes, message buffers for parallel query operations, buffers for Recovery Manager (RMAN) I/O workers, and advanced queuing memory table storage.
- The **deferred inserts pool** is used by the fast ingest feature, which enables high-frequency, single-row data inserts into the database for tables that are defined as MEMOPTIMIZE FOR WRITE. The inserts by fast ingest are also called deferred inserts. They are initially buffered in the large pool and later written to disk asynchronously by the Space Management Coordinator process (SMCO) and Wxxx worker background processes after 1MB worth of writes per session per object (or after 60 seconds). Sessions can't read any data that is buffered in this pool, even when committed, until the SMCO background process sweeps. The pool is initialized in the large pool at the first inserted row of a memoptimized table. 2G is allocated from the large pool when there is enough space.
- The **parallel execution message pool (px msg pool)** is used by parallel query execution processes to send messages to each other.

Large Pool & Benefits

- In **Oracle Database 23i**, the "**Large Pool**" is a dedicated memory area within the System Global Area (SGA) that is used to allocate large chunks of memory for specific features like shared server environments, parallel query operations, Recovery Manager (RMAN) processes(backup & restore), and other large data processing tasks, effectively separating these memory needs from the smaller allocations managed by the shared pool, allowing for better performance and scalability when dealing with large data volumes.

RMAN Backup Concepts



Large Pool & Benefits

- **Benefits of Large Pool:**
- **Improved performance** for large-scale operations like **parallel queries**
- Better memory management by separating large allocations from smaller ones
- **Improved Stability:** It prevents large memory operations from causing contention or memory exhaustion in the Shared Pool, leading to more stable and predictable database performance.
- **Memory Allocation for Specific Operations:** The Large Pool is used to allocate memory for operations such as Recovery Manager (RMAN) backup and restore, shared server processes, parallel query execution, and large allocations for session memory.
- **Reduce Fragmentation:** By handling these memory-intensive operations separately, the Large Pool helps to reduce fragmentation in the Shared Pool, ensuring more efficient memory usage.
- **Optimized Performance:** By segregating large memory allocations, the Large Pool optimizes the performance of memory-intensive operations without affecting the general memory usage in the Shared Pool.
- **Reduced Overhead:** The Large Pool does not have the overhead of managing the library and data dictionary cache, making it more efficient for its designated purposes.

3.6 Shared Pool

- **The shared pool:** is a component of the system global area (SGA) within the database instance. It's responsible for caching various types of program data. For example, the shared pool stores parsed SQL, PL/SQL code, system parameters, and data dictionary information.

Shared Pool

Library Cache
Shared SQL & PL-SQL Area

Data Dictionary Cache
Dictionary Data

Server Result Cache
SQL Query ResultCache / PL-SQL Function Result Cache

Reserved Pool
Allocate large contiguous chunks of memory

Enqueues, latches, information lifecycle
management (ILM) bitmap tables, ASH buffers

Shared Pool & Components



- **The shared pool consists of the following subcomponents:**
- **The library cache** is a shared pool memory structure that stores executable SQL and PL/SQL code. This cache contains the shared SQL and PL/SQL areas and control structures, such as locks and library cache handles. When a SQL statement is executed, the database attempts to reuse previously executed code. If a parsed representation of a SQL statement exists in the library cache and can be shared, the database reuses the code.
- **The data dictionary cache** stores information about database objects (that is, dictionary data). This cache is also called the row cache because it holds data as rows instead of buffers. Its having metadata (tables & indexes)
- **The server result cache** is a memory pool within the shared pool that contains the SQL query result cache and PL/SQL function result cache, which share the same infrastructure. The SQL query result cache stores the results of queries and query fragments. Most applications benefit from this performance improvement. The PL/SQL function result cache stores function result sets.
- **The reserved pool** is a memory area in the shared pool that Oracle Database can use to allocate large contiguous chunks of memory. The database allocates memory from the shared pool in chunks. Chunking allows large objects (over 5 KB) to be loaded into the cache without requiring a single contiguous area.
- Other components include **enqueues, latches, information lifecycle management (ILM) bitmap tables, active session history (ASH) buffers**, and other minor memory structures. Enqueues are shared memory structures (locks) that serialize access to database resources. They can be associated with processes or sessions. Examples are control file transaction, instance recovery, media recovery, job queue, and row cache. Latches are used as a low-level serialization control mechanism to protect shared data structures in the SGA from simultaneous access. Examples are cache buffers chains and redo allocation.

Scripts 1: Shared Pool



- **Immediate Action Scripts:**

1. Clear shared pool (emergency only):

- ALTER SYSTEM FLUSH SHARED_POOL;

2. Check and flush aged cursors:-- Cursors not used in last hour

- SELECT COUNT(*) old_cursors
- FROM v\$sqlWHERE last_active_time < SYSDATE - 1/24;-- Optional flushALTER SYSTEM FLUSH SHARED_POOL;

3. Find and fix literal SQL: Top literal SQL patterns

- SELECT SUBSTR(sql_text,1,50) sql_prefix,COUNT(*) versions
- FROM v\$sqlGROUP BY SUBSTR(sql_text,1,50)HAVING COUNT(*) > 100 ORDER BY COUNT(*) DESC;

4. Increase shared pool (if needed): Check current size

- SELECT component, current_size/1024/1024 current_mb
- FROM v\$sga_dynamic_components
- WHERE component LIKE '%shared pool%';-- Increase by 500MBALTER SYSTEM SET shared_pool_size = 2000M SCOPE=memory;

5. Monitor real-time allocations:

- SELECT name, heap_name,
- bytes/1024/1024 size_mb
- FROM v\$sgastat
- WHERE pool = 'shared pool'AND bytes > 10*1024*1024 -->10MBORDER BY bytes DESC;

- **Troubleshooting Steps:**

- Low hit ratio? → Check for literal SQL, increase pool size
- High reloads? → Pin large packages, increase reserved pool
- ORA-4031 errors? → Flush pool, identify memory hogs
- Poor performance? → Check for non-sharable SQL, bind variables



Scripts 2: Shared Pool



1. Shared Pool Size Check:

- -- Current shared pool usage
- `SELECT pool, ROUND(bytes/1024/1024,2) size_mb, ROUND(bytes_free/1024/1024,2) free_mb`
- `FROM v$sgstat`
- `WHERE pool = 'shared pool';`

2. Top Cursors by Memory

- `SELECT`
- `sql_id, executions,`
- `sharable_mem, persistent_mem,`
- `runtime_mem, ROUND(sharable_mem/1024,2) kb`
- `FROM v$sql WHERE sharable_mem > 1024*1024 -- >1MB ORDER BY sharable_mem`
`DESC FETCH FIRST 10 ROWS ONLY;`

3. Library Cache Load

- `SELECT namespace,`
- `gets, gethits, ROUND(gethitratio*100,2) hit_ratio,`
- `pins, pinhits, ROUND(pinhitratio*100,2) pin_hit_ratio,`
- `reloads, invalidations`
- `FROM v$librarycache`
- `WHERE namespace IN ('SQL AREA', 'TABLE/PROCEDURE', 'BODY', 'TRIGGER');`
- Hit ratio should be >95%

4. Shared Pool Fragmentation

- `SELECT name, free_count,`
- `avg_free_size, max_free_size,`
- `used_count, avg_used_size,`
- `max_used_size`
- `FROM v$shared_pool_reserved`
- `WHERE free_count > 0;`

5. Shared Pool Advice

- `SELECT shared_pool_size_for_estimate,`
- `shared_pool_size_factor,`
- `estd_lc_time_saved,`
- `estd_lc_memory_object_hits`
- `FROM v$shared_pool_advice;`





5. Find Large SQL/PLSQL Objects

- -- Large objects in shared pool
- SELECT owner, name,type, sharable_mem, executions, loads
- FROM v\$db_object_cache
- WHERE sharable_mem > 1024*1024 -- >1MBORDER BY sharable_mem DESC;

6. Identify Non-Sharable SQL:

- -- SQL with multiple versions (literal issues)
- SELECT sql_text,COUNT(DISTINCT sql_id) versions,SUM(executions) total_execs
- FROM v\$sqlWHERE parsing_schema_name = USERGROUP BY sql_text
- HAVING COUNT(DISTINCT sql_id) > 5ORDER BY versions DESC;

7. Flush Specific Cursors:

- -- Identify cursor to flush
- SELECT address, hash_value, sql_id
- FROM v\$sqlWHERE sql_text LIKE '%your_query%';-- Flush single cursorDECLARE
- c INTEGER;BEGIN
- c := DBMS_SHARED_POOL.PURGE('&address,&hash_value','C');END;/

8. Shared Pool Advice

- SELECT shared_pool_size_for_estimate,
- shared_pool_size_factor,
- estd_lc_time_saved,
- estd_lc_memory_object_hits
- FROM v\$shared_pool_advice;





9. Pin Objects to Keep in Pool:

- -- Pin large PL/SQL packagesEXEC
DBMS_SHARED_POOL.KEEP('SCHEMA.PACKAGE_NAME', 'P');-- List kept
objectsSELECT name, type, kept
- FROM v\$db_object_cache
- WHERE kept = 'YES';

10. Memory Usage by Module:

- SELECT
- module,COUNT(*) cursors,SUM(sharable_mem)/1024/1024 total_mb
- FROM v\$sqlWHERE module IS NOT NULLGROUP BY module
- ORDER BY total_mb DESC;

11. Reserved Pool Usage:

- -- Check if using reserved pool
- SELECT free_space, avg_free_size,
- free_count, used_space, avg_used_size,
- used_count
- FROM v\$shared_pool_reserved;

12. Quick Health Check:

- -- Shared pool health metricsSELECT
- metric_name,value,
- metric_unit
- FROM v\$sysmetric
- WHERE metric_name IN ('Library Cache Hit Ratio','Cursor Cache Hit
Ratio','Memory Sorts Ratio')AND group_id = 2; -- Last minute



4

Oracle Data Access

**"Read ahead is the optimizer's
foresight. Blocks are fetched before
they are needed."**

4.1 Data Access

- **Oracle data access tuning** refers to the process of optimizing how your application retrieves data from an Oracle database by analyzing and adjusting query execution plans, indexing strategies, and other database configuration settings to achieve the best possible performance for your specific data access patterns.

1. Chained Rows:

The Internet of Things (IoT) is a network of devices that can communicate with each other and with the internet

2. Internet of Things:

A chained row is a row that is too large to fit into a single database data block



Data Access



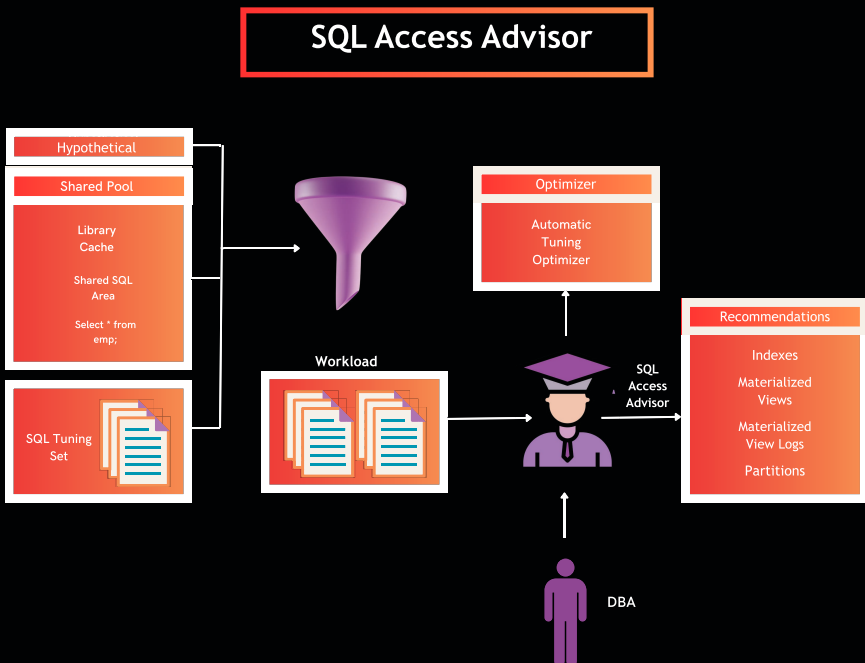
- **Toxinomy of Oracle data access tuning:**
- **SQL Query Optimization:**
 - a. Analyzing execution plans: Using the "explain plan" feature to understand how Oracle intends to execute a query and identify potential bottlenecks like full table scans or inefficient joins.
 - b. Index optimization: Creating appropriate indexes on frequently queried columns to improve data retrieval speed.
 - c. Query rewriting: Modifying the SQL statement to improve its efficiency, such as adding filters, using appropriate join types, and avoiding unnecessary calculations.

Oracle Data Access

- **Database Configuration Tuning:**
 - a. **Parameter settings:** Adjusting database parameters like cache sizes, session parameters, and optimizer settings to optimize performance based on your workload.
 - b. **Statistics management:** Keeping database statistics up-to-date to ensure the optimizer generates accurate execution plans.
- **Application Design Considerations:**
 - a. **Batching queries:** Grouping multiple related queries together to minimize round trips to the database.
 - b. **Parameterization:** Using bind variables to avoid repetitive query parsing and improve performance.
 - c. **Efficient data access patterns:** Choosing appropriate data access methods like cursors or bulk operations depending on the scenario.
- **Oracle Data Access:**
- To achieve optimal performance for data-intensive queries, materialized views and indexes are essential for SQL statements. However, implementing these objects does not come without cost. Creation and maintenance of these objects can be time-consuming. Space requirements can be significant. SQL Access Advisor enables you to optimize query access paths by recommending materialized views and view logs, indexes, SQL profiles, and partitions for a specific workload.

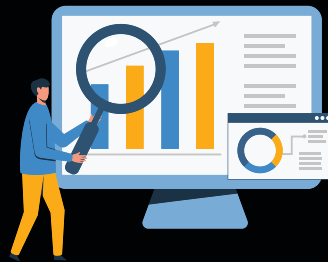
Optimizing Data Access Path

- SQL Access Advisor also recommends bitmap, function-based, and B-tree indexes. A **bitmap index** reduces response time for many types of ad hoc queries and can **also reduce storage space compared to other indexes**. A **function-based index** computes the value of a function or expression involving **one or more** columns and stores it in the index. B-tree indexes are commonly used to **index unique or near-unique keys**.
- Using SQL Access Advisor involves the following tasks:
 - Running SQL Access Advisor
 - Reviewing the SQL Access Advisor Recommendations
 - Implementing the SQL Access Advisor Recommendations

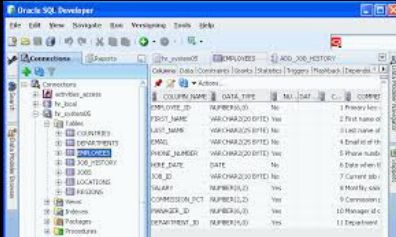


4.2 Tools Oracle Data Access

- **Commands used for Oracle Data Access:**
- To tune Oracle data access, key commands include: "explain plan" to visualize the execution plan of a query, "set autotrace on" to trace statement execution, "select * from v\$sql" to view currently executing SQL statements, "analyze table" to gather statistics on tables, and utilizing the "SQL Tuning Advisor" to automatically suggest optimizations based on workload analysis, all accessible through SQL*Plus or other Oracle tools.
- **Oracle Data Access Tuning commands:**
- **Explain Plan:**
- Provides a visual representation of how Oracle plans to execute a query, highlighting potential bottlenecks in the execution path.
- **Autotrace:**
- Enables detailed tracing of SQL statement execution, including timing information and execution plan details.
- **V\$SQL Views:**
- Dynamic performance views that offer insights into currently running SQL statements, including execution counts, execution times, and associated cursors.
- **Analyze Table:**
- Collects statistics about table data distribution, which the optimizer uses to generate efficient execution plans.
- **SQL Tuning Advisor:**
- An Oracle tool that analyzes SQL statements and recommends potential optimizations based on execution plans and statistics.

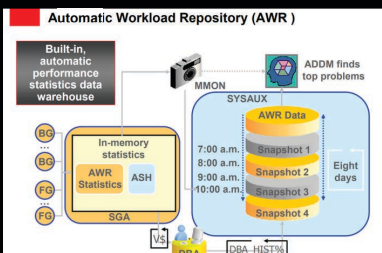
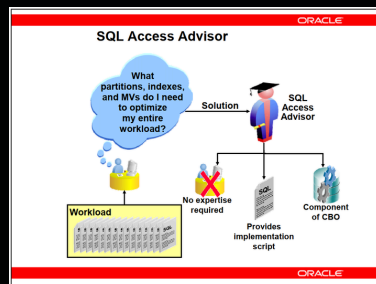


Tools in Oracle Data Access



- **Tools used for ODA :**
- **Oracle SQL Developer:**
- Provides a visual interface to analyze execution plans, view statistics, and generate tuning recommendations.

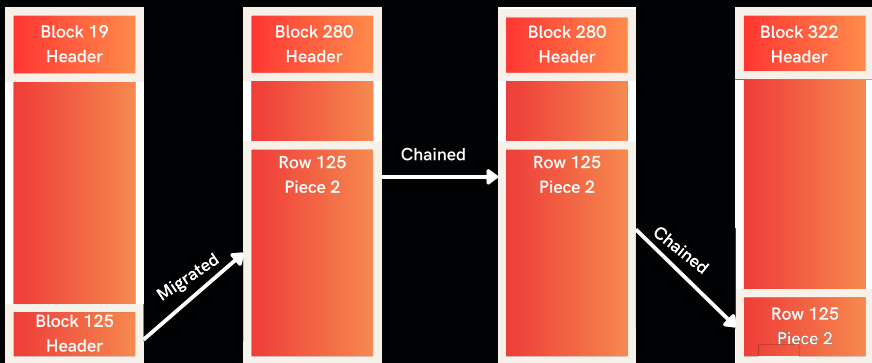
- **SQL Tuning Advisor:**
- An Oracle feature that automatically suggests optimizations for SQL statements based on analysis of execution plans and statistics.



- **Automatic Workload Repository (AWR):**
- Monitors database activity and identifies performance bottlenecks to suggest potential tuning actions.

4.3 Chained Rows

- **Chained Rows:**
- Chained rows in data access refers to a database scenario where a single row of data is too large to fit within a single data block, causing it to be split across multiple blocks, essentially forming a "chain" of blocks that need to be accessed sequentially to retrieve the entire row, which can significantly impact query performance due to increased I/O.



- **How to identify chained rows:**
- Analyze table command: Most database systems provide a command to analyze tables and identify the presence of chained rows, often including metrics like the percentage of chained rows.
- **How to mitigate chained rows:**
- Optimize data types: Choose appropriate data types to minimize the size of each column.
- Normalize data: Break large tables into smaller, more manageable tables with appropriate relationships.
- Adjust block size: In some cases, adjusting the database block size might be necessary, but should be considered carefully as it can impact other aspects of performance.

Identify Chained Rows

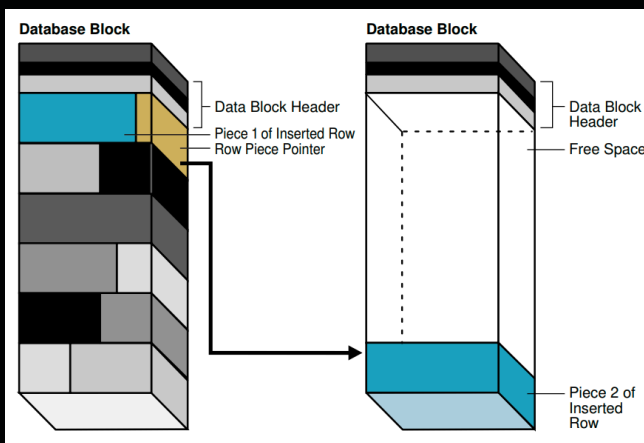
Chained Rows:

- A chained row is a row that is too large to fit into a single database data block.
- For example, if you use a 4KB blocksize for your database, and you need to insert a row of 8KB into it, Oracle will use 3 blocks and store the row in pieces.

- **Key points about chained rows:**
- **Cause:**
- Typically happens when a table has columns with large data types or when a row is updated to become larger than the block size.
- **Impact on performance:**
- Accessing a chained row requires reading multiple blocks from storage, leading to slower query execution, especially when many rows are chained.
- **Database systems where it occurs:**
- This is particularly relevant in Oracle databases, where the concept of "row chaining" is commonly discussed.

Identify Chained Rows

- **Some conditions that will cause row chaining are:**
 - Tables whose row size exceeds the blocksize
 - Tables with long and long row columns are prone to having chained rows
 - Tables with more then 255 columns will have chained rows as Oracle break wide tables up into pieces.
- **Detecting row chaining**
 - This query will show how many chained (and migrated) rows each table has:
`SELECT owner, table_name, chain_cnt FROM dba_tables WHERE chain_cnt > 0;`
- **To see which rows are chained:**
 - `ANALYZE TABLE tablename LIST CHAINED ROWS;`
 - This will put the rows into the CHAINED_ROWS table which is created by the utlchain.sql script (in `$ORACLE_HOME/rdbms/admin`).
 - `SELECT * FROM chained_rows;`



4.4 Internet of Things(IoT)

- **Internet of Things(IoT):**
- Data access for the Internet of Things (IoT) is the process of making data available for authorized users and systems to retrieve and use. It's important for making decisions and improving operational efficiency.
- **How is data access achieved in IoT?**
- **Cloud-based storage:** Provides on-demand access to data from anywhere
- **APIs:** Allow data to be exchanged between systems
- **Web portals and apps:** Provide access to data through a web browser or mobile app



Internet of Things(IoT)

- **What are some considerations for data access in IoT?**
- **Security:** How secure is the platform?
- **User-friendliness:** How easy is it to use the platform?
- **Device settings:** How easy is it to change device settings?
- **Network connectivity:** How well connected are the devices?
- **User authentication:** How are users authenticated?

- **Benefits of data access in IoT**
- **Improved decision-making:** Data-driven insights can help organizations make better decisions
- **Increased innovation:** Access to data can lead to new products and services
- **Improved operational efficiency:** Data-driven insights can streamline processes and reduce costs
- **Enhanced customer experience:** Data-driven personalization can improve customer satisfaction



IoT- Oracle Database

- **Internet of Things(IoT): Oracle Database**
- An example of an IoT Oracle database could be a system where a network of industrial sensors on a factory floor continuously transmits data like machine temperature, vibration levels, and production output to an Oracle database, allowing for real-time monitoring, anomaly detection, and predictive maintenance analysis, leveraging the database's capabilities to handle high volumes of data efficiently; essentially using Oracle to store and analyze data from connected machines in a manufacturing environment.



Example - Internet of Things(IoT)

- **Key points about this example:**
- **Data types:**
 - The database would store various sensor data types like numerical values (temperature, pressure), timestamps, and potentially event flags indicating potential issues.
- **Index-Organized Tables (IOT):**
 - To optimize the retrieval of large amounts of time-series data, Oracle's IOT feature could be used, which stores data in a sorted manner based on the timestamp, allowing for quick queries on recent or historical data.
- **Stream processing:**
 - Integration with Oracle's streaming capabilities could be used to process data in real-time as it arrives from sensors, enabling immediate alerts for critical situations.
- **Other IoT scenarios using Oracle database:**
- **Smart home monitoring:**
 - Storing sensor data from smart devices like thermostats, lights, and security cameras to analyze energy usage and potential security threats.
- **Fleet management:**
 - Tracking vehicle location, fuel consumption, and engine diagnostics from connected vehicles in a fleet, allowing for optimized routing and maintenance scheduling.
- **Wearable device data:**
 - Storing health metrics like heart rate, steps taken, and sleep patterns from wearable devices for personalized health insights.

4.5 Data Manipulation

1

Insert

The INSERT statement is a core DML command used to add new rows or records to a table.



Update

Use the UPDATE statement to change data values in one or more columns, usually based on specific criteria.

2

3

Delete

The DELETE command is a Data Manipulation Language (DML) command that removes records from a table in a database.



Rollback

ROLLBACK is the SQL command that is used for reverting changes performed by a transaction.

4



5

Commit

The COMMIT command is a Transaction Control Language (TCL) statement that makes the changes within a transaction permanent, saving them to the database.



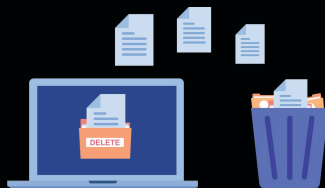
DML - Statements

- **Oracle Data Manipulation:**
- Oracle data manipulation is the process of adding, changing, or removing data in Oracle Database tables. This is done using Data Manipulation Language (DML) statements. You can use DML statements in the SQL*Plus or SQL Developer environments.
- **DML Statements:**
- INSERT: Adds records to a table
- UPDATE: Changes values in a record
- DELETE: Removes records from a table
- ROLLBACK: Restores the database to the condition before the changes were made
- COMMIT: Permanently saves the changes
- Bulk insert, move, delete, using Partition
- **Transactions:**
- A transaction is a sequence of one or more SQL statements that Oracle treats as a unit
- Until a transaction is committed, it can be rolled back
- Committing a transaction makes its changes permanent
- **Entering DML Statements:**
- In the SQL*Plus environment, you can enter a DML statement after the SQL> prompt
- In the SQL Developer environment, you can enter a DML statement in the SQL Worksheet
- Committing a transaction:
- Committing a transaction makes its changes permanent, erases its savepoints, and releases its locks



DML Environment

- **Oracle Data Manipulation:**
- Data manipulation language (DML) statements access and manipulate data in existing tables.
- In the **SQL*Plus environment**, you can enter a DML statement after the SQL> prompt.
- In the **SQL Developer environment**, you can enter a DML statement in the Worksheet. Alternatively, you can use the SQL Developer Connections frame and tools to access and manipulate data.
- To see the effect of a DML statement in SQL Developer, you might have to select the schema object type of the changed object in the Connections frame and then click the Refresh icon.
- The effect of a DML statement is not permanent until you commit the transaction that includes it. A transaction is a sequence of SQL statements that Oracle Database treats as a unit (it can be a single DML statement). Until a transaction is committed, it can be rolled back (undone). For more information about transactions, see "About Transaction Control Statements".
- **About the INSERT Statement:**
- The INSERT statement inserts rows into an existing table.
- **About the UPDATE Statement:**
- The UPDATE statement updates (changes the values of) a set of existing table rows.
- **About the DELETE Statement:**
- The DELETE statement deletes rows from a table.



Bulk insert/move/delete

- Bulk Insert and delete
- Hi, I have two tables: employees and emp_arch.
- My requirement is insert rows from employees to emp_arch and delete these inserted rows from employees.
- I created this procedure but row count is really big (about one million). And I have to do this operation periodically.
- CREATE OR REPLACE procedure SI_EMP_DEL_INSERT1 IS
- CURSOR si_c_emp
- IS
- SELECT m1.ROWID rrid, m1.*
- FROM employees m1
- WHERE salary < 5000;
- BEGIN
- FOR c2 IN si_c_emp
- LOOP
- EXECUTE IMMEDIATE 'INSERT INTO emp_arch
- SELECT *
- FROM employees
- WHERE ROWID = ''' || c2.rrid || ''';
- COMMIT;
- DELETE FROM EMPLOYEES WHERE rowid =c2.rrid;
- commit;
- COMMIT;
- END LOOP;
- END;

Bulk insert/move/delete

- **Bulk Insert and delete**

- I wonder is it possible to do it with bulk insert and delete.
- I try to write it with using bulk option like this:
- But here I confused how to delete these inserted rows.
- create or replace procedure SI_EMP_DEL_INSERT2 is
- cursor si_c_emp is
- select employee_id,
first_name,last_name,email,phone_number,hire_date,job_id,salary,commission_pct,
manager_id,department_id from employees where salary<5000;
- type si_t_emp_table is table of employees%rowtype;
- si_t_emp si_t_emp_table;
- begin
- open si_c_emp;
- loop
- fetch si_c_emp bulk collect into si_t_emp;
- forall si_index in 1..si_t_emp.count save exceptions
- insert into emp_arch values si_t_emp(si_index);
- exit when si_c_emp%notfound;
- end loop;
- close si_c_emp;
- EXCEPTION WHEN OTHERS THEN
- if SQLCODE=-24381 then
- for si_index_2 in 1..SQL%BULK_EXCEPTIONS.count
- loop
- begin
- dbms_output.put_line(SQL%BULK_EXCEPTIONS(si_index_2).error_index);
- end;
- end loop;
- else RAISE;
- end if;
- end;



5

Storage

"Oracle storage has two faces: logical and physical. The database knows the logical; the operating system sees the physical."

Storage



1. **Optimize Storage:**

Storage Optimization is the process of saving data in a way that allows it to occupy less space, cost less, and work more efficiently.

2. **Storage Performance:**

To optimize storage performance, consider techniques like data deduplication, compression, and storage tiering and optimization of your storage drives

3. **Storage Type:**

Choosing appropriate storage types (like EBS SSD), ensuring sufficient IOPS, and improve performance.

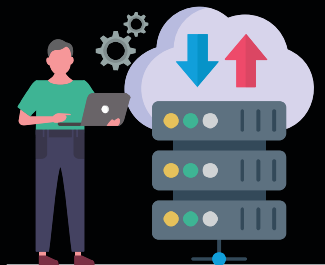
4. **Logical & Physical Storage:**

Physical storage refers to the actual physical hardware where data is stored, while logical storage describes how the database organizes data, providing a structured and abstract view.

5.1 Optimize Storage

- Your IT team will likely want to minimize the need to source additional storage, in order to keep storage costs low. You can reduce costs by partitioning underneath large databases.

- Generally, businesses use **two storage tiers**. The first, the **high-end storage array**, is used by **mission-critical applications**. The second is used by **applications that have lower requirements**, and it **costs less**. However, the average database features a mix of active data and passive data. Active data – for example, call records – may only use a small portion of your database storage. Passive data – e.g., orders taken in the previous year – usually takes up significantly more storage space. If you want to optimize your storage performance, you can size your disk storage so that it matches the workload.



- **Random input/output (I/O) processes** are actuated by inserts, deletes, and updates, which are activated by queries scanning the table in batches. A random I/O operation is measured in I/O operations per second. **Sequential I/O operations** are measured in the number of megabytes that are scanned per second (MB/s). Storage arrays should be sized based on the number of I/O operations per second that they can deliver. To figure out the MB/s and total number of I/O operations per second required for your current databases, check the reports for the workload repository.

5.2 Storage Performance



To improve **Oracle database storage performance**, focus on optimizing disk I/O, memory regions (SGA and PGA), and SQL statements through indexing, statistics collection, and query optimization.

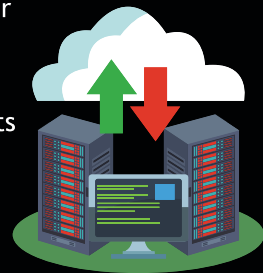
key areas:

1. Disk I/O Optimization:

- **Storage Hardware:** Ensure you have fast and reliable storage (e.g., SSDs).
- **RAID Configuration:** Use appropriate RAID levels for your workload (e.g., RAID 10 for high availability and performance).
- **I/O Queuing:** Monitor and adjust I/O queue lengths to prevent bottlenecks.
- **Data Placement:** Strategically place frequently accessed data on faster storage devices.

2. Memory Region Optimization (SGA & PGA):

- **SGA (Shared Global Area):**
 - **Buffer Cache:** Ensure sufficient buffer cache size to minimize disk I/O for data access.
 - **Redo Log Buffer:** Optimize the redo log buffer size to handle transaction logs efficiently.
 - **Shared Pool:** Adjust the shared pool size to accommodate frequently used SQL statements and library cache.
- **PGA (Program Global Area):**
 - **PGA_AGGREGATE_TARGET:** Set the PGA aggregate target to an appropriate value based on your workload and available memory.
 - **Memory for Sorts and Hashing:** Ensure sufficient PGA memory for sorting and hashing operations.



Storage Performance

3. SQL Statement Optimization:

- **Indexing:** Create appropriate indexes to speed up data retrieval.
- **Statistics:** Regularly collect and update table statistics to help the optimizer choose efficient execution plans.
- **Query Optimization:**
- **Rewrite Queries:** Rewrite poorly performing queries to improve performance.
- **Use Hints:** Use query hints judiciously to guide the optimizer.
- **Partitioning:** Consider partitioning large tables to improve query performance.
- **Statement Caching:** Utilize statement caching to reduce parsing overhead.

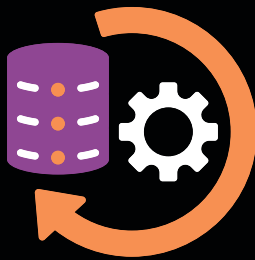


4. Monitoring and Tuning:

- Automatic Workload Repository.
- Use AWR to monitor database performance and identify potential bottlenecks.
- Automatic Database Diagnostic Monitor (ADDM):
- Use ADDM to analyze database performance and suggest tuning recommendations.
- Performance Tuning Tools:
- Utilize Oracle's built-in performance tuning tools and third-party tools to monitor & analyze performance.

5.3 Storage Type

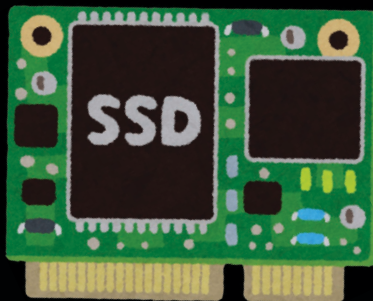
- **For optimal Oracle database performance**, focus on efficient storage management, including choosing appropriate storage types (like EBS SSD), ensuring sufficient IOPS, and monitoring storage utilization to prevent bottlenecks.
- Here's a more detailed breakdown:
 1. **Choosing the Right Storage Type:**
 - For high and consistent IOPS:
 - Amazon **EBS General Purpose SSD** or **Amazon EBS Provisioned IOPS SSD** volumes are well-suited for Oracle Database workloads.
 - **Considerations for Oracle Cloud Infrastructure (OCI):**
 - **Use block storage** for the database system's root volume and data volumes, and consider using a high-performance storage option like "**Block Storage with High Performance**" for optimal performance.
 - Monitor storage space utilization:
 - **Use tools like Health Monitor** and HeatWave Service metric, DB Volume Utilization, to track storage space utilization and get warnings about potential shortages.



Storage Optimization

2. Optimizing Storage Performance:

- Ensure Sufficient IOPS:
- IOPS (Input/Output Operations Per Second) is a crucial factor for database performance. Make sure your storage solution provides enough IOPS to handle the database workload.
- Optimize SQL Statements:
- Tune SQL statements to minimize disk I/O by using appropriate indexes, query hints, and collecting statistics.
- Monitor Top SQL Statements:
- Identify and address high-cost SQL statements that are causing performance issues.
- Identify Resource Bottlenecks:
- Use tools like Automatic Database Diagnostic Monitor (ADDM) and Automatic Workload Repository (AWR) to identify and address resource bottlenecks.
- Proactive Tuning:
- Regularly monitor and tune your database to maintain optimal performance.



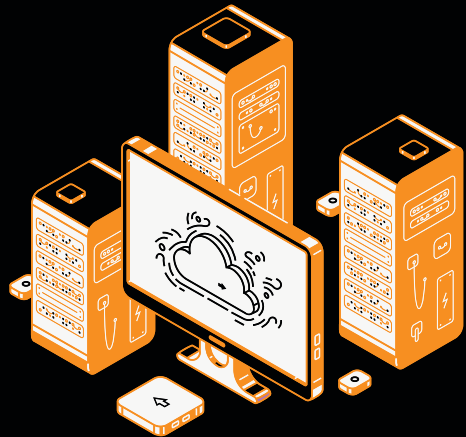
Storage Capacity

3. Increasing Storage Capacity:

- Consider the initial storage size:
- The number of volumes you can have depends on the initial storage size of your DB system.
- Maximum Storage Size:
- The maximum storage size varies based on the initial storage size and the number of volumes.
- Check the storage of a DB system:
- Use Health Monitor and HeatWave Service metric, `DbVolumeUtilization` , to track storage space utilization for a DB system and to warn you if there is a shortage of space on the disk.

4. Oracle Storage Solutions:

- Oracle offers storage solutions designed to accelerate diverse customer workloads with low latencies and high throughput.
- These solutions can automate Oracle Database protection, allowing DBAs to back up business-critical databases faster and recover them quickly.



5.4 Logical & Physical Storage

- **An Oracle database has both physical and logical structures.**
- **Physical structures** are the files on disk, while logical structures are the structures created by the Oracle database server
- **Physical structures:**
 - Data files: Contain data structures like tables and indexes
 - Control files: Track the physical components of the database
 - Online redo log: Contains records of changes made to data
- **Logical structures :**
 - Tablespaces: The primary logical structure in a database
 - Segments: Contain one or more extents, which contain multiple data blocks
 - Extents: Contain multiple data blocks
 - Data blocks: Contain information about the block, including disk address and segment type
 - Index blocks: Manage the logical storage space in an index
 - Benefits of logical storage Helps users locate specific data, Improves the efficiency of the retrieval process, and Enables a modular data storage system.
- **Managing the database structure:**
 - Oracle database server can automate the management of its structure
 - Oracle Enterprise Manager provides a Web-based GUI to enable easy management and monitoring of your database



6

Network

"On commit refreshes immediately; on demand waits for the call. One trusts the transaction; the other trusts the scheduler."

Network



1. Mview & Benefits

In Oracle, a materialized view is a read-only table that stores the results of a query, enabling faster access to frequently accessed data, especially when the underlying data changes infrequently

2. Tune Mview SQL

If you want to optimize Oracle view performance, you should begin by tuning the underlying query. Once you have done this, the next step is to use the view to tune the query.

3. Mview Methods & Mlog

In Oracle databases, an "mview" refers to a materialized view, a pre-computed, stored representation of a query result, and "mlog" (or MLOG\$) is the table associated with the master table of a materialized view that stores change information for fast refresh. .

4. Mview Refresh & Management

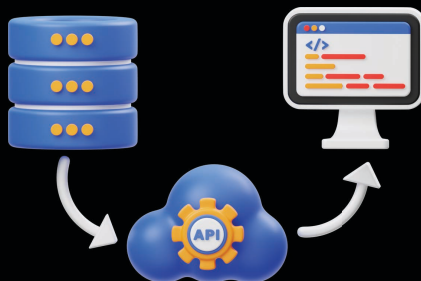
you can refresh materialized views using either complete or fast refresh methods, with options for on-demand / on-commit refreshes. Fast refresh is generally faster, but requires creating materialized view logs on the base tables.

5. Stored Procedure

A stored procedure is a named set of PL/SQL statements designed to perform an action. Stored procedures are stored inside the database. They define a programming interface for the database rather than allowing the client application to interact with database objects directly.

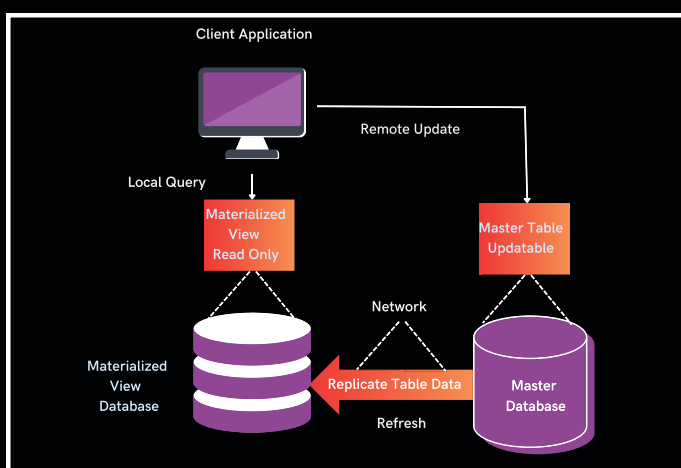
6.1 Mview and Benefits

- To improve Oracle database performance, especially with complex queries, consider using materialized views (MVs)
- which precompute and store query results, offering faster query performance by reducing recomputation. However, ensure MVs are properly managed and refreshed to maintain data consistency.
- To improve performance with Oracle Materialized Views (MVs), focus on tuning the underlying query, ensuring the MV is refreshed efficiently (using ON COMMIT or ON DEMAND with appropriate refresh methods), and optimizing MV indexes and storage.
- **Here's a breakdown of performance tuning with Oracle Materialized Views:**
 - **Understanding Materialized Views:**
 - Materialized views are tables that store the results of a query, allowing for faster retrieval of data compared to repeatedly executing the original query.
 - They are particularly useful for complex queries, data warehousing, and distributed databases.
 - By precomputing and storing query results, materialized views reduce the load on source tables and improve query response times.



View and Benefits

- **Benefits of Using Materialized Views:**
- **Reduced Execution Time:** Complex queries with joins and aggregate functions can be significantly faster.
- **Improved Query Performance:** Oracle's optimizer can automatically use deployed materialized views to improve query execution plans.
- **Low Maintenance:** Incremental data changes from base tables are automatically added to materialized views in a synchronous manner.
- **Enable local access,** which improves response times and availability
- When the materialized view is in a different database than its source, offload queries from the master database, because users can query the local materialized view instead
- **Increase data security** by enabling you to replicate only a selected subset of the target master's data set



Mview and Read-Only MView Concepts:

- A **Materialized View (MV)** is a database object that stores the results of a query, providing a physical representation of the query's output for faster data retrieval, especially beneficial for complex queries and data warehousing.
- **Replication Databases:**
 - Replication is the process of copying and maintaining database objects, such as tables, in multiple databases that comprise a distributed database system. One method that supports replication is read-only materialized views.
 - Replication environments support two basic types of databases: master databases and materialized view databases. Materialized view databases contain an image, or materialized view, of the table data from a certain point in time. The table on which a materialized view is defined is the master table for the materialized view. Typically, a materialized view is refreshed periodically to synchronize it with its master table.
- **Read-Only Materialized Views:**
 - A read-only materialized view contains a complete or partial copy of a target master table from a single point in time. A partial copy can include a subset of row, a subset of columns, or both.
 - Read-only materialized views can provide read-only access to the master table's data. Applications can query data from read-only materialized views to avoid network access to the master database, regardless of network availability. However, applications throughout the system must access data at the master database to perform data manipulation language changes (DML). Figure 36-1 illustrates basic, read-only replication.

6.2 Tune Mview SQL

- **Tuning the Underlying Query:**
- **Optimize the SQL:**
- Before creating an MV, ensure the query you intend to use for the MV is well-optimized.
- **Use Indexes:**
- Properly index the tables involved in the MV query, especially columns used in WHERE clauses, JOIN conditions, and ORDER BY clauses.
- **Analyze Tables:**
- Collect statistics on the tables using DBMS_STATS to help the optimizer choose the best execution plan.
- **Use Query Hints:**
- If necessary, use query hints to guide the optimizer towards a more efficient execution plan.
- **Understand Execution Plans:**
- Use EXPLAIN PLAN to analyze the execution plan of the query and identify potential bottlenecks.



Mview Use



- **When to Use Materialized Views:**
- **Complex Queries:** When queries involve multiple joins, aggregations, or other computationally expensive operations.
- **Data Warehousing:** For storing and querying historical data.
- **Distributed Databases:** For replicating data across multiple databases.
- **When to Avoid Materialized Views:**
- **Real-Time Data:** Materialized views are not ideal for applications requiring real-time data accuracy because they store snapshots of data.
- **Frequent Updates:** If the underlying data is frequently updated, the overhead of refreshing the materialized view may outweigh the benefits.



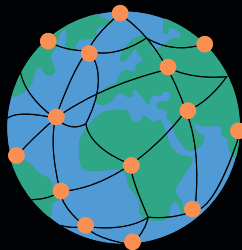
- **Tools for Performance Tuning:**
- **V\$ Performance Views:** These views provide performance information used by Oracle Database performance tuning tools.
- **Automatic Workload Repository (AWR):** Oracle recommends using AWR to gather performance data.
- **EXPLAIN PLAN:** This statement displays execution plans chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements.

Mview & Example

- **-- Create a materialized view**
- CREATE MATERIALIZED VIEW my_mv AS
- SELECT column1, column2, SUM(column3) AS total
- FROM my_table
- WHERE condition1
- GROUP BY column1, column2;

- **-- Refresh the materialized view on demand**
- REFRESH MATERIALIZED VIEW my_mv;

- **-- Refresh the materialized view on commit (fast refresh)**
- CREATE MATERIALIZED VIEW my_mv_fast
- REFRESH FAST ON COMMIT
- AS
- SELECT column1, column2, SUM(column3) AS total
- FROM my_table
- WHERE condition1
- GROUP BY column1, column2;



6.3 Mview methods & Mlog

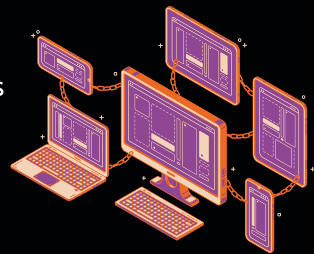
Materialized View Creation and Refresh:

- Choose the Right Refresh Method:
- **ON COMMIT (Fast Refresh):** Ideal for frequently updated data, where the MV needs to reflect changes quickly. However, it can impact DML performance on the underlying tables.
- **ON DEMAND (Fast Refresh):** Allows you to control when the MV is refreshed, but requires manual or scheduled refresh processes.
- **COMPLETE Refresh:** Recomputes the MV from scratch, useful for infrequent updates or when data is heavily changed.



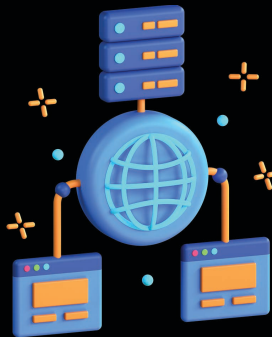
Materialized View Logs:

- If using fast refresh, create materialized view logs on the detail tables involved in the MV.
- **Index the MV:**
- Create indexes on the MV to improve query performance, especially on columns used in WHERE clauses and JOIN conditions.
- **Consider Parallelism:**
- Use the PARALLEL hint during MV creation and refresh to speed up the process.
- **TUNE_MVIEW Procedure:**
- Use the DBMS_MVIEW.TUNE_MVIEW procedure to add additional aggregate columns and materialized view logs, making the MV fast refreshable.



Mview Refresh Methods

- **Performance Tuning Considerations:**
- **Materialized View Refresh:** Ensure the materialized view is kept up-to-date with the underlying data.
- **Indexing:**
- Create indexes on the columns of the detail table that are specified in the SELECT statement that creates the join.
- Consider creating an index on the materialized view itself to improve the performance of keeping the materialized view updated.
- **Materialized View Logs:**
- For fast refresh, create materialized view logs on all detail tables involved in a materialized view.
- **Performance Impact of Updates, Inserts, and Deletes:** The performance of UPDATE, INSERT, and DELETE operations may be impacted if the updated table is referenced in a materialized view.



6.4 Mview refresh and Mgmt

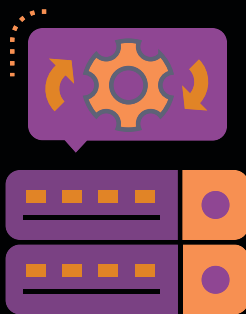
- **Refresh Performance Tuning:**
- **Monitor Refresh Time:** Track the time it takes to refresh the MV and identify any bottlenecks.
- **Tune Refresh Processes:** Adjust the refresh method, frequency, and parameters to optimize performance.
- **Use DBMS_SYNC_REFRESH or DBMS_MVIEW Packages:** Use the APIs in these packages for synchronous or asynchronous refresh.
- **Consider Real-Time Materialized Views:** For very high-frequency data changes, consider real-time materialized views.



- **Materialized View Management:**
- **Monitor MV Usage:**
- Track which MVs are being used and how effectively they are improving performance.
- **Measure Space Usage:**
- Monitor the space used by MVs and ensure they are not consuming excessive storage.
- **Identify Unused MVs:**
- Regularly review MVs and drop those that are no longer used or are not improving performance.
- **Archive Old Data:**
- Archive old detail and MV data that is no longer useful to free up storage space.

6.5 Stored Procedure

- A **Stored Procedure** is a **set of Structured Query Language (SQL) statements** with an assigned name, which are stored in a relational database management system (**RDBMS**) as a group, so it can be **reused and shared by multiple programs**.
- Stored procedures can access or modify data in a database, but it is not tied to a specific database or object, which offers a number of advantages.
- **Benefits of using stored procedures:**
- A stored procedure provides an important layer of security between the user interface and the database. It supports security through data access controls because end users may enter or change data, but do not write procedures. A stored procedure preserves data integrity because information is entered in a consistent manner. It improves productivity because statements in a stored procedure only must be written once.



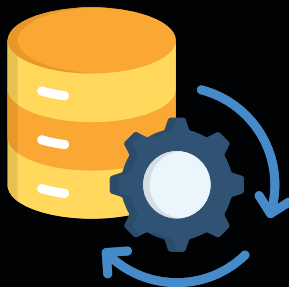
Stored Procedure Vs Function

- **Stored Procedure Vs. Function:**
- Stored procedures and Functions can be used to accomplish the same task. Both can be custom-defined as part of any application, but functions are designed to send their output to a query or T-SQL statement. Stored procedures are designed to return outputs to the application, while a user-defined function returns table variables and cannot change the server environment or operating system environment.
- **CALL (PROCEDURE) Statement:**
- The CALL (PROCEDURE) statement is used to call procedures. A call to a procedure does not return any value.
- When a procedure with definer's rights is called, the current default schema is set to the eponymously named schema of the definer. For example, if the defining user is called OWNER, the default schema will also be set to OWNER. The definer's rights include the right to set the current role to a role for which the definer has privileges. When the procedure is first invoked, no role is set; even if the invoker has set a current role, the procedure running with definer's rights has no current role set initially.



Stored Procedure Vs Function

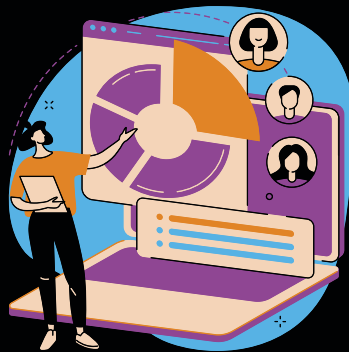
- When a procedure with invoker's rights is called, the current default schema and current role are unchanged initially within the procedure. Similarly, if SQL authorization mode is not enabled, the current default schema is unchanged initially within the procedure.
- When the call returns, any changes made inside the procedure to the default current schema (and current role, if relevant) are reset (popped).
- For information about definer's rights, see EXTERNAL SECURITY.
- **Syntax:**
- `CALL procedure-Name ( [ expression [, expression]* ] )`
- **Example:**
- `CREATE PROCEDURE SALES.TOTAL_REVENUE(IN S_MONTH INTEGER,IN S_YEAR INTEGER, OUT TOTAL DECIMAL(10,2))
PARAMETER STYLE JAVA READS SQL DATA LANGUAGE JAVA
EXTERNAL NAME
'com.example.sales.calculateRevenueByMonth';`
- `CALL SALES.TOTAL_REVENUE(?,?,?)`



Advantages of Stored Procedures

1. Performance

- Stored procedures are compiled once and stored in an executable form. As a result, procedure calls are quick and efficient. Executable code is automatically cached and shared among users. This lowers memory requirements and invocation overhead.
- By grouping SQL statements, a stored procedure allows the statements to be processed with a single call. This reduces network traffic and improves round-trip response time.
- Additionally, stored procedures enable you to take advantage of the computing resources of the server. For example, you can move computation-bound procedures from client to server, where they will run faster. Stored functions enhance performance by running application logic within the server.



Advantages of Stored Procedures

2. Productivity and Ease of Use:

- By designing applications around a common set of stored procedures, you can avoid redundant coding and increase the productivity. Moreover, stored procedures let you extend the functionality of the database.

3. Scalability:

- Java in the database inherits the scalable session model of Oracle Database. Stored procedures increase scalability by isolating application processing on the server. In addition, automatic dependency tracking for stored procedures helps in developing scalable applications.

4. Maintainability

- After a stored procedure is validated, you can use it with confidence in any number of applications. If its definition changes, then only the procedure is affected, not the applications that call it. This simplifies maintenance and enhancement. Also, maintaining a procedure on the server is easier than maintaining copies on different client computers.

5. Interoperability

- Java in Oracle Database fully conforms to the Java Language Specification (JLS) and furnishes all the advantages of a general-purpose, object-oriented programming language. Also, as with PL/SQL, Java provides full access to Oracle data. As a result, any procedure that is written in PL/SQL can also be written in Java.

Advantages of Stored Procedures

- **Replication:**

- With Oracle Advanced Replication, you can replicate stored procedures from one Oracle Database instance to another. This enables you to use stored procedures to implement a central set of business rules. Once you write the procedures, you can replicate and distribute them to work groups and branch offices throughout the company. In this way, you can revise policies on a central server rather than on individual servers.

- **Security:**

- **Security is a large arena that includes:** Network security for the connection
- Access and execution control of operating system resources or of JVM and user-defined classes
- Bytecode verification of JAR files imported from an external source.
- You can restrict access to Oracle data by allowing users to manipulate the data only through stored procedures that run with their definer's privileges. For example, you can allow access to a procedure that updates a database table, but deny access to the table itself.



7

Oracle Reports

"AWR is the repository of record. It does not diagnose; it remembers. Every hour, it writes down what the database did, where it waited, and how long it hesitated."

"ASH in memory speaks every second. ASH on disk speaks every ten seconds. The archive remembers one-tenth of the truth."

"ADDM is the physician who never sleeps. After every snapshot, it reads the chart. It finds the fever before you feel it."

Oracle Reports

- **AWR Report:**

Automatic Workload Repository (AWR) collects, processes, and maintains performance statistics for problem detection and self-tuning purposes.

- **ASH Report:**

ASH report contains data that helps you to: Triage transient performance problems that typically last for a few minutes

- **ADDM Report:**

Automatic performance diagnostic report every hour by default Database & Redo Buffer



7.1 What is AWR Report?

- An Oracle **AWR (Automatic Workload Repository) report** is a tool for diagnosing DB performance issues by analyzing data collected between two points in time, offering insights into workload profiles, wait events, SQL performance & more.
- **What is an AWR Report?**
- **Purpose:** AWR reports are designed to help database administrators (DBAs) identify and resolve performance bottlenecks and understand the overall health of an Oracle database.
- **Data Collection:** AWR collects performance statistics from various database dynamic performance views (like V\$SYSSTAT, V\$SYSTEM_EVENT, V\$SEGMENT_STATISTICS, and V\$SQL) at regular intervals, storing them in the AWR repository.
- **Snapshot:** Each data collection is called an AWR snapshot, and an AWR report shows statistics captured between two snapshots.



- **AWR reports provide a comprehensive view of the database's workload, including:**
- **Workload Profile:** Provides an overview of the system's activity during the selected time period.
- **Top SQL Statements:** Identifies SQL statements that are consuming the most resources (CPU, elapsed time).
- **Wait Events:** Shows which wait events are causing the database to wait, helping pinpoint bottlenecks.
- **System and Session Statistics:** Provides detailed information about system and session activity.
- **I/O Statistics:** Helps identify potential I/O issues.
- **Memory Statistics:** Provides insights into memory usage and potential memory problems.

Benefits of AWR Report

- **Performance Tuning:**
 - AWR reports are invaluable for identifying and resolving database performance issues.
- **Capacity Planning:**
 - They provide insights into the database's workload, which can be used for capacity planning and resource allocation.
- **Problem Detection:**
 - AWR reports can help detect performance problems early, before they impact users.
- **Baseline:**
 - AWR reports can be used to create baselines of database performance, allowing for easier comparison of performance over time.
- **Troubleshooting:**
 - AWR reports can help troubleshoot replication performance issues by categorizing them as workload, database-side misconfiguration, or performance bottleneck issues.



AWR & AWR generation command

- **Oracle Automatic Workload Repository (AWR):**
- AWR is a monitoring tool that collects, analyzes, and stores performance statistics, helping database administrators identify and resolve performance problems. When tuning a database, each component has its own unique set of statistics. To provide a common scale for comparison, database performance statistics are reported in a database-timed model. This model offers details on how much time is spent on performing database activities.
- **Running Oracle AWR Reports:**
- One can generate the AWR report either through Oracle Enterprise Manager or via command line.
- **Running AWR from Command Line:**
- To run AWR from the command line on the server where the database is installed, go to \$ORACLE_HOME/rdbms/admin
- Login to database:
- sqlplus “/as sysdba”
- @awrrpt.sql
- Type in HTML or txt for report type
- Enter the value for num_days, for the number of days you want to see AWR snapshots.
- Enter the start and end snapshots, enter filename, and select enter.
- Running AWR from Oracle Enterprise Manager
- Access the AWR Report by navigating to Performance -> AWR -> AWR Report on the database page.

AWR - Performance Hub Page



- **Automatic Workload Repository Report (AWR):**
 - To download an AWR report, identify the database version and follow the respective steps.
 - Open the Performance Hub page from the database details page, if not already open.
- **For Oracle Database 19c and later:**
 1. In the Quick Select menu, select a time period for the AWR report.
 2. Click the Reports button in the upper-right corner and select Automatic Workload Repository. It opens the Generate Automatic Workload Repository dialog box.
 3. Select the begin time of the snapshot from Start Snapshot and the end time from End Snapshot.
 - You can also get information about a specific instance and monitor its performance. This option is not enabled by default. To include details of a particular instance in the AWR report, click the check box Instance Level Report and select the instance from the drop-down list.
 4. Click Download to generate the report and download it.
 - Oracle Database generates the report in the format AWRReport_date_range.html for the selected time range. When the report is complete, the report name is displayed at the top of the screen, and the report is downloaded automatically.

How to Generate an AWR Report?



- **Oracle Enterprise Manager:**
 - You can also generate AWR reports through Oracle Enterprise Manager.
- **AWR Explorer:**
 - In Oracle Database Cloud, you can use AWR Explorer to analyze database performance and generate AWR reports.
- **HTML Report:**
 - AWR reports are typically generated in HTML format, allowing for easy navigation and analysis.
- **Performance Hub:**
 - You can generate AWR reports through the Performance Hub in Oracle Cloud Infrastructure (OCI).
 - Steps:
 - Open the Performance Hub page from the database details page.
 - Click the "Reports" button in the upper-right corner.
 - Select "Automatic Workload Repository".
 - In the "Generate Automatic Workload Repository" dialog box, you can:
 - Generate a report from two snapshots close to the current date and time.
 - Generate a report from a custom time range.
 - Include details of a particular instance in the AWR report (for Oracle Real Application Clusters (RAC) databases).
 - Click "Download" to download the AWR report.



Reading AWR Report

- **Reading AWR Reports**

- The AWR has multiple sections that cover various database activities. To find what you need quickly, you may want to navigate directly to the appropriate section. Important sections in the AWR are discussed below:

- **Top Timed Events**

- The Top Timed Events section provides an overview of CPU and I/O constraints and displays where the most time is spent waiting for resources. Further breakdowns are available in the Foreground and Background Events sections.

- **System I/O Statistics**

- In the System I/O Statistics section, you can view the overall system statistics and confirm if the I/O subsystem can handle the I/O demands made by the database. You can also verify the redo log switches and adjust the size of the redo logs to ensure that there are 3 to 4 switches per hour.

- **SQL CPU & I/O Consumption**

- In this section, you can find details about the SQLs that were executed during the AWR interval. This section is important because it helps identify which SQLs were using the most system resources, leading to performance degradation. The SQLs with high consumption of CPU, I/O, or unoptimized reads are good candidates for tuning.

- **Segment Statistics**

- The Segment Statistics section provides a list of the most active segments. You can review this list to determine if any of the segments could benefit from a reorg or purpose of data, etc.



Scripts 1 : AWR



1. Generate AWR Report (Database Level)

-- Text report for last hour

```
SELECT * FROM TABLE(
  DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT(
    l_dbid    => (SELECT dbid FROM v$database),
    l_inst_num => (SELECT instance_number FROM v$instance),
    l_bid     => (SELECT snap_id - 1 FROM (
      SELECT snap_id FROM dba_hist_snapshot
      ORDER BY snap_id DESC FETCH FIRST 1 ROWS ONLY)),
    l_eid     => (SELECT snap_id FROM dba_hist_snapshot
      ORDER BY snap_id DESC FETCH FIRST 1 ROWS ONLY)
  )
);
```

2. Generate AWR Report with Snap IDs

-- Get snapshots first

```
SELECT
  snap_id,
  TO_CHAR(begin_interval_time, 'DD-MON-YY HH24:MI') begin_time,
  TO_CHAR(end_interval_time, 'DD-MON-YY HH24:MI') end_time
FROM dba_hist_snapshot
ORDER BY snap_id DESC
FETCH FIRST 10 ROWS ONLY;
```

-- Generate report between two snapshots

```
SELECT OUTPUT FROM TABLE(
  DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT(
    l_dbid    => &dbid,
    l_inst_num => &instance_number,
    l_bid     => &begin_snap_id,
    l_eid     => &end_snap_id
  )
);
```

3. Generate HTML AWR Report

-- HTML format (save to file)

SET PAGESIZE 0 LONG 1000000 LINESIZE 1000

SPOOL awr_report.html

```
SELECT OUTPUT FROM TABLE(
  DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_HTML(
    (SELECT dbid FROM v$database),
    (SELECT instance_number FROM v$instance),
    &begin_snap_id,
    &end_snap_id
  )
);
```





4. Compare Periods AWR Report

```
-- Compare two time periods
SELECT OUTPUT FROM TABLE(
  DBMS_WORKLOAD_REPOSITORY.AWR_DIFF_REPORT_TEXT(
    l_dbid => (SELECT dbid FROM v$database),
    l_inst_num => (SELECT instance_number FROM v$instance),
    l_bid1 => &begin_snap1,    l_eid1 => &end_snap1,    l_bid2 =>
&begin_snap2,
    l_eid2 => &end_snap2
  )
);
```

5. Quick AWR Summary

```
-- Top 5 wait events from last snapshot
SELECT
  event_name, total_waits, time_waited_micro,
  ROUND(time_waited_micro/1000000,2) seconds
FROM dba_hist_system_event
WHERE snap_id = (SELECT MAX(snap_id) FROM dba_hist_snapshot)
ORDER BY time_waited_micro DESC
FETCH FIRST 5 ROWS ONLY;
```

6. AWR Snapshot Management

```
sql
-- List snapshots
SELECT snap_id, begin_interval_time,
  end_interval_time
FROM dba_hist_snapshot
WHERE begin_interval_time > SYSDATE - 7
ORDER BY snap_id DESC;

-- Create manual snapshot
EXEC DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT();

-- Drop old snapshots
EXEC DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE(
  low_snap_id => &begin_snap,
  high_snap_id => &end_snap);
```



Scripts 3 : AWR



7. Key AWR Metrics

sql

-- Top metrics from last hour

```
SELECT
    metric_name,
    ROUND(value,2) value,
    metric_unit
FROM dba_hist_sysmetric_summary
WHERE snap_id = (SELECT MAX(snap_id) FROM dba_hist_snapshot)
AND metric_name IN (
    'CPU Usage Per Sec',
    'Database CPU Time Ratio',
    'Database Wait Time Ratio',
    'Physical Reads Per Sec',
    'Logical Reads Per Sec'
);
```

8. ASH (Active Session History) Report

sql

-- Generate ASH report for last 15 minutes

```
SELECT OUTPUT FROM TABLE(
    DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_TEXT(
        l_dbid    => (SELECT dbid FROM v$database),
        l_inst_num => (SELECT instance_number FROM v$instance),
        l_btime    => SYSDATE - 15/1440, -- 15 minutes ago
        l_etime    => SYSDATE
    )
);
```

9. AWR Retention Settings

sql

-- Check current AWR settings

```
SELECT
    retention,
    interval,
    topnsql
FROM dba_hist_wr_control;
```

-- Modify retention (days) and interval (minutes)

```
EXEC DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(
    retention => 43200, -- 30 days in minutes
    interval  => 60); -- 60 minutes
```



Scripts 4: AWR



10. Top SQL from AWR

```
-- Top 10 SQL by elapsed time
SELECT
    sql_id,   executions_delta,   elapsed_time_delta/1000000 elapsed_sec,
    cpu_time_delta/1000000 cpu_sec,
    ROUND(elapsed_time_delta/NULLIF(executions_delta,0)/1000,2) avg_ms
FROM dba_hist_sqlstat
WHERE snap_id = (SELECT MAX(snap_id) FROM dba_hist_snapshot)
AND elapsed_time_delta > 0
ORDER BY elapsed_time_delta DESC
FETCH FIRST 10 ROWS ONLY;
```

11. Generate AWR to File Automatically

```
-- Script to generate AWR for last hour
DEFINE begin_snap = "SELECT MAX(snap_id)-1 FROM dba_hist_snapshot"
DEFINE end_snap   = "SELECT MAX(snap_id) FROM dba_hist_snapshot"
DEFINE dbid       = "SELECT dbid FROM v$database"
DEFINE inst_num   = "SELECT instance_number FROM v$instance"
```

```
SET TERMOUT OFF PAGES 0 LINES 2000 TRIMSPOOL ON
SPOOL awr_last_hour.html
```

```
SELECT OUTPUT FROM TABLE(
    DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_HTML(
        &dbid,   &inst_num,   &begin_snap,
        &end_snap
    )
);
```

```
SPOOL OFF
SET TERMOUT ON
```

12. Quick AWR Health Check

```
-- Load profile comparison
SELECT   metric_name,
        ROUND(value,2) current_value,
        ROUND(average,2) hourly_avg
FROM dba_hist_sysmetric_summary
WHERE snap_id = (SELECT MAX(snap_id) FROM dba_hist_snapshot)
AND metric_name IN (
    'Logical Reads Per Sec',
    'Physical Reads Per Sec',
    'Redo Generated Per Sec',
    'User Calls Per Sec',
    'Executions Per Sec'
);
```





One-Liners for Common Tasks:

1. Last hour AWR:

```
SPOOL awr.html
SELECT OUTPUT FROM
TABLE(DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_HTML((SELECT dbid FROM
v$database),(SELECT instance_number FROM v$instance),(SELECT MAX(snap_id)-1 FROM
dba_hist_snapshot),(SELECT MAX(snap_id) FROM dba_hist_snapshot)));
SPOOL OFF
```

2. Generate for specific time:

```
SELECT &begin_snap, &end_snap FROM (SELECT snap_id, begin_interval_time FROM
dba_hist_snapshot
WHERE begin_interval_time BETWEEN SYSDATE-2 AND SYSDATE-1);
```

3. Check AWR space usage:

```
SELECT * FROM dba_hist_wr_control;
```

4. Quick top wait events:

```
SELECT * FROM TABLE(DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT((SELECT
dbid FROM v$database),(SELECT instance_number FROM v$instance),(SELECT
MAX(snap_id)-1 FROM dba_hist_snapshot),(SELECT MAX(snap_id) FROM
dba_hist_snapshot)))WHERE OUTPUT LIKE '%Top 5 Timed%' OR OUTPUT LIKE '%Wait
Class%';
```

Analysis Tips:

- **Load Profile:** Check for spikes
- **Top Waits:** Identify bottlenecks
- **SQL Stats:** Find expensive SQL
- **Instance Efficiency:** Should be >90%
- **Compare periods** for trend analysis
- Use **AWR Diff** for before/after changes



7.2 Active Session History(ASH)

- **The Active Session History (ASH)** is a diagnostic tool that records the information about all the active sessions in an Oracle database.
- The ASH diagnostic tool captures transient performance problems by taking samples of active sessions every second and storing the sampled data in a circular buffer in the shared global area (SGA). Any session that is connected to the database and is waiting for an event that does not belong to the Idle wait class is considered as an active session. By capturing only active sessions, a manageable set of data is represented with its size being directly related to the work being performed, rather than the number of sessions allowed on the system.
- ASH enables you to examine and perform detailed analysis on the sampled session activity using the V\$ACTIVE_SESSION_HISTORY view. The data present in ASH can be rolled up in various dimensions that it captures over a specified duration and gathered into an ASH report.



- **Generating Active Session History Reports**
- **ASH reports enable you to perform analysis of:**
 - Transient performance problems that typically last for a few minutes
 - Scoped or targeted performance analysis by various dimensions or their combinations, such as time, session, module, action, or SQL identifier
 - ASH reports are divided into multiple sections. The HTML report includes links that can be used to navigate quickly between sections. The content of the report contains ASH information used to identify blocker and waiter identities, their associated transaction identifiers, and SQL statements for a specified duration.

ASH - Performance Hub Page



- **To download an ASH report:**
- Open the Performance Hub page from the database details page, if not already open.
- Note: For more information about how to navigate to the Performance Hub page, see:
- **For Autonomous Database**
- **For Base Database Service and Exadata Databases**
- **For External Database Service**
- From the menu in the upper-right corner, click Reports and select Active Session History. The Generate ASH Report dialog box is displayed.
- Select the date of the Start Time. In the calendar and time dialog boxes that are displayed, select the start date and time for the report.
- Repeat this step for the End Time date and time.
- Click Download to generate the report for the selected time range and download it.
- While the report is being generated, a message Report generation is in progress appears at the upper-right corner of the screen. When the report is complete and downloaded, a confirmation message with the name of the report file is displayed.

Generating an ASH Report Command-Line



- **Generating an ASH Report Using the Command-Line Interface**
- **Generating an ASH Report on the Local Database Instance**
- The ashrpt.sql SQL script generates an HTML or text report that displays ASH information for a specified duration on the local database instance.
- To generate an ASH report on the local database instance using the command-line interface:
- At the SQL prompt, enter:
@\$ORACLE_HOME/rdbms/admin/ashrpt.sql
- **Generating an ASH Report on a Specific Database Instance**
- The ashrpti.sql SQL script generates an HTML or text report that displays ASH information for a specified duration on a specified database and instance.
- To generate an ASH report on a specific database instance using the command-line interface:
- At the SQL prompt, enter: **@\$ORACLE_HOME/rdbms/admin/ashrpti.sql**
- **Generating an ASH Report for Oracle RAC**
- The ashrpti.sql SQL script generates an HTML or text report that displays ASH information for a specified duration for specified databases and instances in an Oracle Real Application Clusters (Oracle RAC) environment. Only ASH data that is written to disk will be used to generate the report. This report will only use ASH samples from the last 10 minutes that are found in the DBA_HIST_ACTIVE_SESS_HISTORY table.
- **To generate an ASH report for Oracle RAC:**
- **@\$ORACLE_HOME/rdbms/admin/ashrpti.sql**

ASH - Usage



- While AWR is great for historical analysis, ASH is what you'll rely on when a user calls saying, "The system is slow right now!" Just generate an ASH report and check:
- Session Activity: What are users or batch jobs doing?
- Top Blocking Sessions: Who's holding the locks?

- **Step-by-Step Process to Use ASH Reports:**

- If you're wondering how to make the most of ASH reports, here's a quick guide:

1. **Look for Active Sessions**

- Focus on sessions with high wait times.
- Identify whether these sessions are user queries, background processes, or system jobs.

2. **Analyze Wait Events**

- Check for bottlenecks like I/O waits, latch contention, or log buffer waits.
- Match these events with SQL IDs to understand which queries or processes are causing delays.

3. **Investigate Blocking Sessions**

- Find sessions holding locks or causing deadlocks.
- Use the SID (Session ID) to kill or debug problematic sessions.

4. **Trace SQL Execution**

- Use the SQL IDs from the ASH report to trace specific queries.
- Check execution plans to identify inefficiencies, such as full table scans or missing indexes.

5. **Monitor Resource Consumption**

- Check CPU, memory, and disk I/O usage for each session.
- Focus on sessions consuming abnormally high resources.

6. **Save and Review**

- Always save ASH reports for future reference. Patterns often emerge over time, helping you build a clearer picture of database behavior.

Scripts 1: ASH



1. Generate ASH Report (Current Session)

```
-- HTML ASH report for last 15 minutes
SET PAGESIZE 0 LONG 1000000 LINESIZE 200
SPOOL ash_report.html
SELECT OUTPUT FROM TABLE(
  DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_HTML(
    l_dbid    => (SELECT dbid FROM v$database),
    l_inst_num => (SELECT instance_number FROM v$instance),
    l_btime   => SYSDATE - 15/1440, -- 15 minutes ago
    l_etime   => SYSDATE
  )
);
SPOOL OFF
```

2. Text ASH Report

```
-- Text format ASH report
SELECT OUTPUT FROM TABLE(
  DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_TEXT(
    (SELECT dbid FROM v$database),
    (SELECT instance_number FROM v$instance),
    SYSDATE - 30/1440, -- Last 30 minutes
    SYSDATE
  )
);
```

3. Quick ASH Session Analysis

```
-- Active sessions in last 5 minutes
SELECT  COUNT(*) active_sessions, session_state, wait_class
FROM v$active_session_history
WHERE sample_time > SYSDATE - 5/1440
GROUP BY session_state, wait_class
ORDER BY active_sessions DESC;
```

4. Top SQL from ASH

```
-- Top SQL consuming resources last 10 minutes
SELECT  ash.sql_id, sql.sql_text, COUNT(*) samples,
        ROUND(COUNT(*)*10,0) approx_secs -- Each sample ~1 second
FROM v$active_session_history ash
JOIN v$sql sql ON ash.sql_id = sql.sql_id
WHERE ash.sample_time > SYSDATE - 10/1440 AND ash.session_state = 'ON CPU'
GROUP BY ash.sql_id, sql.sql_text
ORDER BY samples DESC
FETCH FIRST 10 ROWS ONLY;
```



Scripts 2: ASH



5. Blocking Sessions Analysis

```
-- Blocking sessions in last 15 minutes
SELECT   blocking_session,
         COUNT(*) blocked_samples,
         event,   sql_id
FROM v$active_session_history
WHERE blocking_session IS NOT NULL
AND sample_time > SYSDATE - 15/1440
GROUP BY blocking_session, event, sql_id
ORDER BY blocked_samples DESC;
```

6. ASH Time Series

```
-- Activity by minute
SELECT   TO_CHAR(sample_time, 'HH24:MI') minute,   COUNT(*) active_sessions,
         SUM(CASE WHEN session_state = 'ON CPU' THEN 1 ELSE 0 END) on_cpu,
         SUM(CASE WHEN session_state = 'WAITING' THEN 1 ELSE 0 END) waiting
FROM v$active_session_history
WHERE sample_time > SYSDATE - 60/1440 -- Last hour
GROUP BY TO_CHAR(sample_time, 'HH24:MI')
ORDER BY minute;
```

7. Wait Analysis from ASH

```
-- Top wait events last 30 minutes
SELECT   event,   wait_class,
         COUNT(*) total_waits,
         ROUND(COUNT(*)*10,0) approx_wait_secs
FROM v$active_session_history
WHERE session_state = 'WAITING'
AND sample_time > SYSDATE - 30/1440
GROUP BY event, wait_class
ORDER BY total_waits DESC
FETCH FIRST 10 ROWS ONLY;
```

8. Module/Program Analysis

```
-- Activity by module/program
SELECT   NVL(module, program) source,   COUNT(*) samples,
         ROUND(100*COUNT(*)/SUM(COUNT(*) OVER(),1) pct
FROM v$active_session_history
WHERE sample_time > SYSDATE - 15/1440
GROUP BY NVL(module, program)
ORDER BY samples DESC;
```





9. Historical ASH (DBA_HIST_ACTIVE_SESS_HISTORY)

```
-- ASH analysis for specific time yesterday
SELECT TO_CHAR(sample_time, 'HH24:MI') time,
       session_state, event, COUNT(*) samples
FROM dba_hist_active_sess_history
WHERE sample_time BETWEEN
      TRUNC(SYSDATE-1) + 10/24 AND -- Yesterday 10AM
      TRUNC(SYSDATE-1) + 11/24 -- Yesterday 11AM
GROUP BY TO_CHAR(sample_time, 'HH24:MI'), session_state, event
ORDER BY time, samples DESC;
```

10. Generate ASH for Specific Session

```
-- ASH report for specific session ID
SELECT OUTPUT FROM TABLE(
  DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_TEXT(
    (SELECT dbid FROM v$database),
    (SELECT instance_number FROM v$instance),
    SYSDATE - 5/1440, -- Last 5 minutes
    SYSDATE,
    0, -- Not used
    0, -- Not used
    &session_id, -- Your session ID
    &serial_num -- Your serial#
  )
);
```

11. ASH for Specific SQL_ID

```
-- Analyze specific SQL execution
SELECT sample_time, session_state, event,
       wait_time, time_waited
FROM v$active_session_history
WHERE sql_id = '&sql_id'
AND sample_time > SYSDATE - 60/1440
ORDER BY sample_time DESC;
```

12. Real-Time ASH Monitoring

```
sql
-- Live ASH every 5 seconds (run multiple times)
SELECT TO_CHAR(SYSDATE, 'HH24:MI:SS') current_time,
       COUNT(*) active_sessions,
       SUM(CASE WHEN session_state = 'ON CPU' THEN 1 ELSE 0 END) on_cpu,
       SUM(CASE WHEN session_state = 'WAITING' THEN 1 ELSE 0 END) waiting,
       LISTAGG(event, ', ') WITHIN GROUP (ORDER BY COUNT(*) DESC) top_events
FROM v$active_session_history
WHERE sample_time > SYSDATE - 10/1440 -- Last 10 seconds
GROUP BY 1;
```



Scripts 4: ASH



13. ASH Disk I/O Analysis

```
-- Top files/objects causing I/O waits
SELECT  obj.object_name,  ash.current_obj#,  ash.event,  COUNT(*) samples
FROM v$active_session_history ash
LEFT JOIN dba_objects obj ON ash.current_obj# = obj.object_id
WHERE ash.event LIKE '%db file%'
AND ash.sample_time > SYSDATE - 15/1440
GROUP BY obj.object_name, ash.current_obj#, ash.event
ORDER BY samples DESC
FETCH FIRST 10 ROWS ONLY;
```

14. Quick ASH Diagnostic Scripts

Check current ASH buffer usage:

```
SELECT  'ASH Buffer' metric,  ROUND(SUM(bytes)/1024/1024,2) size_mb
FROM v$sgstat
WHERE pool = 'shared pool'
AND name LIKE '%ASH%';
Generate ASH for last incident:
```

-- Find last ASH sample

```
SELECT
  MIN(sample_time) first_sample,  MAX(sample_time) last_sample,  COUNT(*) total_samples
FROM v$active_session_history
WHERE sample_time > SYSDATE - 1/24; -- Last hour
```

-- Generate report for that period

```
SELECT OUTPUT FROM TABLE(
  DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_HTML(
    (SELECT dbid FROM v$database),
    (SELECT instance_number FROM v$instance),
    &begin_time,
    &end_time
  )
);
```

One-liner ASH report (last 10 minutes):

```
SPOOL ash.html
SELECT OUTPUT FROM TABLE(DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_HTML(
  (SELECT dbid FROM v$database),(SELECT instance_number FROM v$instance),
  SYSDATE-10/1440,SYSDATE));
SPOOL OFF
```

ASH vs AWR comparison:

-- ASH shows real-time, AWR shows historical trends

-- ASH samples every second, AWR every hour by default

```
SELECT  'ASH' source,  COUNT(*) samples,  MIN(sample_time) first,  MAX(sample_time) last
FROM v$active_session_history
UNION ALL
```

```
SELECT  'AWR' source,  COUNT(*) snapshots,  MIN(begin_interval_time) first,  MAX(end_interval_time) last
FROM dba_hist_snapshot;
```

ASH Analysis Tips:

High ON CPU: Check CPU-bound SQL

High WAITING: Identify wait events

Blocking sessions: Look for locks

Time patterns: Identify peak periods

SQL_ID focus: Drill into specific queries

Compare with AWR: ASH for details, AWR for trends

Most useful: Generate HTML ASH report for last 15 minutes when performance issue occurs.



7.3 Automatic Database Diagnostic Monitor (ADDM)

You can run the **Automatic Database Diagnostic Monitor (ADDM)** manually to monitor **current and historical database** performance. Typically, you use the automatic diagnostic feature of ADDM to identify performance problems with the database. As described in Automatic Database Performance Monitoring, **ADDM runs once every hour by default**. You can configure ADDM to run at a different time interval. However, in some cases you may want to run ADDM manually. You can run ADDM manually to analyze a time period that is longer than one ADDM analysis period. For example, you may want to analyze database performance in a workday by analyzing 8 consecutive hours. You could analyze each of the individual ADDM periods within the workday, but this approach may become complicated if performance problems appear in only some ADDM periods.

Alternatively, you can run ADDM manually with a pair of Automatic Workload Repository (AWR) snapshots that encompass the 8-hour period. In this case, ADDM identifies the most critical performance problems in the entire time period. Manually Running ADDM to Analyze Current Database Performance By default, ADDM runs every hour to analyze snapshots taken by AWR during this period. In some cases you may notice performance degradation that did not exist in the previous ADDM analysis period, or a sudden spike in database activity on the Performance page, as described in Monitoring Real-Time Database Performance . If the next ADDM analysis is not scheduled to run for 30 minutes, then you can run ADDM manually to identify and resolve the performance problem. When you run ADDM manually, a manual AWR snapshot is created automatically. This manual run may affect the ADDM run cycle. For example, if you scheduled ADDM to run hourly at the start of each hour and the last ADDM run was at 8:00 p.m., running ADDM manually at 8:30 p.m. causes the next scheduled run to start at 9:30 p.m., not 9:00 p.m. Subsequent ADDM runs continue on the new run cycle, occurring hourly at the half-hour instead of the start of each hour.

ADDM analyze current DB performance

To analyze current database performance by manually running ADDM:

1. Access the Database Home page.

See "Accessing the Database Home Page" for more information.

2. From the Performance menu, select Advisors Home.

If the Database Login page appears, then log in as a user with administrator privileges. The Advisor Central page appears.

3. Under Advisors, select ADDM.

The Run ADDM page appears.

4. Select Run ADDM to analyze current performance and click OK.

The Confirmation page appears.

5. Click Yes. The Processing:

Run ADDM Now page appears while the database takes a new AWR snapshot. An ADDM run occurs for the time period between the new and the previous snapshot. After ADDM completes the analysis, the Automatic Database Diagnostic Monitor (ADDM) page appears with the results.

6. Click View Report.

7. The View Report page appears.

Optionally, click **Save to File** to save the results of the ADDM task in a report for later access.



ADDM to Analyze Historical DB Performance



Manually Running ADDM to Analyze Historical Database Performance You can run ADDM manually to analyze historical database performance by selecting a pair or range of AWR snapshots as the analysis period. This technique is useful when you have identified a previous time period when database performance was poor. In the Performance page, you can monitor historical performance by selecting Historical from the View Data list.

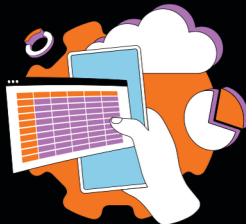
In the Historical view, you can monitor database performance in the past, up to the duration defined by the AWR retention period. If you notice performance degradation, then you can drill down from the Performance page to identify historical performance problems with the database, as described in *Monitoring Real-Time Database Performance* . If you identify a problem, then you can run ADDM manually to analyze a particular time period. To analyze historical database performance by manually running ADDM: Access the Database Home page: See "Accessing the Database Home Page" for more information. From the Performance menu, select Advisors Home. If the Database Login page appears, then log in as a user with administrator privileges

The Advisor Central page appears. Under Advisors, select ADDM. The Run ADDM page appears. Select Run ADDM to analyze past performance. Specify a time period for analysis by selecting a pair of AWR snapshots. Complete the following steps: Select Period Start Time. Below the chart for the starting snapshot, click the snapshot you want to use for the start time. A play icon (displayed as an arrow) appears over the snapshot icon.

Accessing Previous ADDM Results

- **Accessing Previous ADDM Results**
- If you ran ADDM manually to analyze current or historical database performance, the results are displayed on the Automatic Database Diagnostic Monitor (ADDM) page after the ADDM run has completed.
- You can access the ADDM results at a later time, or access the ADDM results from previous run cycles.
- To access the ADDM results:
- Access the Database Home page.
- See "Accessing the Database Home Page" for more information.
- From the Performance menu, select Advisors Home.
- If the Database Login page appears, then log in as a user with administrator privileges. The Advisor Central page appears.
- **Complete the following steps:**
- Running ADDM from command line:
- Under AdvisorTasks, select ADDM from the Advisory Type list.
- Select the appropriate search criteria. For example, you can select All in the Advisor Runs list to view all ADDM tasks.
- Click Go.

- Running ADDM from command line:
- SQL script for getting ADDM Report on RAC database:
- SQL> @\$ORACLE_HOME/rdbms/admin/addmrpti.sql
- SQL script for getting ADDM Report for single instance:
- SQL> @\$ORACLE_HOME/rdbms/admin/addmrpt.sql



ADDM & Analysis

The Automatic Database Diagnostic Monitor (ADDM):

It is a tool that analyzes data in the [Automatic Workload Repository \(AWR\)](#) performing accurate and timely diagnosis of the problem. It locates the root cause and provides recommendations for correcting the problem. For those who are new to performance tuning, AWR (Automatic Workload Repository) is a built-in repository (in the SYS_AUX tablespace) that exists in every Oracle Database. At regular intervals, the Oracle Database makes a snapshot of all of its vital statistics and workload information and stores them in the AWR to know more about AWR check our previous blog on Automatic Workload Repository (AWR): Database Statistics.

• The ADDM Analysis Includes the following:

- CPU load
- Memory usage
- I/O usage Resource
- Intensive SQL Resource
- Intensive PL/SQL
- Java RAC issues
- Application issues
- Database configuration issues
- Concurrency issues
- Object contention



ADDM monitor & Recommendation



- **ADDM also monitors the non-problem areas of the system like wait event classes**
- that are not significantly impacting the system performance, they are identified and removed from the tuning consideration at an initial stage, saving time and effort that would have been spent on items that do not impact overall system performance.
- ADDM not only recommends problem diagnostics it also provides possible solutions for the problem.
- ADDM recommends one or more solutions for the DBA to choose from consisting of a variety of recommended solutions including:
 1. Hardware changes
 2. Database configuration
 3. Schema changes
 4. Application changes
 5. Using other advisors



ADDM Analysis:

- An ADDM analysis will be performed on a pair of AWR snapshots and a collection of instances from equivalent information. The pair of AWR snapshots outline the period of time for analysis, and the set of instances define the target for analysis.
- If you're using Oracle Real Application Clusters (Oracle RAC).
- The findings (problems) are listed in order of potential impact on database performance, along with recommendations to resolve the issue and the symptoms which lead to its discovery.

Scripts 1: ADDM



1. Generate ADDM Report (Last Analysis)

```
-- Get latest ADDM task
SELECT task_id, task_name, execution_start, execution_end
FROM dba_advisor_tasks WHERE advisor_name = 'ADDM'
ORDER BY execution_start DESC
FETCH FIRST 1 ROWS ONLY;

-- Generate report for latest task
SET LONG 1000000 LONGCHUNKSIZE 1000
SELECT DBMS_ADVISOR.GET_TASK_REPORT('&task_name', 'TEXT', 'ALL')
FROM dual;
```

2. ADDM Report for Specific Time Period

```
-- Run ADDM analysis for last hour
DECLARE
    task_name VARCHAR2(30);
BEGIN
    task_name := 'ADDM_' || TO_CHAR(SYSDATE, 'YYYYMMDD_HH24MI');
    DBMS_ADVISOR.CREATE_TASK('ADDM', task_name);
    DBMS_ADVISOR.SET_TASK_PARAMETER(
        task_name,
        'START_SNAPSHOT',
        (SELECT MAX(snap_id)-1 FROM dba_hist_snapshot)
    );
    DBMS_ADVISOR.SET_TASK_PARAMETER(
        task_name,
        'END_SNAPSHOT',
        (SELECT MAX(snap_id) FROM dba_hist_snapshot)
    );
    DBMS_ADVISOR.EXECUTE_TASK(task_name);
END;
/

-- Get the report
SELECT DBMS_ADVISOR.GET_TASK_REPORT(task_name, 'TEXT', 'ALL')
FROM dual;
```

3. Quick ADDM Findings Summary

```
-- Recent ADDM findings
SELECT f.finding_name, f.type, f.impact, f.message
FROM dba_advisor_findings f
JOIN dba_advisor_tasks t ON f.task_id = t.task_id
WHERE t.advisor_name = 'ADDM'
AND t.execution_start > SYSDATE - 7
ORDER BY t.execution_start DESC, f.impact DESC;
```

4. HTML ADDM Report

```
-- Generate HTML report
SET PAGESIZE 0 LONG 1000000 LINESIZE 200
SPOOL addm_report.html
SELECT DBMS_ADVISOR.GET_TASK_REPORT(
    (SELECT task_name FROM dba_advisor_tasks
     WHERE advisor_name = 'ADDM'
     ORDER BY execution_start DESC FETCH FIRST 1 ROWS ONLY),
    'HTML',
    'ALL'
) FROM dual;
SPOOL OFF
```



Scripts 2: ADDM



5. ADDM Recommendations

```
-- Get recommendations from latest ADDM
SELECT rec.rank, rec.type, rec.title, rec.message, f.finding_name
FROM dba_advisor_recommendations rec
JOIN dba_advisor_findings f ON rec.finding_id = f.finding_id
JOIN dba_advisor_tasks t ON f.task_id = t.task_id
WHERE t.advisor_name = 'ADDM'
AND t.execution_start = (SELECT MAX(execution_start)
                        FROM dba_advisor_tasks
                        WHERE advisor_name = 'ADDM')
ORDER BY rec.rank;
```

6. List All ADDM Tasks

```
-- View all ADDM analyses
SELECT task_name, execution_start, execution_end, status
FROM dba_advisor_tasks
WHERE advisor_name = 'ADDM'
ORDER BY execution_start DESC;
```

7. ADDM for Specific Snapshots

```
-- Create ADDM for specific snapshot range
DECLARE
    task_name VARCHAR2(30);
BEGIN
    task_name := 'ADDM_CUSTOM_' || TO_CHAR(SYSDATE, 'HH24MISS');
    DBMS_ADVISOR.CREATE_TASK('ADDM', task_name);
    DBMS_ADVISOR.SET_TASK_PARAMETER(task_name, 'START_SNAPSHOT', &begin_snap);
    DBMS_ADVISOR.SET_TASK_PARAMETER(task_name, 'END_SNAPSHOT', &end_snap);
    DBMS_ADVISOR.EXECUTE_TASK(task_name);
    DBMS_OUTPUT.PUT_LINE('Task created: ' || task_name);
END;
```

/

8. Quick ADDM Health Check

```
-- Top 5 findings from last analysis
SELECT f.finding_name, f.impact_percent, f.message
FROM dba_advisor_findings f
WHERE task_id = (SELECT MAX(task_id)
                FROM dba_advisor_tasks
                WHERE advisor_name = 'ADDM')
AND ROWNUM <= 5
ORDER BY f.impact_percent DESC;
```

9. ADDM Actions

```
-- Recommended actions from ADDM
SELECT a.command, a.attr1, a.attr2, a.attr3, r.message
FROM dba_advisor_actions a
JOIN dba_advisor_recommendations r ON a.rec_id = r.rec_id
JOIN dba_advisor_tasks t ON r.task_id = t.task_id
WHERE t.advisor_name = 'ADDM'
AND t.execution_start > SYSDATE - 1
ORDER BY a.action_id;
```

10. One-Line ADDM Report Generation

```
-- Generate ADDM for last hour automatically
EXEC DBMS_ADVISOR.QUICK_TUNE('ADDM', 'ADDM_QUICK');
SELECT DBMS_ADVISOR.GET_TASK_REPORT('ADDM_QUICK', 'TEXT') FROM dual;
```



Scripts 3: ADDM



Quick Commands:

1. Last ADDM report:

```
SELECT DBMS_ADVISOR.GET_TASK_REPORT(  
  (SELECT task_name FROM dba_advisor_tasks  
   WHERE advisor_name='ADDM'  
   ORDER BY execution_start DESC FETCH FIRST 1 ROWS ONLY),  
  'TEXT'  
) FROM dual;
```

2. Check if ADDM is enabled:

```
SELECT parameter_name, parameter_value  
FROM dba_advisor_parameters  
WHERE task_name = 'ADDM'  
AND parameter_name = 'ANALYSIS_SCOPE';
```

3. Delete old ADDM tasks:

```
DECLARE  
CURSOR c_tasks IS  
SELECT task_name  
FROM dba_advisor_tasks  
WHERE advisor_name = 'ADDM'  
AND execution_start < SYSDATE - 30;  
BEGIN  
FOR rec IN c_tasks LOOP  
DBMS_ADVISOR.DELETE_TASK(rec.task_name);  
END LOOP;  
END;  
/
```

4. ADDM Comparison Report

```
-- Compare two ADDM analyses  
SELECT DBMS_ADVISOR.GET_TASK_REPORT(  
  '&task1_name', 'TEXT', 'COMPARE', '&task2_name'  
) FROM dual;
```

ADDM Analysis Tips:

Impact > 50%: Critical issue

SQL issues: Top finding usually

Configuration issues: Parameter recommendations

Compare reports: Before/after changes

Schedule regular: Weekly ADDM analysis

Combine with AWR: ADDM for analysis, AWR for details

Most useful: Run ADDM after performance issues to get automatic root cause analysis.





Scan to open Script Bank

