

Oracle® Retail Data Store Implementation Guide



G44738-01
October 2025



This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

1 Implementation Overview

Schema Architecture	2
Audit and Delete Tracking	2
APEX Workspaces	2
Auxiliary Workspaces	3
Usage and Access Control	3
Privilege Model	4
Developer Access in RDS	4
Database Developer Privileges	4
Restrictive Custom Privilege Models	4
Report Developer Privileges (Oracle Analytics Server)	5
Report Consumer Privileges (Oracle Analytics Server)	5
Oracle Analytics Server Roles	5
Supported Cloud Services	5

2 Typical Implementation Events

3 Customer Responsibilities

Performance and Scalability	1
Resource Monitoring	1
Maintenance Preparedness	1
Pre-Maintenance Responsibilities	1
Post-Maintenance Responsibilities	2
Extension Management	2
Email Setup	2
Communication with Oracle	2

4 Getting Started

Step 1: Setting Up the Workspace Administrators	1
Prerequisites	1
Steps	1

Step 2: Setup RDS Database Operations Console Access	2
Prerequisites	2
Steps	2
APEX User Management	2

5 Oracle Analytics Server

Determining Your Tenant Name	1
Prerequisites	1
Steps to Identify the Tenant Name	1
Setup Access to OAS	1
Prerequisites	2
Steps to Assign a User to an OAS Group	2

6 Getting Started with Extensions

Overview	1
Access to Replicated Data	1
Custom Schemas and Workspaces	1
Primary Custom Schema	1
Auxiliary Custom Schemas (Per Cloud Service)	1
Auxiliary APEX Workspaces (RDS_CUSTOM_1, RDS_CUSTOM_2, RDS_CUSTOM_3)	2
Summary: Schema and Workspace Types	2
Schema Access and Grants	2
Grants to Custom Objects	2
Grants to Replicated Objects	3
Grants to Auxiliary Workspace Schema	3
Summary Table	3
Synonym Management	4
Create Synonyms	4
Drop Synonyms	4
Cleanup Synonyms	4
Tools for Extension	5
Oracle APEX	5
Oracle REST Data Services (ORDS)	5
Object Storage and DBMS_CLOUD	5
Environment Considerations	5
Prerequisites	6
Accessing the APEX UI	6

7	Obtaining ORDS Service Credentials	
	Generating an ORDS Access Token	3
	Generating an Access Token Using cURL	3
	Generating an Access Token Using POSTMAN	4
8	Accessing Object Store	
	Overview	1
	Obtaining a Pre-Authenticated Request (PAR) URL	1
	Constructing an Object Storage Object URL	2
	Obtaining Object Storage Credentials	4
9	ORDS RESTful Services	
	Overview	1
	Implementing a RESTful Service in APEX	1
	Invoking a RESTful Service from POSTMAN	4
10	ORDS PRE-HOOK	
11	Using RDS to Build a Data Producing Service	
	Overview	1
	GET Services	1
	POST Services	2
	Long Responses	3
12	Using RDS to Build a Data Consuming Service	
13	Exporting Data to Object Storage	
	Prerequisites	1
	Exporting Data Using a PAR	1
	Exporting Data with a Credential	2
14	Importing Data from Object Storage	
	Prerequisites	1
	Importing Data Using a PAR	1

15 DBMS Scheduler Jobs

16 Retail Home Integrations

In Context Launch of an APEX App	2
Launching APEX Apps from Retail Home	3

17 Invoking External Services

18 Retail DB Ops Console

Home	1
AWR Reports	2
Search Generated AWR Reports	2
Generate AWR Custom Reports	4
Top SQL	4
DBMS Jobs	6
Database Metrics	8
Application Properties	16
Session Management	17
Session Management API	21
Package Overview	22
Procedure and Function Details	22
LIST_RDS_SESSIONS	22
KILL_RDS_SESSION (SID and SERIAL)	22
KILL_RDS_SESSION (AUDSID)	22

19 Sending Email from APEX

20 Monitoring Replication Lag

21 Notification-Based Monitoring

Setting up the Notification Type	1
Implementing a RESTful Service in RDS	1
Setting up the POM Job	6

22 Storage and CPU Usage

23 Version Updates

24 Additional Notes

Not Supported/Available in RDS	1
APEX Roles and Privileges Not Supported	1
APEX Workspace Administration Disabled	1
GoldenGate Source/Target Use Not Supported	1
PWA-Enabled APEX Apps Not Supported	1
No Support for SFTP	1
Preconfigured Schema and Workspace Model	1
Immutable Privilege Model for Replicated Objects	2
OAuth Scope Enforcement Not Supported	2
Limited in RDS Environment	2
Service-Initiated Queries Limited to 300 Seconds	2
Service Source Length Limited to 4000 Characters	2
Only OCI SMTP Server Supported	2
APEX Limitations in Autonomous Database	2
Workarounds	2
Explain Plan via APEX UI Requires Workaround	3
Exporting Query Results in APEX	3
Enhancing OAuth Security with ORDS Prehook	4
Retrieving RDS Outbound IP Address	4
Internal Use Only	4
Reserved Application ID Range	4
GoldenGate Metadata Fields Are Not CDC	4

A Query Tuning

Introduction	A-1
How SQL Executes - A Conceptual Overview	A-1

Relational Databases: What You Need to Know	A-1
A Mental Model You Can Use	A-2
Where Problems Show Up	A-2
Start Here: Performance Triage	A-2
Next Step	A-3
Baselining Your Query	A-3
Why Baselining Matters	A-3
Baseline Tuning	A-3
What to Capture	A-3
When to Baseline	A-4
Baseline Template (Example)	A-4
Baseline SQL Script (for Recent Performance Snapshot)	A-5
Output Columns	A-5
Baselining Procedure	A-5
Step 1: Create a Tracking Table	A-5
Step 2: Create the Baselining Procedure	A-6
Step 3: Example Call	A-7
Optional: Automate It Nightly for Key Queries	A-7
Step 1: Create a Tracking Table	A-7
Step 2: Create a Batch Baselining Procedure	A-7
Step 3: Optional: Schedule the Job (DBMS_SCHEDULER)	A-7
Is My Query Slow?	A-8
Start with your Requirements	A-8
How to Think About "Fast Enough"	A-8
Measuring Execution Time	A-9
Don't Tune Without a Target	A-9
Diagnosing a Slow Query	A-9
Step 1: Get the SQL_ID	A-9
Step 2: Get the Execution Plan	A-9
Step 3: Ask These Questions	A-10
Step 4: Check for Waits	A-10
Step 5: Cross Check Stats	A-11
Step 6: Consider Plan History	A-12
Diagnosing System Wide Performance Issues	A-12
Step 1: Use AWR Reports to Understand System Load	A-12
Step 2: Use Database Metrics (DB Ops Console)	A-18
Step 3: Triage with "Top SQL"	A-18
Check DBMS Jobs (If Relevant)	A-18
Step 5: Session Management (Admin Role)	A-19
Understanding Plan Instability	A-19
What is Plan Instability?	A-19
Why Plans Change	A-20

How to Detect Plan Instability	A-20
When Plan Changes are Bad	A-20
What To Do	A-21

B Using FTS: An Example

Overview	B-1
Functions and Procedures	B-1
get_idcs_token	B-1
generate_par	B-1
put_object_in_store	B-2
export_query_to_blob	B-2
Example Usage	B-2
Error Handling	B-2
Securely Obtaining an IDCS Token	B-3
Overview	B-3
Security Considerations	B-3
Using the get_idcs_token_for_planning_app Function	B-3
Example Usage	B-3
Using the export_query_to_object_store Procedure	B-3
Prerequisites	B-3
Usage	B-4
Customization	B-4
Example	B-4
get_idcs_token	B-4
generate_par	B-5
put_object_in_store	B-6
export_query_to_blob	B-7

C Transferring Table Data

Assumptions	C-1
Step 1: Export Tables from Source ADW	C-1
Example PL/SQL Block	C-1
Step 2: Import Tables to Destination ADW	C-2
Example PL/SQL Block	C-2
Notes	C-3
Optional Enhancements	C-3

D Data Access Schema (DAS) vs. Retail Data Store (RDS)

Why, What, How	D-1
What is the Same	D-1
What is Different	D-1
Considerations When Moving from DAS to RDS	D-2
Key Takeaways	D-3

E CPU Utilization Metric Tile Example

Overview	E-1
Function: get_cpu_count	E-1
Purpose	E-1
Error Handling	E-1
Function: get_rh_chart_data_json	E-2
Purpose	E-2
Key Features	E-2
Design Note	E-2
Function: get_rh_summary_json	E-3
Purpose	E-3
Output Format	E-3
Use Case	E-3
Function: get_rh_cpu_utilization_json	E-4
Purpose	E-4
Design Pattern	E-4
Why FORMAT JSON is Used	E-4
Anonymous Block (REST Service Source)	E-4
Purpose	E-4
Key Elements	E-5
Context	E-5
Example REST Service Response	E-5
Summary	E-6

Preface

This guide describes the administration tasks for Oracle Retail Data Store.

Audience

This guide is intended for administrators, and describes the administration tasks for Oracle Retail Data Store.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

<https://support.oracle.com>

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Oracle Help Center (docs.oracle.com)

Oracle Retail Product documentation is available on the following website <https://docs.oracle.com/en/industries/retail/html>

Comments and Suggestions

Please give us feedback about Oracle Retail Help and Guides. You can send an e-mail to: retail-doc_us@oracle.com

Oracle Retail Cloud Services and Business Agility

Oracle Retail Merchandising Cloud Services is hosted in the Oracle Cloud with the security features inherent to Oracle technology and a robust data center classification, providing significant uptime. The Oracle Cloud team is responsible for installing, monitoring, patching, and upgrading retail software.

Included in the service is continuous technical support, access to software feature enhancements, hardware upgrades, and disaster recovery. The Cloud Service model helps to free customer IT resources from the need to perform these tasks, giving retailers greater

business agility to respond to changing technologies and to perform more value-added tasks focused on business processes and innovation.

Oracle Retail Software Cloud Service is acquired exclusively through a subscription service (SaaS) model. This shifts funding from a capital investment in software to an operational expense. Subscription-based pricing for retail applications offers flexibility and cost effectiveness.

1

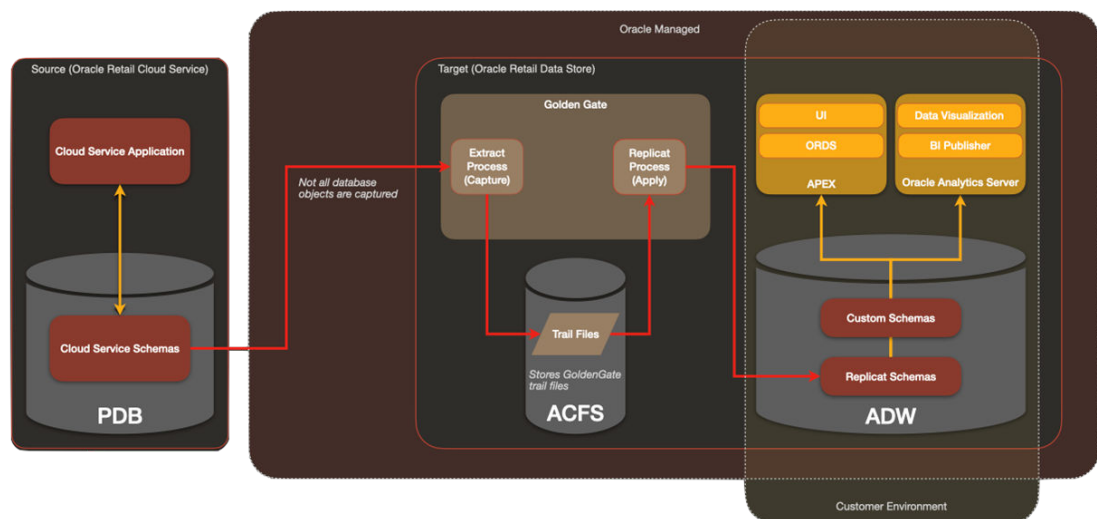
Implementation Overview

Oracle Retail Data Store (RDS) is a centralized data infrastructure that consolidates data from one or more Oracle Retail Cloud Services, depending on your subscription. It enables the customization of retail data without modifying the underlying cloud services. See "Data Replication to RDS via GoldenGate".

Your extensions—including database objects, web services, and APEX applications—are built on top of the replicated data, which serves as the foundation for your enhancements. This consolidation is achieved by replicating data from multiple cloud services into a single RDS instance using Oracle's Golden Gate. Not every database object in the source is replicated. Each Oracle Retail Cloud Service App team selects database objects for replication-based business value and suitability for replication.

This Implementation Guide provides an overview of RDS and explains how you can use it to extend one or more cloud services.

Figure 1-1 Data Replication to RDS through GoldenGate



- **PDB** - Pluggable Data Base. The source applications in the RDBU that will be replicating to RDS store their data in pluggable database instances.
- **ACFS** - Advanced Cluster File System. A file system used internally by GoldenGate to store the trail files that hold data replication information.
- **ORDS** - Oracle Rest Data Services. An Oracle tool that allows customers to create web services connected directly to data in an Oracle database. RDS customers will use this to create web services to access their custom data.
- **APEX** - Application Express. An Oracle tool that allows customers to create UI-based applications connected directly to data in an Oracle database. RDS customers will use this to create applications that operate on their custom data.

- **ADW** - Autonomous Data Warehouse. An Oracle Autonomous Database offering that is tailored toward data warehousing use cases. RDS stores its replicated data and the customer's custom data here.

Schema Architecture

The schema architecture of RDS consists of read-only replicate schemas that mirror source cloud service data and read-write custom schemas for extensions.

- Replicated data from each cloud service is structured into a primary replicate schema and, if required, one or more auxiliary replicate schemas (*only some cloud services require auxiliary schemas*).
- Replicate schemas (both primary and auxiliary) are strictly read-only and mirror the data in the source cloud service.
- No modifications can be made to replicate schemas in RDS. This includes adding indexes, partitions, or other database objects. It also includes changing history retention rules. RDS will mirror the source cloud service history retention rules.

Each replicate schema (both primary and auxiliary) is linked to a custom schema, which is read-write.

- Each custom schema is granted SELECT privileges on views in all replicate schemas, meaning it can query data from its associated replicate schema and from other cloud services.
- All extensions reside in the custom schema.

Unlike replicate views, custom database objects are not accessible by default to other schemas. The schema owner must explicitly grant access privileges before they can be accessed by other schemas.

Audit and Delete Tracking

By default, basic replication ensures that the source and target datasets remain identical. When a record is deleted from the source, it is deleted from the target. When a record is updated, the previous state is lost, because the target reflects only the most recent changes.

For a subset of RDS-supported products, audit and delete tracking tables provide visibility into historical changes.

- **Audit Tracking:** If a table is enabled for audit tracking, every INSERT, UPDATE, or DELETE operation inserts a new record into an RDS-only audit tracking table, maintaining a historical log of all changes.
- **Delete Tracking:** If a table is enabled for delete tracking, whenever a record is deleted from the source, a new record is inserted into an RDS-only delete tracking table instead of being permanently removed.

At this time, only the Merchandising Foundation Cloud Service supports audit and delete tracking.

APEX Workspaces

Each primary custom schema is associated with an APEX workspace.

- When logging into the APEX UI, you will select a workspace to work within.

- Your APEX UI session will be connected to the primary custom schema associated with that workspace, providing access to custom objects and replicated views within the RDS environment.

Auxiliary Workspaces

RDS includes three auxiliary APEX workspaces:

- RDS_CUSTOM_1
- RDS_CUSTOM_2
- RDS_CUSTOM_3

These workspaces are isolated environments that do not inherently contain database objects or have access to other schemas. They are intended for sandboxing code, segmenting development efforts, and refining security policies.

Usage and Access Control

Default State:

- No direct access to any schema objects.
- No preloaded content or privileges.

Granting Access to Other Schema Objects:

- Access to specific tables, views, procedures, or packages can be selectively granted from other schemas.

Examples:

```
GRANT SELECT ON schema_name.table_name TO rds_custom_1;  
GRANT EXECUTE ON schema_name.procedure_name TO rds_custom_1;
```

- Refining Security and Limiting Access:
 - Access can be restricted to only necessary objects.
 - Views can be used to control data exposure by creating limited result sets.


```
CREATE VIEW schema_name.view_name AS SELECT column1, column2 FROM  
schema_name.table_name WHERE condition; GRANT SELECT ON  
schema_name.view_name TO  
rds_custom_1;
```
- Security Considerations
 - Least Privilege Principle: Workspaces should only be granted access to necessary objects.
 - No Direct DML Access: By default, these workspaces should not have INSERT, UPDATE, or DELETE permissions unless explicitly required.
- Potential Use Cases
 - Restricted Access: Providing read-only access to custom database objects.
 - Sandboxing Code: Developers can test PL/SQL code or APIs within a controlled environment.

- Segmentation of Data Access: Access can be granted selectively to different workspaces for different use cases. Privilege Model.
- Security Refinement: Restrict access to specific views instead of direct table access to improve data governance.

Privilege Model

RDS is a SaaS environment that enables the extension of Oracle Retail Cloud Services. RDS follows a fixed privilege and architecture model that cannot be customized. More specifically, its privilege model follows the least privilege principle and cannot be modified.

The RDS privilege model is designed to provide developers with the necessary privileges to:

- Develop secure applications and services in a controlled environment.
- Perform diagnostic activities in a live environment when troubleshooting issues.

The same tools and privileges are used for both development and diagnostics. However, while development occurs in a controlled setting, diagnostic activities are performed in a live environment where direct development is not expected.

Developer Access in RDS

There are three broad categories of developers based on how they interact with RDS:

1. Database Developers – Access RDS through the APEX UI or a private endpoint, typically working with PL/SQL and database logic.
2. Report Developers – Access RDS through Oracle Analytics Server (OAS) and focus on authoring reports. Report developers have a role of *ContentAuthor.
3. Report Consumers - Access RDS through OAS and focus on viewing reports. Report consumers have a role of *Consumer.

See the *RDS Security Guide* for details on user roles.

The privilege models employed for each user category differ in meaningful ways but never exceed the baseline RDS privilege model.

Database Developer Privileges

- Any database developer with access to a primary custom schema (*excluding* *RDS_CUSTOM_1*, *RDS_CUSTOM_2*, and *RDS_CUSTOM_3*) has SELECT privileges on all replicated data.
- By default, database developers do not have access to database objects in other custom schemas.

Restrictive Custom Privilege Models

The schemas *RDS_CUSTOM_1*, *RDS_CUSTOM_2*, and *RDS_CUSTOM_3* begin with no privileges to other schemas. These schemas support a more restrictive privilege model than standard RDS. Conceptually, they serve as Custom Privilege Models 1, 2, and 3, where access can be granted incrementally.

Report Developer Privileges (Oracle Analytics Server)

The Oracle Analytics Server (OAS) as a development environment is used for constructing reports.

Supported Cloud Services

- Report developers have SELECT access to all tables and views in RDS, including:
 - Replicated data
 - Custom tables and views
- Report developers can be restricted to specific tools:
 - Data Visualization
 - BI Publisher
- Report developers can both create reports and set report permissions, but those permissions encompass either everybody or nobody.

Report Consumer Privileges (Oracle Analytics Server)

Report consumers have view-only access to all reports where viewing by others is permitted. Like report developers, consumers can be restricted to specific tools.

Oracle Analytics Server Roles

Oracle Analytics Server roles are limited to those provided. You cannot create new roles. Nor can you modify or delete existing roles.

Supported Cloud Services

The supported cloud services are listed in the "Supported Cloud Services" table. Each listed service has a:

- Cloud service name: the name it is commonly known as
- A workspace name: the name seen in the APEX UI launch screen
- The primary custom schema name, the schema within which extensions reside
- The primary replication schema, the schema in which replicated objects reside
- Auxiliary replication schema, if any

Table 1-1 Supported Cloud Services

Cloud Service Application	Workspace Name	Primary Custom Schema	Primary Replication Schema	Auxiliary Schema
Merchandising Foundation	MFCS	MFCS_RDS_CUSTOM	MFCS_RDS	
Customer Engagement	CE	CE_RDS_CUSTOM	CE_RDS	
Store Inventory Operations	SIOCS	SIOCS_RDS_CUSTOM	SIOCS_RDS	

Table 1-1 (Cont.) Supported Cloud Services

Cloud Service Application	Workspace Name	Primary Custom Schema	Primary Replication Schema	Auxiliary Schema
Order Broker	OB	OB_RDS_CUSTOM	OB_RDS	
XOffice	XO	XO_RDS_CUSTOM	XO_RDS	XO_XADMIN
Supplier Evaluation	SE	SE_RDS_CUSTOM	SE_RDS	
Brand Compliance	BC	BC_RDS_CUSTOM	BC_RDS	
Retail Integration	RICS	RICS_RDS_CUSTOM	RICS_RDS	
Supply Chain Hub	RSCH	RSCH_RDS_CUSTOM	RSCH_RDS	
Order Administration	OM	OM_RDS_CUSTOM	OM_RDS	
N/A	RDS CUSTOM 1	RDS_CUSTOM_1		
N/A	RDS CUSTOM 2	RDS_CUSTOM_2		
N/A	RDS CUSTOM 3	RDS_CUSTOM_3		

2

Typical Implementation Events

In any implementation including RDS, there are many steps along the way before a system is running. In the discussion below, cloud services, such as Merchandising Foundation, are the source cloud services and RDS is the target.

- Provisioning
 - Provisioning includes the installation of the RDS Cloud Service including initial infrastructure required. This includes an ADW instance with schemas available for replication and extension, ORDS workspaces, and integration into Oracle Retail Home for display of usage metrics.
- Initial Data Load through Data Pump
 - The next step is creating an initial data load into RDS from the source cloud service using Oracle Data Pump tools. This step is taken by Oracle when the retailer indicates they are ready to move forward.
 - A prerequisite to this step is that the source cloud service must have data ready to be replicated. The initial load may be an involved process depending on the cloud service in question. Refer to documentation for the source cloud service.
 - The result of this step is that a baseline set of data has been replicated from the source cloud service to the RDS read-only schema.
- GoldenGate Hub Configuration
 - A GoldenGate Hub instance is configured to replicate data from the source cloud service's database to the RDS read-only schema.
 - This is done by Oracle when the retailer indicates they are ready to move forward.
 - The result of this is that the GoldenGate Hub is running and performing active replication from the source cloud service's database.
- Extension
 - In this step, the retailer uses the tools that are part of RDS to build the custom extensions they need.

3

Customer Responsibilities

The primary custom schema in RDS is a powerful, writable environment that enables you to extend Oracle Retail Cloud Services. With this flexibility comes responsibility. As the schema owner, you are accountable for the performance, reliability, and behavior of your extensions.

This chapter outlines the key responsibilities that ensure your extensions remain performant, maintainable, and compliant with the RDS operational model.

Performance and Scalability

You are responsible for ensuring that all custom services and queries operate within acceptable performance boundaries.

- **Query Duration Limits:** All operations must complete in **300 seconds or less**. Ideal baseline performance should be under **150 seconds**.
- **Traffic Intensity:** Estimate request arrival rates and average processing time to avoid overload.
- **Baseline Testing:** Collect timing metrics under representative workloads for all services and reports.
- **Scalability Awareness:** RDS has finite resources. Excessive CPU consumption or session usage can affect availability.

Resource Monitoring

You are expected to monitor and manage consumption of your schema's allocated resources.

- **Session Management:** Use the DB Ops Console to identify and terminate runaway, idle, or zombie sessions.
- **CPU and Storage:** Track utilization and proactively request increased capacity when needed.
- **Efficiency:** Optimize queries and background jobs to minimize resource usage.

Maintenance Preparedness

Custom objects may become **invalid** during RDS maintenance when underlying views or dependencies are dropped and recreated. Maintenance preparedness helps prevent downtime and ensures a smooth return to service.

Pre-Maintenance Responsibilities

1. **Notify Users:** Let users know in advance to reduce disruption and prevent long-running or orphaned sessions.
2. **Disable Custom Jobs:** Suspend scheduled jobs (for example, DBMS_SCHEDULER) to avoid interference during maintenance.

3. **Suspend External Orchestration:** Pause external workflows that target RDS to avoid unexpected connections.
4. **Terminate Active Sessions:** Ensure that no active sessions are holding locks on custom objects. Locked objects within dependency chains can cause recompilation failures.

Post-Maintenance Responsibilities

1. **Check for Lingering Sessions:** Ensure no sessions are preventing recompilation or consuming resources unnecessarily.
2. **Handle Invalid Objects:** Invalid custom objects may remain after maintenance due to:
 - a. **Long dependency chains**, where objects must be compiled in order.
 - b. **Locked objects**, which prevent successful recompilation.
 - c. **Timeouts**, if recompilation takes too long or encounters blocked dependencies.
 - d. You are responsible for resolving invalids in your schema. Refer to the [RDS Compile Invalids Reference Paper \(Doc ID 2899701.1\)](#) for instructions on resolving these issues.
3. **Notify Users:** Let users know that RDS is available again and safe to resume work.
4. **Re-enable Jobs and Workflows:** Restart any disabled jobs or paused orchestration processes.

Extension Management

Your primary custom schema is fully writable, and you are responsible for managing everything created within it.

- **Custom Object Ownership:** All custom tables, views, packages, and procedures reside in your schema.
- **Cross-Schema Access:** Other schemas do not have access to your objects unless explicitly granted.
- **Security and Governance:** Apply least privilege principles and use views to control access when necessary.
- **Resilience:** Design extensions to handle failure scenarios and resume cleanly after terminations.
- **Grants:** You are responsible for managing grants on replicated objects in RDS_CUSTOM_1/2/3. See [Grants to Auxiliary Workspace Schema](#).

Email Setup

To send email from your RDS ADW instance using the APEX_MAIL package, you must provision an OCI SMTP server. Only Oracle's OCI SMTP server is supported for sending email from RDS ADW.

Communication with Oracle

Proactive communication with Oracle Support helps ensure system stability and availability, especially during non-routine operations.

Examples of when to notify Oracle:

- **High-volume data events**, such as store rollouts or data migrations.
- **Custom retention strategies**, such as large-scale archiving.
- **Extended integration testing**, especially if it may impact system load or replication.

4

Getting Started

Step 1: Setting Up the Workspace Administrators

Before you can log in to any of the above APEX Workspaces, you must set up the login details for the administrator of each workspace.

Prerequisites

1. Access to the OCI Console
2. You have the Retail Home URL for each environment you wish to set up (for example, PROD and STG)
3. Access to Retail Home
4. The ability to create new users in OCI IAM

Steps

1. Log in to the OCI Console.
2. Create a default workspace administrator account from your OCI Console in OCI IAM for each subscribed cloud service using the primary schema names in [Table 1-1](#).
 - a. The Default Workspace Administrator account passwords and their lifecycle will then be managed by the customer in OCI IAM going forward.
 - b. There is no need to synchronize this user with APEX. The only requirement is that the usernames match.
 - c. For example, create a Workspace Administrator account in OCI IAM with the username MFCS_RDS_CUSTOM. Once you have created this account you can log in to the APEX MFCS Workspace using the MFCS_RDS_CUSTOM user ID.
 - d. These logins will work for all your environments (for example, the MFCS_RDS_CUSTOM login will grant you access to the MFCS workspace in both STAGE and PRODUCTION).
 - e. The default administrator logins are secure by default. They are, however, only intended to provide the initial access necessary to establish access for each workspace administrator.
3. Log in to the Default Workspace Administrator account on your OCI Console. You will have to logout first.
 - a. Verify the reachability of the workspace launch page for your environment using the Retail Home Application Navigator and selecting **RDS APEX/ORDS (RDS APEX/ORDS** is included in the Application Navigator by default).
 - b. Note that the default workspace administrator will only be able to reach one workspace.
4. Create one or more workspace administrators for this workspace (see [APEX User Management](#) below). One of those workspace administrators should be yourself.

- a. Once you have created your user administrators, logout of the APEX UI.
 - b. Login as yourself.
 - c. Disable the default administrator accounts in OCI IAM. You can always recreate the accounts in the future, if necessary.
5. Verify the reachability of the workspace launch page for your environment using the Retail Home Application Navigator and selecting RDS APEX/ORDS.
 6. Repeat steps 2-5 for each cloud service to which you have subscribed.
 7. Using your workspace administrator account set up workspace developer accounts, see [APEX User Management](#) below.

Step 2: Setup RDS Database Operations Console Access

Prerequisites

1. Access to the OCI Console
2. You have the Retail Home URL for each environment you wish to set up (for example, PROD and STG)
3. Access to Retail Home
4. The ability to assign users to groups in OCI IAM

Steps

1. Log in to the OCI Console.
2. Assign each workspace administrator with access to pre-production environments to the RDS_MANAGEMENT_ADMINISTRATOR_PREPROD group.
 - a. Verify reachability of the RDS DB Ops Console for your pre-production environment using the Retail Home Application Navigator and selecting **RDS DB Ops Console** (**RDS DB Ops Console** is available in the Retail Home Application Navigator by default).
 - b. Assign developers to the appropriate RDS_MANAGEMENT pre-production groups based on the level of access required. See the *Retail Data Store Security Guide* for additional details.
3. Assign each workspace administrator with access to production environments to the RDS_MANAGEMENT_ADMINISTRATOR group.
 - a. Verify reachability of the RDS DB Ops Console for your production environment using the Retail Home Application Navigator and selecting **RDS DB Ops Console**.
 - b. Assign developers to the appropriate RDS_MANAGEMENT production groups based on the level of access required. See the *Retail Data Store Security Guide* for additional details.

APEX User Management

For the purposes of this documentation, there are two types of APEX users, end users and development users. End users are users with access to the applications built with APEX. They will log into and use those applications but not be involved in their development or management. Development users, on the other hand, can create and manage the APEX

applications the end users use. Within this set of users, there are Developer and Workspace Administrator roles. From the Oracle APEX UI, developers can quickly create web apps including custom database objects, reports, forms, and RESTful services using a low code interface.

Note

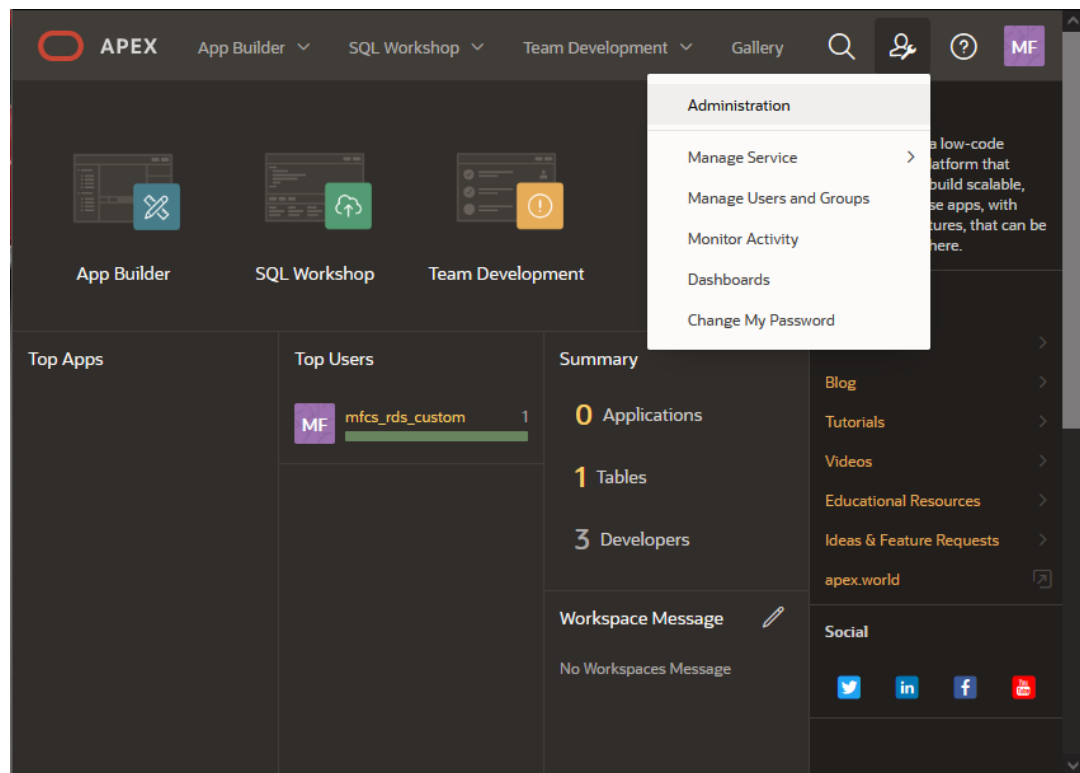
You will typically grant additional users workspace administrator permissions rather than continuing to use the default workspace administrator account.

This document will focus on managing Development users. End user authentication is managed by the Workspace Administrator, who can choose any supported form of authentication for the APEX applications developed. For details on supported models, please reference the *APEX App Builder User's Guide*, section 21.4 "Establishing User Identity Through Authentication."

Development user authentication is provided through integration with Oracle Cloud Infrastructure Identity and Access Management (OCI IAM). The APEX Workspaces provisioned for RDS are configured to use HTTP Header Variable authentication. For full details on this model, please refer to the *APEX App Builder User's Guide*, section 21.4.2.4 "HTTP Header Variable."

In most cases, teams will need to create additional development users to facilitate the development of APEX applications and REST endpoints. The Workspace Administrator account has the permissions to create additional Developer and Workspace Administrator users through the APEX UI. Any additional users created will need to follow the same pattern as the default user accounts. Create the users in APEX and create matching usernames in OCI IAM. Like the default Workspace Administrator accounts, these new accounts will have their passwords live in OCI IAM.

1. Log in to APEX using your workspace administrator account, such as MFCS_RDS_CUSTOM, which you previously created in OCI IAM.
2. From the APEX start page, access the Administration menu in the upper right corner and select the **Manage Users and Groups** option.



3. Click the **Create User** button within the User Management screen.
4. On the create user form, enter the **Username** and **Email**, which are identical to the OCI IAM user account you wish to add.
5. Under Account Privileges, select whether the **User is a workspace administrator** or the **User is a developer**.

If neither option is selected, then the user will not have the ability to create anything in APEX but may be able to access applications that are already created.

Account Privileges

Default Schema

RTLWSP01

?

Accessible Schemas (null for all)

?

User is a workspace administrator:

☐ Yes ☒ No

?

User is a developer:

☒ Yes ☐ No

?

App Builder Access

Yes

?

SQL Workshop Access

Yes

?

Team Development Access

No

?

Set Account Availability

Unlocked

?

6. For the **Password**, you may enter any value you wish. The APEX password is not used when authentication is managed by OCI IAM.
7. Ensure the option **Require Change of Password on First Use** is set to No, as we do not want APEX to manage the authentication.
8. Under Group Assignments, add one or more privileges to the user if they are a Developer or Administrator. When you are finished, click **Create User** at the top of the screen to add them to APEX.

For details on other user management activities in APEX, refer to the *APEX Administration Guide* chapter “Understanding Workspace Administration”.

5

Oracle Analytics Server

RDS is provisioned with Oracle Analytics Server (OAS), offering **Data Visualization** and **BI Publisher** (also known as **Analytics Publisher** or **BIP**) capabilities. These tools are accessible through the **Retail Home Application Navigator** by selecting either **Data Visualization** or **Analytics Publisher**.

Access is protected by the same OCI IAM instance used throughout Retail Data Store and supports **Single Sign-On (SSO)**. Further access control for both tools is managed through **OCI Groups**.

This section guides you through the steps required to configure access to OAS.

For more guidance on using Data Visualization and BI Publisher, see [Visualizing Data in OAS](#).

Determining Your Tenant Name

Prerequisites

- You have the **Retail Home URL** for each environment you plan to configure (for example, PROD, STG).
- You can access **Retail Home** through a supported browser.

Steps to Identify the Tenant Name

1. Navigate to **Retail Home** for the desired environment using the URL provided by your administrator.
2. In the **Application Navigator** in the left-hand panel, click **Data Visualization** or **Analytics Publisher**.
3. Observe the **URL** in your browser's address bar. It will follow a structure similar to: `https://<hostname>/<tenant-name>/...`
4. Identify the **tenant name**, which follows the hostname. For example, if the URL is: `https://retail.example.com/acme/...` then the tenant name is `acme`.
5. You will use this tenant name as a prefix when assigning users to OAS access groups.

Setup Access to OAS

OAS access groups have the following form:

Group Name	Privilege
<tenante-name>-DVConsumer	Data Visualization report viewing
<tenant-name>-DVContentAuthor	Data Visualization report creation and maintenance
<tenant-name>-BIConsumer	Analytics Publisher report viewing
<tenant-name>-BIContentAuthor	Analytics Publisher report creation and maintenance

Note

You cannot create additional roles for Data Visualization or Analytics Publisher. Nor can you alter the privileges granted to these roles.

Prerequisites

- You have permission to **manage IAM users and groups** in the OCI tenancy.
- You know the **tenant name** (for example, acme).
- You know which users need access and their intended roles (viewer or content author; Data Visualization or BI Publisher).

Steps to Assign a User to an OAS Group

1. **Sign in to the OCI Console** for your tenancy.
Use the identity domain that Retail Data Store is deployed in.
2. In the **left navigation menu**, go to: **Identity & Security -> Domains**.
3. Click the domain where your Retail Data Store is deployed. If you're unsure which one to use, ask your tenancy administrator.
4. Click the **User Management** menu item. **Groups** will be found at the bottom of the page.
5. Search for and select the group you want to assign the user to. Group names follow this pattern:
 - <tenant-name>-DVConsumer
 - <tenant-name>-DVContentAuthor
 - <tenant-name>-BIConsumer
 - <tenant-name>-BIContentAuthor
6. In the group details, click the **Users** menu item.
7. Search for users by name or email. Select the appropriate user by checking the box next to their name, then click **Assign user to group**.
8. Repeat for additional users and groups as needed.

Assignment of OAS roles (OCI Groups) may take 15 minutes or more to propagate.

6

Getting Started with Extensions

Overview

Oracle Retail Data Store (RDS) enables you to extend Oracle Retail Cloud Services by developing custom applications, RESTful APIs, and database logic within a controlled, cloud-based environment. These extensions are built in writable schemas, separate from the read-only data replicated from the source systems.

This chapter summarizes the structures, tools, and environment characteristics that shape how you build and deploy extensions in RDS.

Access to Replicated Data

Each cloud service replicates selected operational data into read-only replicate schemas using Oracle GoldenGate. These schemas expose views only; tables are not accessible or modifiable. Replicated objects are accessed using a fully qualified name.

For more details on replication design and schema structure, see [Chapter 1](#).

Custom Schemas and Workspaces

All RDS extensions are implemented within **writable schemas**. These schemas are distinct from the read-only replicate schemas and are the foundation for all customer-developed applications, services, and custom logic.

There are **three types of writable schemas** used in RDS:

Primary Custom Schema

Each cloud service that supports RDS includes a **primary custom schema**:

- It is **writable** and **associated with an APEX workspace**.
- Customers use it to create APEX applications, RESTful services, PL/SQL packages, and custom tables or views.
- It is granted **SELECT privileges** on one or more read-only replicate schemas, allowing access to cloud service data through views.
- All APEX development takes place within this schema's workspace.

This is the **default environment** for customer extensions.

Auxiliary Custom Schemas (Per Cloud Service)

Some cloud services provide one or more auxiliary custom schemas, such as X0_XADMIN or RICS_BDI1:

- These are read-only and not associated with an APEX workspace.

- They are provided for organizational or functional separation within a specific cloud service.
- They do not have access to replicate views or other schemas by default.
- Customers must explicitly grant privileges from their primary custom schema to access or use these schemas.

These schemas are useful for modular development or isolating specific components of a solution.

Auxiliary APEX Workspaces (RDS_CUSTOM_1, RDS_CUSTOM_2, RDS_CUSTOM_3)

In addition to service-specific schemas, RDS provides three general-purpose writable schemas:

- These are writable and associated with APEX workspaces, but not tied to any specific cloud service.
- They start with no privileges—access to replicate views or other schemas must be granted manually.
- These schemas are ideal for:
 - Sandboxing or experimenting with logic.
 - Restricting access to certain users or development efforts.
 - Refining security by isolating sensitive processes.

They allow advanced users to segment their development environment as needed.

Summary: Schema and Workspace Types

Type	Writable	APEX Workspace	Tied to Cloud Service	Notes
Primary Custom Schema	Yes	Yes	Yes	Main development environment. SELECT access to all replicated schema
Auxiliary Custom Schema	Yes	No	Yes	SELECT access to all replicated schema
Auxiliary APEX Workspaces (RDS_CUSTOM_1/2/3)	Yes	Yes	No	General-purpose; no privileges by default

Schema Access and Grants

RDS supports two distinct grant scenarios for enabling access across schemas:

Grants to Custom Objects

Custom tables, views, and packages created in a primary or auxiliary custom schema can be shared with other schemas using standard GRANT statements.

- These grants must be managed by the object owner.

- Use cases:
 - Sharing a lookup table with a REST service defined in another schema.
 - Allowing a secondary schema to run a reporting package.

Example:

```
GRANT SELECT ON \<a-custom-mfcs-table\> TO xocs\_rds\_custom;
```

These grants are not recursive—the grantee cannot grant access to others unless explicitly given the `WITH GRANT OPTION`.

Grants to Replicated Objects

Access to views in replicate schemas is initially granted to the primary and auxiliary custom schema with the `WITH GRANT OPTION`. This means the primary schema can grant access to replicated views to auxiliary workspace schemas; that is, `RDS_CUSTOM_1`, `RDS_CUSTOM_2`, or `RDS_CUSTOM_3`.

- This is how auxiliary APEX workspaces get access to replicated data.
- No need for intervention from Oracle or additional metadata configuration.

Example:

```
GRANT SELECT ON \<an-xocs-view\> TO rds\_custom\_1;
```

Because of the `WITH GRANT OPTION`, any schema that has access to a replicated view can grant it to another schema. This provides flexibility for customers managing multi-schema solutions or enforcing separation of concerns.

Grants to Auxiliary Workspace Schema

You should encapsulate grants for each Auxiliary workspace in a procedure, one per auxiliary workspace. The owner of the procedure should be one of the product workspace schemas (for example, `MFCS`), which schema owns which grant procedure is your decision.

Execute permission for each grant procedure should be granted to the appropriate Auxiliary workspace. The procedure should be declared with "authid definer". The later should allow the procedure to be called from `RDS_CUSTOM_1/2/3` and achieve the desired result.

Ideally, the customer invokes the procedure periodically using a scheduled job, but it can be invoked in an ad hoc manner. Although we are working on a solution to avoid dropping views which then drops grants, there is no guarantee that we will not drop a replicated view in the future.

Summary Table

Grant Type	Object Location	Granting Schema	Notes
Custom object	Primary/Auxiliary Custom / Auxiliary workspace Schema	Object owner	Explicit grant needed
Replicated object	Replicate Schema	Any schema with grant option (primary or auxiliary schema)	Grants allowed due to <code>WITH GRANT OPTION</code>

Synonym Management

A synonym management API is provided to simplify access to replicated objects. Synonyms allow replicated objects to be referenced without needing to use a fully qualified name. The API has three procedures: one for creating synonyms; one for dropping synonyms; and lastly, one for cleaning up synonyms.

Note

Synonym management only manages synonyms for replicated objects. All API procedures reside in the `RDS_OBJECT_MGMT` package. For the sake of brevity, the package name is not shown in the API description below. Nonetheless, in order to access the procedures described, you will need to use the full qualified procedure name (for example, `RDS_OBJECT_MGMT.create_synonyms`).

Create Synonyms

The `create_synonyms` procedure creates synonyms for all replicated views in a selected source schema.

Parameters:

<code>I_source_schema</code>	<code>IN VARCHAR2</code>	- Source schema name.
<code>I_target_schema</code>	<code>IN VARCHAR2</code>	- Target schema name, if NULL defaults to <code>current\user</code> .
<code>I_identifier</code>	<code>IN VARCHAR2</code>	- Optional prefix for synonym names.
<code>O_status</code>	<code>OUT VARCHAR2</code>	- Return status.

Drop Synonyms

The `drop_synonyms` will drop all synonyms associated with a selected source schema.

Parameters:

<code>I_source_schema</code>	<code>IN VARCHAR2</code>	- Source schema name.
<code>I_target_schema</code>	<code>IN VARCHAR2</code>	- Target schema name, if NULL defaults to <code>current_user</code> .
<code>I_identifier</code>	<code>IN VARCHAR2</code>	- Optional prefix for synonym names.
<code>O_status</code>	<code>OUT VARCHAR2</code>	- Return status.

Cleanup Synonyms

The `cleanup_synonyms` procedure identifies and drops invalid synonyms (pointing to nonexistent objects) for a selected source schema.

Parameters:

<code>I_source_schema</code>	<code>IN VARCHAR2</code>	- Source schema name.
<code>I_target_schema</code>	<code>IN VARCHAR2</code>	- Target schema name, if NULL defaults to

```
current_user.  
0_status          OUT VARCHAR2 - Return status.
```

Tools for Extension

Oracle APEX

APEX provides a low-code platform for building applications within any custom schema.

- Each APEX workspace is mapped one-to-one with a primary custom schema.
- Developers build forms, reports, dashboards, and services through the browser.
- SQL Workshop provides tools to manage schema objects and write PL/SQL.
- Database objects can be accessed in replicated and auxiliary schema using fully qualified names.

Oracle REST Data Services (ORDS)

ORDS allows you to expose data and logic as a RESTful services.

- Services execute within your writable schema and can access both custom and replicate views.
- Methods supported: GET, POST, PUT, and DELETE.
- Designed for data-producing (queries, exports) and data-consuming (inserts, updates) services.

Services must return within 300 seconds to avoid timeouts. Bulk data flows should be offloaded to object storage.

Object Storage and DBMS_CLOUD

Bulk data movement is handled through Oracle Object Storage.

- Export using `DBMS_CLOUD.EXPORT_DATA` and a writable PAR or credentialed URI.
- Import using `DBMS_CLOUD.COPY_DATA`.

Environment Considerations

RDS is a SaaS platform with a distinct development model. While powerful, it comes with important constraints.

- APEX-Centric Development: Most development is performed through the browser. There is no access to SQL*Developer, SQL*Loader, or shell-level tools.
- No Native DevOps Tooling: RDS does not include version control, staging environments, or deployment pipelines. Teams must manage the code lifecycle externally.
- Low-Code + PL/SQL: Applications and APIs are created through APEX and SQL Workshop. Custom code can be hundreds or thousands of lines, but large-scale engineering efforts should be staged externally.
- Strict Object Boundaries: Only replicate views are accessible across schemas. Custom objects must be explicitly shared using GRANT statements.

- Single-Schema Workspaces: Each APEX workspace provides access to one primary custom schema only.
- Resource Constraints: Session, CPU, and runtime limits apply. Resource-intensive tasks should be optimized or offloaded to batch processes.

Prerequisites

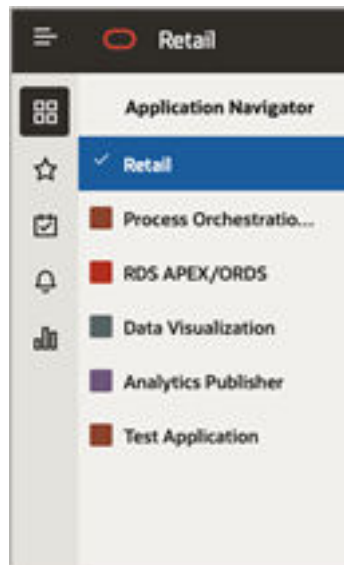
To implement any meaningful extensions, you will need to meet the prerequisites listed below. Furthermore, the examples in this chapter can be replicated in your RDS environment. Replicating the examples will both help you understand the development context and provide assurance that the prerequisites have been met prior to starting implementation.

- A Retail Home instance. Contact your RDS System Administrator for details – Oracle Support does not provide this information.
- An IDCS Authorization Server host. Contact your RDS System Administrator for details – Oracle Support does not provide this information.
- An Oracle Cloud account. Contact your RDS System Administrator for details on setting up your Oracle Cloud account.
- A working knowledge of Oracle Retail Home.
- A working knowledge of the Oracle Cloud Console (on the web search for *Using the Oracle Cloud Console* for the latest documentation).
- Access to an APEX workspace within an RDS tenant (see user management above).
- Access to a suitable Object Storage service.
- Access to a suitable object storage bucket. RDS does not automatically come with a customer accessible object storage bucket. Provisioning an object storage bucket for use with RDS is a customer responsibility. *Bear in mind, FTS, when available, will not be able to produce usable writable PARs for DBMS_CLOUD.EXPORT_DATA (EXPORT_DATA is expecting a prefix or bucket URI, not an object URI). Readable PARs generated by FTS for importing data into RDS, however, are usable with DBMS_CLOUD.COPY_DATA.*

Accessing the APEX UI

You will need access to Retail Home endpoint.

APEX is a browser-based application. You access APEX by navigating to the Retail Home Application Navigator and selecting **RDS APEX/ORDS** (RDS APEX/ORDS is included in the Application Navigator by default).



It is the responsibility of RDS workspace admin to create development user accounts for each user requiring access to one or more APEX workspaces. See the [APEX User Management](#) section above for additional details.

Before proceeding:

1. Verify access to Retail Home
2. Verify access to the relevant APEX workspaces.

7

Obtaining ORDS Service Credentials

ORDS services use OAUTH 2 for authentication. All services are authenticated. What this means in practice is that a short-lived token is used for authentication. That token is generated using a well-known service, which authenticates using basic auth. The basic auth credentials (that is, client id and client secret) are obtained from Retail Home.

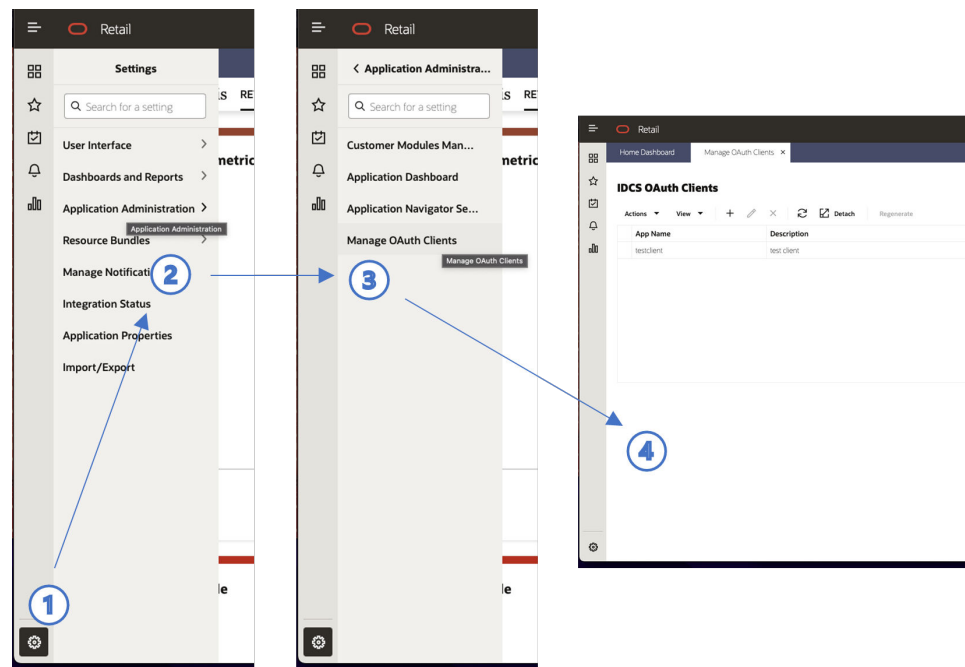
For pre-production environments (for example, stage, dev, uat), administrator users for Retail Home must be assigned the following roles (groups) through the authentication provider for the environment:

- RETAIL_HOME_ADMIN_PREPROD
- PLATFORM_SERVICES_ADMINISTRATOR_PREPROD
- PLATFORM_SERVICES_ADMINISTRATOR_ABSTRACT_PREPROD

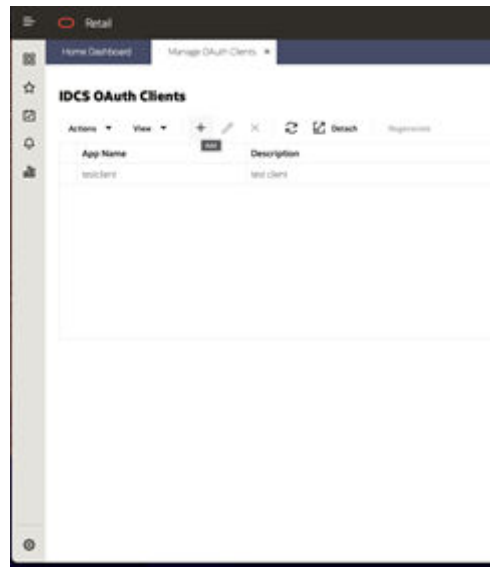
For production environments, administrator users for Retail Home must be assigned the following roles (groups) through the authentication provider for the environment:

- RETAIL_HOME_ADMIN
- PLATFORM_SERVICES_ADMINISTRATOR
- PLATFORM_SERVICES_ADMINISTRATOR_ABSTRACT

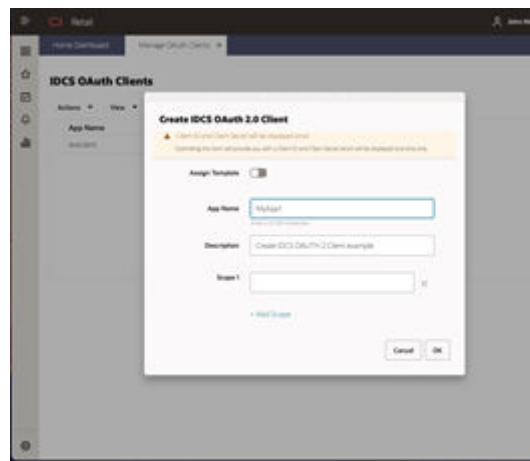
1. In Retail Home, navigate to Manage OAUTH Clients page by tapping settings (1), then tapping the Application Administration menu item (2), and lastly tapping the Manage OAUTH Clients menu item to arrive at the Manage OAUTH Clients page (4).



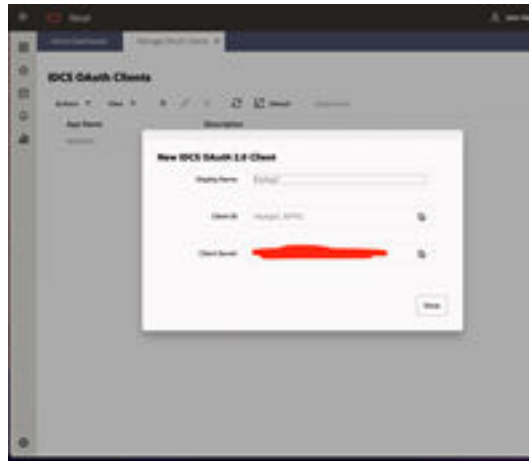
2. Tap the + button.



3. A popup dialog will appear. Provide an *App Name* and *Description*. Leave *Scope* blank. Tap **OK**.



4. A new dialog window will appear with a *Display Name*, *Client ID*, and *Client Secret*. **Retain this information**. It will not be displayed again. Tap **Done** when the information has been copied. Note that new credentials can be created at any time and that production, stage, and development will have different credentials.



Consult Retail Home Application Administration Guide for additional details on managing OAUTH clients.

Note that, the OCI IAM service is rate limited (see [API Rate Limits](#)). Best practice is to reuse tokens until they expire (one hour). If you encounter a 429 error when requesting a token or authenticating, you have hit the rate limit. When you encounter a rate limit, back off for one minute to reset the rate limiter.

Before proceeding:

1. Verify that a client id and secret can be created in Retail Home.
2. Retain the client id and secret for future use.

Generating an ORDS Access Token

You will need an IDCS Authorization Server endpoint URL and ORDS service credentials to perform the steps described below.

One uses an IDCS Authorization Server to generate an ORDS access token. Two access token generation techniques will be described, curl and POSTMAN. One is likely to use both techniques during the development process.

Generating an Access Token Using cURL

The cURL command for generating an access token has five components:

1. The IDCS Authorization Server endpoint URL
2. A content type
3. An authorization
4. A grant type
5. A scope

Only the IDCS Authorization Server endpoint URL and authorization are customer-specific. Content type, grant type, and scope are the same for all customers.

The endpoint URL has the following form:

```
https://<idcs authorization server host>/oauth2/v1/token
```

The authorization uses Basic Auth. You will need to base64 encode your Basic Auth credentials using the following format:

```
clientId:clientSecret
```

Replace *Client ID* and *Client Secret* with credentials obtained using the method described in the 4.3.2 *Obtaining ORDS Service Credentials* section above. Then use a base64 encoding tool to encode the string.

The cURL command to generate a token is as follows:

```
curl --location --request \
POST 'https://<idcs authorization server host>/oauth2/v1/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'Authorization: Basic <base64 clientId:clientSecret>' \
--data-urlencode 'grant_type=client_credentials' \
--data-urlencode 'scope=urn:opc:ldm:__myscopes__'
```

Generating an Access Token Using POSTMAN

Generating an access token in POSTMAN is typically an integral part of calling other services. In this section, we will illustrate the process of generating a token directly and generating it as part of another service invocation. Use the following steps to generate a token directly:

1. Open POSTMAN and create a new request by clicking on the **New** button in the top left corner of the screen.
2. Select HTTP.
3. In the new request tab, select the POST method from the drop-down menu.
4. Enter the IDCS Authorization Server endpoint URL in the "Enter request URL" field.
5. Click the **Authorization** tab to configure authorization.
6. In the **Type** drop-down menu, select **Basic Auth**.
7. Enter your username (client id) and password (client secret) in the fields provided.
8. Next click the **Body** tab to add the grant type and scope parameters.
9. In the menu, select **x-www-form-urlencoded**.
10. Next enter two key-value pairs:

Key	Value
grant_type	client_credentials
scope	urn:opc:ldm:__myscopes__

11. Once you have configured your request, click on the "Send" button to execute it.

12. The response from the service will be displayed in the "Response" section below the request configuration. You can view the response headers and body, as well as any errors or status codes. The response is JSON formatted and should have the following form:

```
{
  "access_token": "<token>",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

13. You can also save the request for future use by clicking on the "Save" button in the top right corner of the screen and giving it a name.

To use OAuth2 in Postman to invoke a ORDS service, you can follow these steps.

1. Open POSTMAN and create a new request.
2. Select the **Authorization** tab from the top of the request builder.
3. Select the **OAuth 2.0** type from the drop-down menu.
4. Scroll down to Configure New Token.
5. Choose a name for the token configuration.
6. Select client credentials as the grant type.
7. Enter your IDCS Authoization server endpoint URL, client id, client secret, and scope as you did above.
8. Set client authentication to Send as Basic Auth Header.
9. Scroll down to get new access token.
10. POSTMAN will then display the token details, such as the access token, refresh token, and token expiration time.
11. Finally, click the **Use Token** to apply the token to your service.

Before proceeding verify your understanding and validate your ORDS service credentials:

1. Unless you do not expect to use cURL, verify your that you can generate a token using cURL.
2. Unless you do not plan to use POSTMAN, then verify your understanding by generating a token using POSTMAN.
3. More than likely, you do not have an ORDS service with which to test authentication at this point. If you do and you expect to use POSTMAN, then verify your understanding by invoking an ORDS service.

8

Accessing Object Store

Overview

You will use object storage for bulk integration operations, high volume import, and export. If you do not anticipate using Object Storage for integration, you can skip this section. You will need to use the Oracle Cloud Console to perform the steps below.

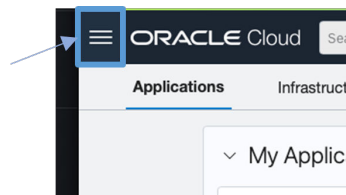
Obtaining a Pre-Authenticated Request (PAR) URL

A pre-authenticated request, or PAR, provide a way to let users access a bucket or object without having their own credentials. Users continue to have access to the bucket or object for as long as the creator of the request has permissions to access those resources.

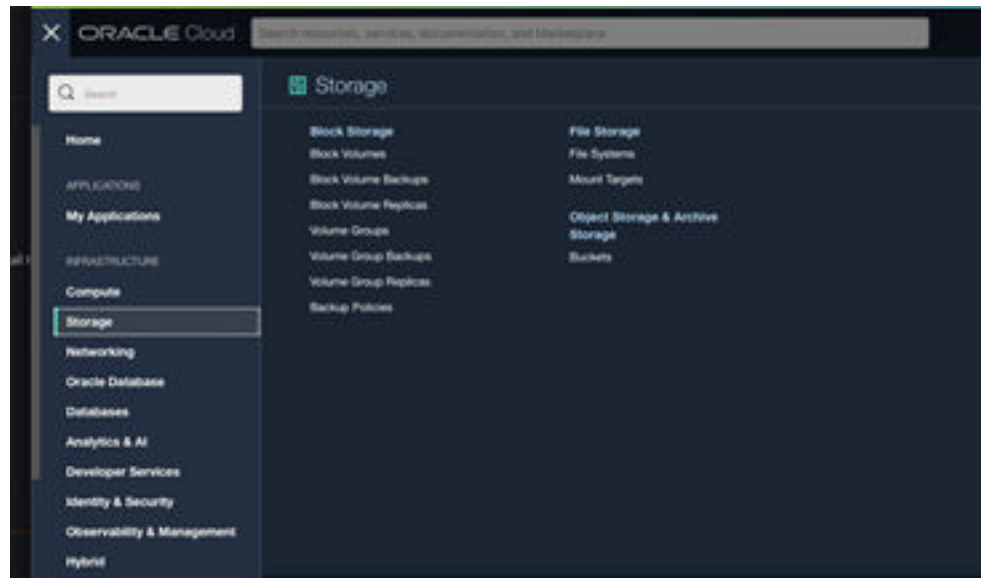
When you create a pre-authenticated request, a unique URL is generated. Anyone you provide with this URL can access the Object Storage resources identified by the pre-authenticated request. See *Using Pre-Authenticated Requests* in the *Oracle Cloud Infrastructure Documentation* for additional details.

The steps to create a writable PAR for specific object are as follows:

1. Log in to you Oracle Cloud account
2. Open the navigation menu in the upper left to work with services and resources. Services and resources are organized by functional group.



3. Open the navigation menu and click **Storage**.



4. Then click Buckets.
5. Select the appropriate compartment in the compartment select box. The object storage buckets in the compartment will be listed.
6. Select the appropriate bucket from the list.
7. In the *Resources* section, click **Pre-Authenticated Request**.
8. Specify the Name, select a Pre-Authenticated Request Target, select the Access Type, and the Expiration date.
9. Click **Create Pre-Authenticated Request**.
10. Copy the pre-authenticated request URL for your records.

Alternatively, one can create a PAR using from a shell using OCI os. Before proceeding verify your understanding and verify that you can create PAR.

Constructing an Object Storage Object URL

If you do not anticipate using Object Storage for integration, you can skip this section. You will need to use the Oracle Cloud Console to perform the steps below.

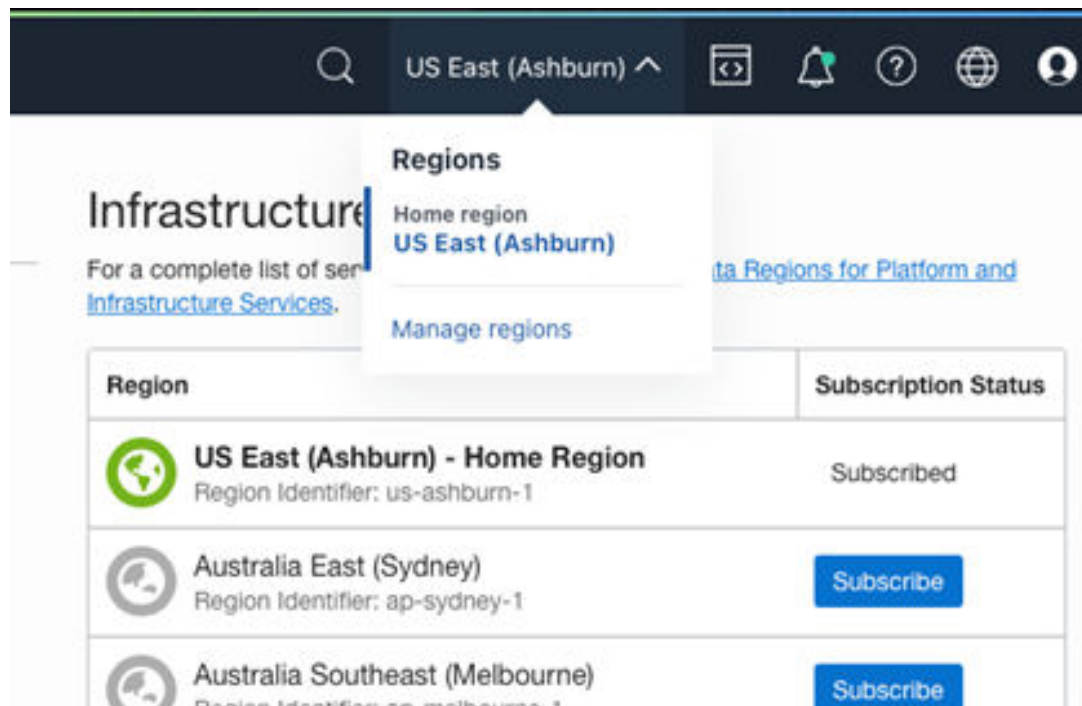
Unlike a PAR, an Object Storage object URL requires the schema user to have their own credentials. Like a PAR, the URL will provide a way for users to identify and access a bucket with a known name. In order to construct a URL, you will need to know:

- the region identifier for your object storage instance
- the namespace in which your object storage is located
- the name of the bucket that you will be using

To obtain the region identifier:

1. Log in to the Oracle Cloud Console
2. In the middle right portion of the tool bar at the top of the console page, you will find the name of the region in which your Object Storage instance is located (you can change regions from here as well, if need be), e.g., *US East (Ashburn)*. Tap the region to reveal the region menu.

3. In the region menu, there is a *Manage regions* menu item. Tap it.
4. A list of regions will be displayed. For each region displayed there is a region identifier. Note the region identifier for your region. See the figure below:



Once you have the region identifier, you can construct a base URI for your object store instance, which has the following form:

```
https://objectstorage.<region-identifier>.oraclecloud.com
```

For example, the host for a URI for *US East (Ashburn)* region is:

```
https://objectstorage.us-ashburn-1.oraclecloud.com
```

Next find the namespace for your bucket at the top of the *General* section in the *Bucket Information*. Lastly, find the name of the bucket at the top of the bucket page.

The complete URL is:

```
https://<host>/n/<namespace>/b/<bucket>/o/<object-prefix>
```

Note that the object prefix is a base name when exporting from ADW. The final object name will have a multi-part identifier (e.g., which is "1", if it is not a multipart export), timestamp suffix, a format extension (e.g., ".json") and, if compression was used, a compression extension (e.g., ".gz"). For example, an object name as listed in the bucket might look like the following:

```
ie_export_test_1_20221108T225927023493Z.json.gz
```

Before proceeding verify your understanding and verify that you can create an object storage object URL.

Obtaining Object Storage Credentials

You will need to use the Oracle Cloud Console to perform the steps below.

In order to read from or write to object storage one will need the necessary credentials. Refer to [Required Keys and OCIDs](#) for details on obtaining credential information for object storage. The easiest way to obtain the needed credentials is as follows:

1. Navigate to one's *My Profile* page in the Oracle Cloud (i.e., tap the profile button/image in the upper right corner and select *My Profile* from the drop down).
2. Next tap the *API Keys* link in the Resources section on the lower left of the screen.
3. Finally tap the *Add API Key* button and follow the instructions. Part of the process is downloading one's private key. The downloaded key is in PEM format. The key will need to be reformatted as a single long string without the leading and trailing dashes when using the credential in create credential script. There should be no new lines in the key.

These instructions will make more sense once one goes through the Add API Key process.

9

ORDS RESTful Services

Overview

The point of this section is not to say everything that needs to be said about RESTful Services. Rather it will describe some patterns one is likely to encounter and how they might be implemented. Those patterns are as follows:

- Data producing services or outbound integrations are one of the most likely patterns one will implement. Such services are typically pull producers that generate a chunk of data in response to an explicit request. A less common producer is one which continues to generate data without external prompting by invoking an external data consuming service. In this case, the service is either governed by the DBMS_SCHEDULER or by an external scheduling system.
- Data consuming services or inbound integrations insert data into RDS. The inbound data originates in an external system. There is no need to insert data found in the participating products or subscribe to RICS data originating in these products because it is already being replicated.
- Bulk import and export services represent another inbound and outbound integration pattern. In this case, however, data is transferred through object storage rather than through the service itself. The role of the service is to initiate the transfer.
- The last services pattern concerns process orchestration and monitoring. These service patterns start, stop, and monitor jobs. In most cases, these services will run asynchronously. For example, they will submit a job for future execution and return immediately.

Oracle RESTful Data Services play a role in every integration. Services either directly transfer the data or initiate that transfer through object storage. For a pull pattern, the service queries ADW and then returns the query result as its response. For a push pattern, the service is invoked with a payload that is inserted in a ADW table. In the case of a bulk export integration, the service initiates the export and then returns a response indicating whether the export was successfully initiated or not. Lastly, a bulk import integration service initiates the import and then returns a response indicating whether the import was successful or not. Additional services are required to monitor the progress of import and export *jobs*.

In most cases, the pattern chooses itself for a given task. Synchronous data services that push or pull data in response to an explicit request are simple to implement. The problem is that simple producers do not tend to scale to large volumes of data. First, there is a non-negotiable hard limit of 300 seconds on query duration. If the query exceeds this limit, the caller will return a socket hang up error. Moreover, it is also worth remembering that the database is not infinitely scalable. Specifically, there are only so many connections available (a max of 100) and there is only so much CPU capacity. One can't split a huge bulk data task into a multitude of smaller subtasks and expect them to require fewer CPU seconds. In the worst case, a backlog of REST invocations builds, and invocations start timing out.

Implementing a RESTful Service in APEX

In order to implement the examples below, you will need:

- Access to the APEX UI and Workspace so that you can create a service

The following paragraphs will only provide an overview of how one creates a RESTful service. Consult Chapter 7 of the SQL Workshop Guide, *Enabling Data Exchange with RESTful Services*. The chapter describes in detail how one creates a RESTful service in APEX. Bear in mind, documentation is version specific. Although documentation across versions tends to be quite similar, it is generally best to consult the documentation for the version of APEX one is using.

To begin, navigate to the APEX UI and select a workspace, such as MFCS, then follow the steps below:

1. From the APEX UI navigate to the **RESTful Services** page (i.e., from the APEX Navigation Bar **SQL Workshop > RESTful Services**).
2. Click **Modules**
3. Click **Create Module**
4. Name your module "imp_guide" and set the **Base Path** to "/imp_guide/"
5. Click **Create Module**

A module represents a collection of related services. Begin creating the first service by creating a template. The steps are as follows:

1. Click **Create Template**
2. Set the URI Template to "hello_world/:name"
3. Click **Create Template**

The ":name" path component allows us to introduce and demonstrate a bind variable.

The last step is to create a handler for the service. Create the handler by following steps below:

1. Click **Create Handler**
2. Set the *Method* to **GET**
3. Set *Source Type* to **Collection Query**
4. Set the *Source* to "select 'Hello World! ' || :name as response from dual"
5. Click **Create Handler**

The full URL is displayed on the ORDS Handler Definition page. The URL has the following form:

```
https://<host>/<tenant-name>/ords/<workspace>/imp_guide/hello_world/<your-name>
```

Note that the URL for service handler is displayed on the handler definition page.

Since this is a GET service, it can be tested from a browser. The response for this service, if *your_name* was *john* would be:

```
{
  "items": [
    {
      "response": "Hello World! john"
    }
  ]
}
```

```

    ],
    "hasMore": false,
    "limit": 25,
    "offset": 0,
    "count": 1,
    "links": [
      {
        "rel": "self",
        "href": "https://<host>/<tenant-name>/ords/mfcs/imp_guide/hello_world/
john"
      },
      {
        "rel": "describedby",
        "href": "https://<host>/<tenant-name>/ords/mfcs/metadata-catalog/
imp_guide/hello_world/item"
      },
      {
        "rel": "first",
        "href": "https://<host>/<tenant-name>/ords/mfcs/imp_guide/hello_world/
john"
      }
    ]
  }
}

```

Query parameters become bind variables. For example:

1. Edit the *Source* of your **hello_world** service to be "select 'Hello World! ' || :name || ' ' || nvl(:last_name, 'Smith') as response from dual"
2. Apply Changes.

The response for this service, if *your_name* was *john*?*last_name=jones* would be:

```

{
  "items": [
    {
      "response": "Hello World! john jones"
    }
  ],
  "hasMore": false,
  "limit": 25,
  "offset": 0,
  "count": 1,
  "links": [
    {
      "rel": "self",
      "href": "https://<host>/<tenant-name>/ords/mfcs/imp_guide/hello_world/
john?last_name=jones"
    },
    {
      "rel": "describedby",
      "href": "https://<host>/<tenant-name>/ords/mfcs/metadata-catalog/
imp_guide/hello_world/item"
    },
    {
      "rel": "first",
      "href": "https://<host>/<tenant-name>/ords/mfcs/imp_guide/hello_world/

```



```
john?last_name=jones"  
  }  
]  
}
```

Before proceeding:

1. Create the **hello_world** service in the APEX UI.
2. Test the service from a browser. You should be challenged (if you have not already been authenticated) when invoking the service. Use your IDCS login credentials to authenticate.

Invoking a RESTful Service from POSTMAN

In order to implement the example below, you will need:

- The **hello_world** service described above.
- Access to POSTMAN. The discussion below assumes familiarity POSTMAN.
- The ORDS OAUTH credentials created in [Obtaining ORDS Service Credentials](#).

Invoking a RESTful service from POSTMAN combines access token generation with endpoint access. Using POSTMAN allows you to test your services as well as simulate the fundamental tasks performed in service-based integration. To invoke your **hello_world** service using POSTMAN, follow these steps:

1. Open Postman and create a new request by clicking on the "New" button in the top left corner of the application. Select HTTP Request.
2. In the "Enter URL or paste text" field, enter the endpoint of the **hello_world** service.
3. The default HTTP method is GET. Examine the other methods, but leaving the setting as GET.
4. Click the Params tab and add the last_name parameter (i.e., key is last_name, value is Smith for example).
5. Click the Authorization tab.
 - a. Select the **OAuth 2.0** type from the drop-down menu.
 - b. Click on the "Get New Access Token" button.
 - c. In the "Get New Access Token" popup window, fill in your access token URL (IDCS Authorization Server endpoint URL), client ID, client secret, grant type, and scopes.
 - d. Once you have filled in the required fields, click on the **Get New Access Token** button.
 - e. POSTMAN will then display the token details, such as the access token, refresh token, and token expiration time.
 - f. Finally, click the **Use Token** to apply the token to your service.
6. Once you have configured the request, click on the "Send" button to send the request to the RESTful service.
7. You will see the response from the RESTful service in the "Response" section of the request window. You can view the response headers, body, and status code to verify that the request was successful.

Before proceeding invoke your **hello_world** service from POSTMAN.

ORDS PRE-HOOK

Oracle REST Data Services (ORDS) provides the ability to use PL/SQL based pre-hook functions that are invoked prior to an ORDS based REST call. These functions can be used for a variety of purposes including auditing, custom authentication and authorization, and metrics gathering.

Each provided RDS workspace comes pre-configured with a simple pre-hook function named `ORDS_PREHOOK`, and it has a default implementation that simply returns `true`. As such, it has no effect on the REST calls made into custom applications. It is provided as a starting point for extension to teams that required additional processing on each REST call. For those teams, replacing the implementation of the `ORDS_PREHOOK` function will enable the additional capabilities they require. For more information on pre-hook functions, please refer to [Oracle REST Data Services Installation, Configuration, and Development Guide: Overview of Pre-hook Functions](#).

Note that, a pre-hook function is invoked for every REST service call. Therefore, the pre-hook function must be designed to be efficient. If a pre-hook function is inefficient, then it has a negative effect on the performance of the REST service call. Invoking the pre-hook involves at least one additional database round trip. It is critical that the ORDS instance and the database are located close together so that the round-trip latency overhead is minimized.

Be aware that some extensions, such as those provided by the Oracle Retail Cloud Value team, use `ORDS PRE_HOOK` to enhance security. Incomplete configuration of these extensions as well as failure to communicate their presence to the broader customer implementation team can result in unexpected authentication failures.

Using RDS to Build a Data Producing Service

Overview

A data producing service can be used to deliver data to a UI or fulfill some data need in an automated business process. The limiting factor is time. *The data producing service must be able to produce a response in less than 300 seconds. If the service exceeds that limit, the caller will hang up and respond with a **socket hang up** error.* The consumer may be able to wait longer than 300 seconds, but ORDS will not.

! Important

ALL REST service calls must complete within 300 seconds. This hard limit is enforced by ORDS and cannot be extended.

Most data producing services are parameterized to one degree or another. The **hello_world** service demonstrated the use of a parameter in the URL template as well as the use of a query parameter. When considering how best to communicate parameters to a service, the developer should be aware that a URL has a maximum size (*Oracle will not guarantee support of any maximum*). If the parameters are complex or lengthy, then parameters should be passed to the service in the body of the request. The format of the request body is up to the developer; however, the body should be easy to parse in PL/SQL, e.g., JSON formatted requests are easy to parse. If the request includes a body, then the request method must be POST or PUT. *GET methods ignore the request body.*

GET Services

In order to implement the example below, you will need:

- Access to the APEX UI.
- The **hello_world** service described above.
- Access to POSTMAN. The discussion below assumes familiarity POSTMAN.
- The ORDS OAUTH credentials created in [Obtaining ORDS Service Credentials](#).

GET services are particularly easy to implement in the APEX UI. The service source is the query. Complicated queries should be implemented, when possible, as views to keep the service source simple. Views are more easily tested. *Bear in mind, the service source is not compiled until the service is invoked. In other words, the first indication of a compilation error is the error message in the service response.*

The **hello_world** service was configured as a collection query that anticipates returning multiple rows. The result is also paged, meaning links to next and previous pages are supplied in the response. The response is also formatted as a JSON object. For additional details refer to the ORDS Developers Guide.

Before proceeding:

- Review the response to the invocation of the **hello_world** and make sure that you understand each element of the response.
- Change the result type to a *collection query item* and invoke the service. Make sure you understand each element of the collection query item response.
- Lastly, navigate in your browser to the ORDS Developers Guide. Take the time to review familiarize yourself with its contents.

POST Services

In order to implement the example below, you will need:

- Access to the APEX UI.
- Access to POSTMAN. The discussion below assumes familiarity POSTMAN.
- The ORDS OAUTH credentials created in [Obtaining ORDS Service Credentials](#).

Sometimes it is necessary to use a POST service in a GET-like setting because the query string would be too complicated (bin64 encoding is an option, but will not be discussed here). POST services, however, are more complicated to implement. In this case, the method is POST and the source type is PL/SQL. The service source is a PL/SQL block. There is no paging. There is no automatic rendering of the response in JSON format. The body or payload is retrieved using the implicit bind variable **:body_text**. The format of the body is up to you. Bear in mind, whatever format you choose, you will have to parse or unpack it.

For example, the following service source:

```
declare
    payload varchar2(128) := :body_text;
    response varchar2(64);
    first_name varchar2(64);
    last_name varchar2(64);
begin
    first_name := json_value(payload, '$.first_name');
    last_name := json_value(payload, '$.last_name');
    select json_object('response' value '"hello ' ||
first_name || ' ' || last_name || '" ' format json)
        into response
    from dual;
    http.prn(response);
end;
```

Illustrates four important tasks:

- Obtaining the service body or payload using **:body_text**. Note, **:body_text** can only be read once.
- Unpacking the payload using **json_value**.
- Building a JSON response.
- Returning a response using **http.prn**.

The service when given a payload of:

```
{"last_name":"smith", "first_name":"john"}
```

Returns with a response of:

```
{"response":"hello john smith"}
```

Before proceeding, create and test the POST service above using POSTMAN.

Long Responses

In some cases, the response from your service exceeds the capacity of **varchar2**. When this happens, replace the **varchar2** response with a **clob**. The procedure, HTP.PRN procedure used above will not, however, work with a **clob**. For http printing clobs, I created the HTP_PRN_CLOB procedure below.

```
create or replace PROCEDURE HTP_PRN_CLOB(PCLOB IN OUT NOCOPY CLOB)
IS
    V_TEMP VARCHAR2(4000);
    V_CLOB CLOB := PCLOB;
    V_AMOUNT NUMBER := 3999;
    V_OFFSET NUMBER := 1;
    V_LENGTH NUMBER := DBMS_LOB.GETLENGTH(PCLOB);
    V_RESULT CLOB;
BEGIN
    WHILE V_LENGTH >= V_OFFSET LOOP
        V_TEMP := DBMS_LOB.SUBSTR(V_CLOB, V_AMOUNT, V_OFFSET);
        HTP.PRN(V_TEMP);
        V_OFFSET := V_OFFSET + LENGTH(V_TEMP);
    END LOOP;
END;
```

12

Using RDS to Build a Data Consuming Service

In order to implement the example below, you will need:

- Access to the APEX UI.
- Access to POSTMAN. The discussion below assumes familiarity POSTMAN.
- The ORDS OAUTH credentials created in [Obtaining ORDS Service Credentials](#).

A data consuming service updates an existing a *custom* RDS table, emphasis on custom. All the replicated views and tables in RDS are read-only. If you want to add data to RDS, you will need to create a table for it. The semantics of http methods strongly encourages you to use the POST, PUT, and DELETE methods modification, specifically, it discourages creating GET handlers that have side effects. From the service implementation perspective, a data consuming service is no different than the POST service described in [POST Services](#). The difference is that the service source unpacks the payload (or query string parameters) and then inserts a new record or updates an existing record based on the results of that unpacking.

For example, the following service source inserts a row into a table name **hello_world_names**. This table has two columns, **last_name** and **first_name**. The first name is part of the URL and the last name is found in the body of the post method.

```
declare
    payload varchar2(128) := :body_text;
    response varchar2(64);
    first_name varchar2(64) := :name;
    last_name varchar2(64);
begin
    last_name := nvl(json_value(payload, '$.last_name'),
                    'no_last_name_given');
    insert into hello_world_names (last_name, first_name)
        values (last_name, first_name);
    http.prn('{"status":"success"}');
end;
```

Before proceeding:

1. Create **hello_world_names** table in your RDS.
2. Add (create) a POST handler for your **hello_world** service described in [POST Services](#) to insert a row in the **hello_world_names**. Use the example source above.
3. Test your new POST handler using POSTMAN.

13

Exporting Data to Object Storage

Prerequisites

To implement the example below, you will need:

- Access to Oracle Cloud Console.
- Access to the Object Storage Service.
- Ability to list buckets and their objects.
- Ability to create a PAR.
- Access to the APEX UI.
- Access to POSTMAN. The discussion below assumes familiarity POSTMAN.
- The ORDS OAUTH credentials created in [Obtaining ORDS Service Credentials](#).

There are two approaches to exporting data to object storage. The approaches only differ in the type for file URI used. The first approach uses a pre-authenticated request (PAR). The second uses a URI that requires authentication.

Exporting Data Using a PAR

Create a bucket level or prefix PAR using the process described in Section 4.3.4. An object level PAR will not work. The example code below demonstrates how a PAR can be used to export data from RDS.

```
Begin
  dbms_cloud.export_data(
    file_uri_list=> '<your-PAR-goes-here>',
    query => 'select 1 from dual',
    format => json_object('type' value 'csv')
  );
  http.prn('{"status":"success"}');
end;
```

Before proceeding:

1. Create a bucket level PAR for an export test.
2. Execute the above code in APEX > SQL Commands
 - a. Verify an export was created by listing the objects in your bucket.
3. Implement an export POST service using the example code above and your PAR.
4. Test the service using POSTMAN.
 - a. Verify an export was created by listing the objects in your bucket.

Exporting Data with a Credential

In [Obtaining Object Storage Credentials](#), you created credentials that can be used to import data from and export data to object storage. You will need those credentials to run the example described here. The first step to exporting data using an unauthenticated file URI is configure a credential for use in ADW. An example follows.

```
begin
  DBMS_CLOUD.CREATE_CREDENTIAL (
    credential_name => 'OCI_KEY_CRED',
    user_ocid => '<your-user-ocid>',
    tenancy_ocid=> '<your-tenancy-ocid>',
    private_key=> '<your-private-key>',
    fingerprint=> '<your-fingerprint>'
  );
end;
```

The export code for an unauthenticated file URI adds this credential to the calling parameters.

```
begin
  dbms_cloud.export_data(
    credential_name => '<your-credential>',
    file_uri_list=> '<your-URI-goes-here>',
    query => 'select 1 from dual',
    format => json_object('type' value 'csv')
  );
  http.prn('{ "status": "success" }');
end;
```

Before proceeding:

1. Configure your credential using the information you obtained in [Obtaining Object Storage Credentials](#).
2. Execute the above code in APEX > SQL Commands.
 - a. Verify an export was created by listing the objects in your bucket.
3. Implement an export POST service using the example code above and your file URI. The format of the URI is described in [Constructing an Object Storage Object URL](#).
4. Test the service using POSTMAN.
 - a. Verify an export was created by listing the objects in your bucket.

Importing Data from Object Storage

Prerequisites

To implement the example below, you will need:

- Access to Oracle Cloud Console.
- Access to the Object Storage Service.
- Ability to list buckets and their objects.
- Ability to create a PAR.
- Access to the APEX UI.
- Access to POSTMAN. The discussion below assumes familiarity POSTMAN.
- The ORDS OAUTH credentials created in section 4.3.2.
- A table in which to import data.

As with exporting, there are two approaches to importing data from object storage. The approaches only differ in the type for file URI used. The first approach uses a pre-authenticated request (PAR). The second uses a URI that requires authentication.

Importing Data Using a PAR

Create a readable PAR (bucket level, prefix, or object level) using the process described in [Obtaining a Pre-Authenticated Request \(PAR\) URL](#). The example code below demonstrates how a PAR can be used to export data from RDS. Note that you will need a table into which your data will be imported and the source file and the destination table will need to be compatible.

```
begin
dbms_cloud.copy_data(
    file_uri_list=> '<your-URI-goes-here>',
    table_name => <your-table-name>,
    format => json_object('type' value 'csv'));
end;
```

Before proceeding:

1. Create a PAR for an import test.
2. Execute the above code in APEX > SQL Commands
 - a. Verify an import was successful examining the data in your destination table.
3. Implement an import POST service using the example code above and your PAR.
4. Test the service using POSTMAN.
 - a. Verify an import was successful examining the data in your destination table.

Importing Data Using a Credential

In [Obtaining Object Storage Credentials](#), you created credentials that can be used to import data from and export data to object storage. You will need those credentials to run the example described here. The first step to exporting data using an unauthenticated file URI is configure a credential for use in ADW. You should have done this in [Exporting Data with a Credential](#). You can reuse that credential here. The import code for an unauthenticated file URI adds this credential to the calling parameters.

```
begin
  dbms_cloud.copy_data(    credential_name => '<your-credential>',
                          file_uri_list=> '<your-URI-goes-here>',    table_name => <your-table-
name>,    format => json_object('type' value 'csv'));
end;
```

Before proceeding:

1. Configure your credential using the information you obtained in Section 4.3.6
2. Execute the above code in APEX > SQL Commands
 - a. Verify an import was successful examining the data in your destination table.
3. Implement an import POST service using the example code above and your file URI. The format of the URI is described in Section 4.3.5.
4. Test the service using POSTMAN.
 - a. Verify an import was successful examining the data in your destination table.

DBMS Scheduler Jobs

An asynchronous approach is generally called for when the likely wait time for process completion is high. A data export to object storage is generally a good candidate for an asynchronous start. In the simplest case, one needs to implement three data services: job start, job stop, and job status. The DBMS_SCHEDULER package provides the functionality one would need for these services. There is, of course, the option to schedule an export job to repeat and obviate the need to create a job start service. One could still use a job start service to invoke an unscheduled export.

One uses the DBMS_SCHEDULER.create_job procedure to create a job that can be started asynchronously. A typical approach would be to use create job to wrap a procedure. The create job invokes the procedure immediately (by setting the start_date to SYSTIMESTAMP) upon creation and is dropped automatically upon completion. The service would return a unique job name or execution id to be used to stop and monitor the job.

Another service is used to monitor the job status using the returned execution id. The monitoring service would be used to poll the status of the job. The job status is obtained by executing a query on the DBMS_SCHEDULER.user_scheduler_job_run_details. A complete reference implementation of an asynchronous job start and monitoring framework is available on My Oracle Support. To view the reference implementation:

1. Log in to my oracle support.
2. Search for *Oracle Retail Data Store Documentation Library*
3. Navigate to Sample Code
4. Click on the link *Sample Code*

16

Retail Home Integrations

A Retail Home integration is an example of outbound integration with a user interface or portal. Retail Home Metric tiles without charts are quite simple to implement. For example, the following data service source (with a source type of collection query) will populate the 2 Metric Tile below:

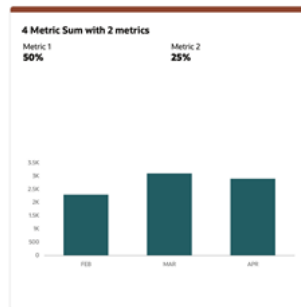


```
select
  'PO Receipts' as NAME, 25680 as VALUE,
  'N' as "VALUE_FORMAT" from dual
union
select
  'In Transit' as name, 112300 as value,
  'N' as "VALUE_FORMAT" from dual
```

This data service response is:

```
{"items":[
  {"name":"In Transit","value":112300,"value_format":"N"},
  {"name":"PO Receipts","value":25680,"value_format":"N"}
],
"hasMore":false,
"limit":25,
"offset":0,
"count":2,
"links":[...]}
}
```

Producing a 4 Metric Summary, however, is more complicated and requires one use a source type of PL/SQL (the following code only provide values for two of four metrics).



```

declare
    response varchar2(4000);
begin
    SELECT json_object (
        'items' value
            json_array(
                json_object ('name' value 'Metric 1',
                    'value' value 0.5,
                    'valueFormat' value 'PC'),
                json_object ('name' value 'Metric 2',
                    'value' value 0.25,
                    'valueFormat' value 'PC')
            ),
        'chart' value
            json_object ('type' value 'bar',
                'items' value
                    json_array(json_object('name' value 'FEB',
                        'value' value 2300),
                        json_object('name' value 'MAR',
                            'value' value 3100),
                        json_object('name' value 'APR',
                            'value' value 2900)
                    ),
                'valueFormat' value 'S',
                'seriesName' value 'Sales',
                'valueLabel' value 'Amount'
            )
    )
    into response FROM DUAL;
    http.print(response);
end;
```

Filters, if used, become query string parameters and values in the URL. The query string parameters manifest in the source as bind variables.

In Context Launch of an APEX App

In context launch of an APEX App entails navigating to an APEX App from within a product application using a URL. Once you have deployed your application, loaded the data, and created users, you can publish your production URL. You can determine the production URL for your application by either:

- Selecting the application on the Application home page and right clicking the **Run** button. Then, select **Copy link address** or **Copy link location** depending on your browser.
- Running the application and then copying the URL.

Invoking an APEX app with one or more query parameters requires that the APEX App *Session State Protection* and *Application Items* be appropriately configured.

1. In your APEX App navigate to *Shared Components > Security > Session State Protection*. Navigate to the *Set Page and Item Protection* page of your published launch page, for example, your *App Home* page. Next, set page access restriction to *Unrestricted*. **Always treat query parameter input as untrusted and sanitize it.**
2. Next navigate to *Shared Components > Application Logic > Application Items*. Create an application item of the same name as your query parameter.

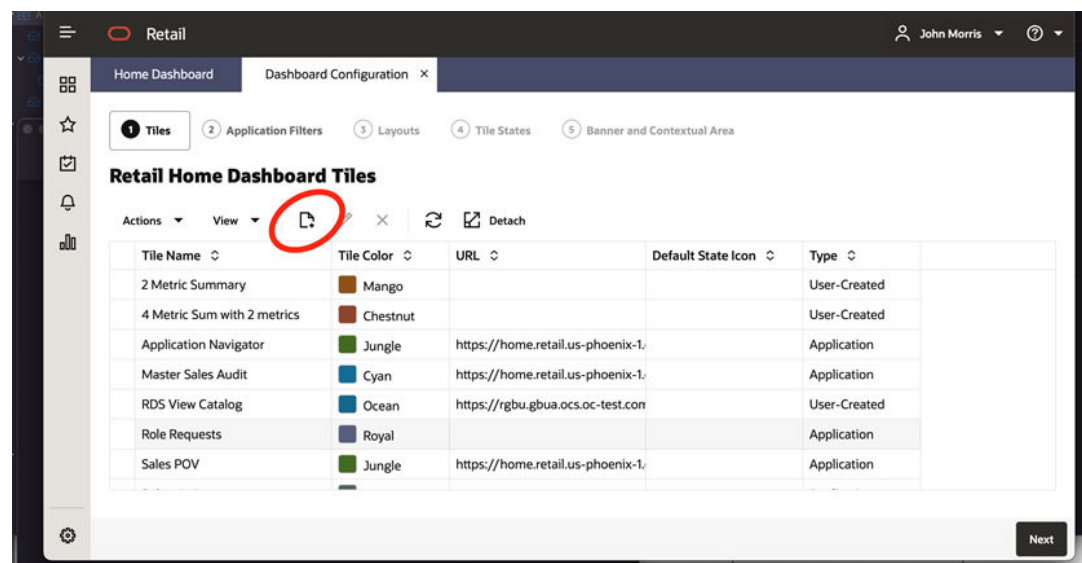
See [Oracle APEX](#) for additional details on URL syntax and managing session state.

Launching APEX Apps from Retail Home

It is quite simple to configure Retail Home to facilitate the launch of an APEX application.

1. Navigate to the Dashboard Configuration tab and tap the Create button.

Figure 16-1 Dashboard Configuration Tab



2. Fill in the Create Dashboard Tile Dialog. Note there is a field for specifying a URL for you new dashboard tile.

Figure 16-2 Create Dashboard Tile

Create Dashboard Tile

Tile Name: RDS View Catalog

Application: None

URL: <https://rgbu.gbua.ocs.oc-test.com/fxgarc7tq7h1y00>
APEX Application URL

Override URL: ☐

Tile Color: Ocean

OK and Add Another Cancel OK

Invoking External Services

Publicly accessible, external services can be invoked using the `APEX_WEB_SERVICE` package. The `UTL_HTTP` package is not supported. If you choose to setup a reverse private endpoint for ADW, you will need to whitelist any external service that you want to invoke from `APEX_WEB_SERVICE`.

Retail DB Ops Console

Customers are responsible for both tuning their extensions and monitoring their CPU and storage capacity utilization. The Retail DB Ops Console provides customers the tools necessary to fulfill this responsibility. Additional capabilities provided in various Oracle DB packages can provide more detailed metrics and statistics. Consult Oracle documentation for further information.

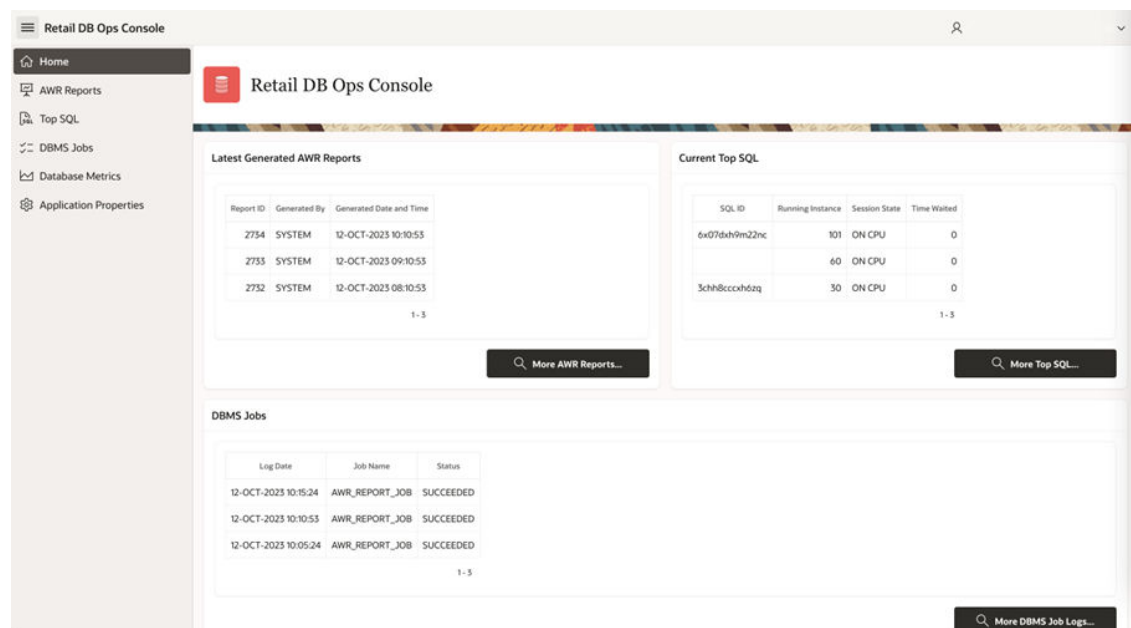
Home

The Retail DB Ops Console can be accessed from Retail Home's Application Navigator. The home page can be accessed by a 'Viewer' having RDS_MANAGEMENT_VIEWER or RDS_MANAGEMENT_VIEWER_PREPROD roles.

The Home page gives a quick view of recent data for the following:

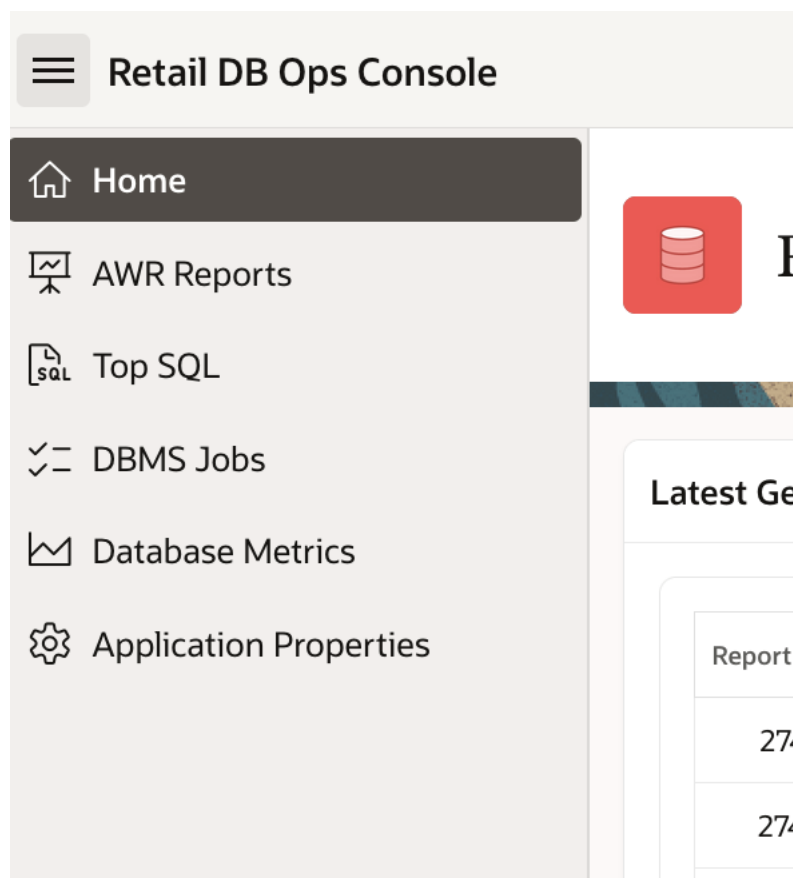
- Latest Generated AWR Reports.
- Current Top SQL
- DBMS Jobs

Figure 18-1 DB Ops Console



All features can be accessed from the left panel.

Figure 18-2 DB Ops Console Menu



AWR Reports

The AWR stands for Automated Workload Repository Report and provides a set of tables into which snapshots of system statistics are stored. Generally, these snapshots are taken on an hourly basis and include wait interface statistics, top SQL, memory, and I/O information that is cumulative in nature up to the time of the capture.

The AWR report process takes the cumulative data from two snapshots and subtracts the earlier snapshot's cumulative data from the later snapshot and then generates a delta report showing the statistics and information relevant for the time period requested.

Search Generated AWR Reports

Search Generated AWR Reports screen can be accessed by a 'Viewer' having RDS_MANAGEMENT_VIEWER or RDS_MANAGEMENT_VIEWER_PREPROD roles and shows a list of generated reports and allows the user to filter the reports based on the following:

- Snap ID – The unique key of a snapshot
- Generated By – System or User
- Interval by Date and Hour

Figure 18-3 AWR Reports

AWR Generated Report Logs

Report ID	Start Snap ID	End Snap ID	Start Interval	End Interval	Generated By	Generated Date and Time	Allow Delete
1993	23506	23507	12-SEP-2023 09:00:05	12-SEP-2023 10:00:02	SYSTEM	12-SEP-2023 11:11:04	N
1994	23507	23508	12-SEP-2023 10:00:02	12-SEP-2023 11:00:11	SYSTEM	12-SEP-2023 12:11:01	N
1995	23508	23509	12-SEP-2023 11:00:11	12-SEP-2023 12:00:02	SYSTEM	12-SEP-2023 13:10:56	N
1996	23509	23510	12-SEP-2023 12:00:02	12-SEP-2023 13:00:14	SYSTEM	12-SEP-2023 14:10:57	N
1997	23510	23511	12-SEP-2023 13:00:14	12-SEP-2023 14:00:01	SYSTEM	12-SEP-2023 15:10:56	N
1998	23511	23512	12-SEP-2023 14:00:01	12-SEP-2023 15:00:01	SYSTEM	12-SEP-2023 16:11:01	N
1999	23512	23513	12-SEP-2023 15:00:01	12-SEP-2023 16:00:02	SYSTEM	12-SEP-2023 17:11:02	N
2000	23513	23514	12-SEP-2023 16:00:02	12-SEP-2023 17:00:02	SYSTEM	12-SEP-2023 18:11:03	N
2001	23514	23515	12-SEP-2023 17:00:02	12-SEP-2023 18:00:01	SYSTEM	12-SEP-2023 19:11:03	N
2002	23515	23516	12-SEP-2023 18:00:01	12-SEP-2023 19:00:01	SYSTEM	12-SEP-2023 20:11:02	N
2003	23516	23517	12-SEP-2023 19:00:01	12-SEP-2023 20:00:03	SYSTEM	12-SEP-2023 21:10:56	N
2004	23517	23518	12-SEP-2023 20:00:03	12-SEP-2023 21:00:17	SYSTEM	12-SEP-2023 22:11:02	N

The list of generated report logs is provided in the “AWR Generated Report Logs” table that contains the following information for every Report ID. The ‘Refresh Report Logs’ button can be used to refresh the table to display the newly generated reports.

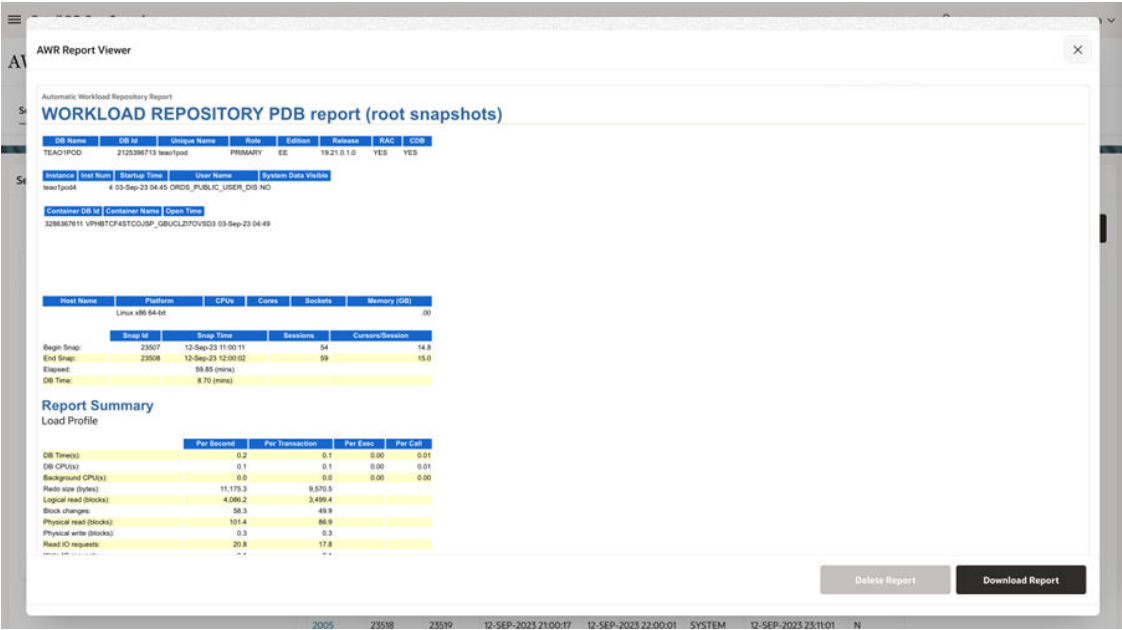
- Report ID: Unique ID of the generated report.
- Start Snap ID: Snap ID of the start snapshot in that time interval.
- End Snap ID: Snap ID of the end snapshot in that time interval.
- Start Interval: Start timestamp of the snapshot interval.
- End Interval: End timestamp of the snapshot interval.
- Generated By: If generated automatically by the system, then SYSTEM is populated. If generated manually by any user, then that user’s ID is populated.
- Generated Date and Time: Timestamp of the report generation.
- Allow Delete: Indicator that mentions if the report can be deleted or not.

Generated reports are retained for 30 days.

AWR Report Viewer

When clicking on the Report ID, a detailed AWR Report is displayed. The report can be downloaded or deleted from this window.

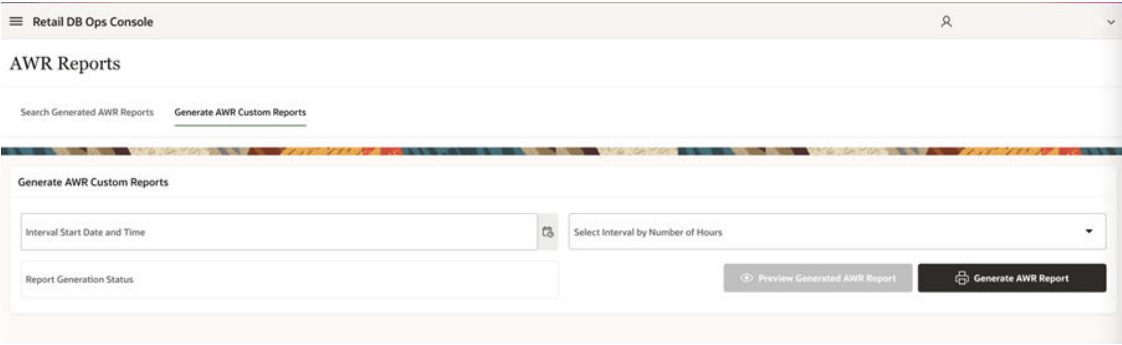
Figure 18-4 AWR Report Viewer



Generate AWR Custom Reports

This screen enables an 'Owner' having RDS_MANAGEMENT_OWNER or RDS_MANAGEMENT_OWNER_PREPROD roles to generate reports based on Interval Start Date and Time and Interval by Number of Hours. Custom reports are retained for 30 days and then purged.

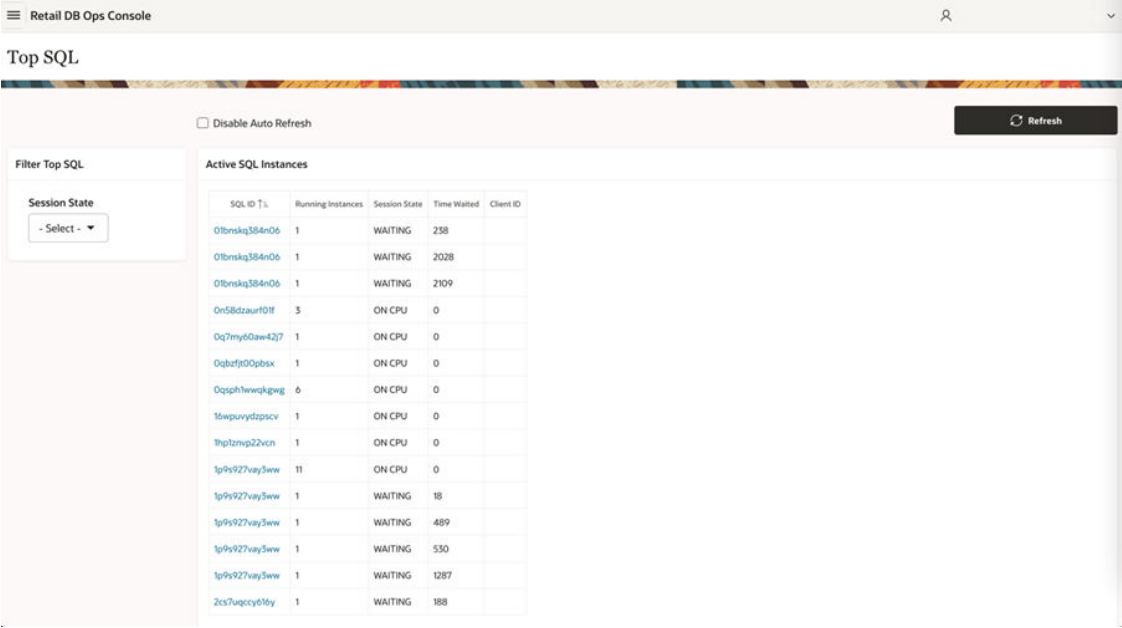
Figure 18-5 AWR Reports Generation



Top SQL

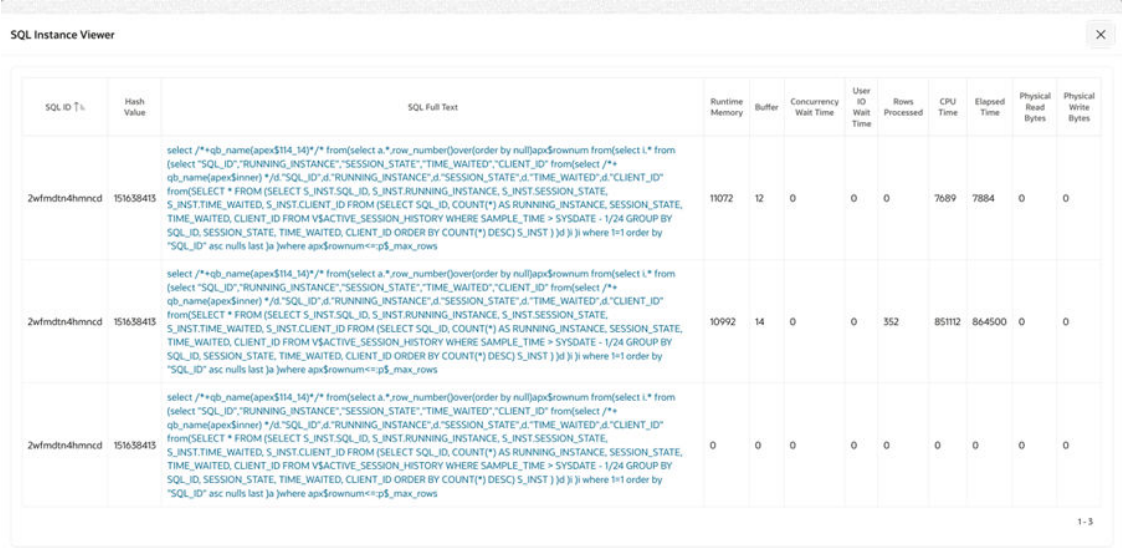
Top SQL screen displays a list of active SQL instances and their details that the snap process collects from SQL statements. The instances also can be filtered based on the Session States – ON CPU and WAITING. This page can be accessed by a 'Viewer' having RDS_MANAGEMENT_VIEWER or RDS_MANAGEMENT_VIEWER_PREPROD roles.

Figure 18-6 Top SQL



Clicking on the SQL ID link opens up a window with more details on the SQL Instance.

Figure 18-7 SQL Instance Viewer



Clicking on the SQL Full Text link shows the complete SQL statement.

Figure 18-8 SQL Text Viewer



DBMS Jobs

The DBMS Jobs table lists the available DB jobs and their details. This page can be accessed by a 'Viewer' having RDS_MANAGEMENT_VIEWER or RDS_MANAGEMENT_VIEWER_PREPROD roles.

Figure 18-9 DBMS Jobs

Available Database Jobs								
Job Name	Job Style	Job Creator	Program Name	Job Type	Job Action	Schedule Owner	Schedule Name	Sub T
AWR_REPORT_JOB	REGULAR	ORDS_PUBLIC_USER_DIS	AWR_REPORT_PROGRAM			RDS_MANAGEMENT	AWR_REPORT_HOURLY	NAM
DELETE_AUTO_AWR_REPORT_JOB	REGULAR	ORDS_PUBLIC_USER_DIS	DELETE_AUTO_AWR_REPORT_PROG			RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_SCHD	NAM
RDS_RDS_MANAGEMENT_GRANT_JOB	REGULAR	RDS_MANAGEMENT		PLSQL_BLOCK	BEGIN RDS_RDS_MANAGEMENT_GRANT_PRIV_SYNONYM; END;			CALI

Clicking on the Job Name opens a window with detailed job log and job run details. For example, the job log and job run details for 'RDS_RDS_MANAGEMENT_GRANT_JOB' is as shown below.

Figure 18-10 DBMS Scheduler Job Logs

DBMS Job Logs

Scheduler Job Logs Scheduler Job Run Details

Filter Job Logs

Job Status

- Select -

Log Date Range

☐ 20231012 00:00:00 - 23:59:59
☐ 20231011 00:00:00 - 23:59:59
☐ 20231010 00:00:00 - 23:59:59
☐ 20231009 00:00:00 - 23:59:59
☐ 20231008 00:00:00 - 23:59:59
☐ 20231007 00:00:00 - 23:59:59
☐ 20231006 00:00:00 - 23:59:59
[Show All](#)

Log ID ↑	Log Date	Owner	Job Name	Job Class	Operation	Status	Additional Information
3786884	12-SEP-2023 03:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3787332	12-SEP-2023 04:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3787782	12-SEP-2023 05:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3788230	12-SEP-2023 06:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3788676	12-SEP-2023 07:40:32	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3789124	12-SEP-2023 08:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3789570	12-SEP-2023 09:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3790020	12-SEP-2023 10:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3790466	12-SEP-2023 11:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3790912	12-SEP-2023 12:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3791358	12-SEP-2023 13:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3791808	12-SEP-2023 14:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3792254	12-SEP-2023 15:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3792702	12-SEP-2023 16:40:31	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	
3793150	12-SEP-2023 17:40:32	RDS_MANAGEMENT	DELETE_AUTO_AWR_REPORT_JOB	DEFAULT_JOB_CLASS	RUN	SUCCEEDED	

1 - 15 Next ▶

Figure 18-11 DBMS Scheduler Job Run Details

DBMS Job Logs

Scheduler Job Logs Scheduler Job Run Details

Filter Job Run Details

Job Status

- Select -

Log Date Range

☐ 20231012 00:00:00 - 23:59:59
☐ 20231011 00:00:00 - 23:59:59
☐ 20231010 00:00:00 - 23:59:59
☐ 20231009 00:00:00 - 23:59:59
☐ 20231008 00:00:00 - 23:59:59
☐ 20231007 00:00:00 - 23:59:59
☐ 20231006 00:00:00 - 23:59:59
[Show All](#)

Log ID ↑	Log Date	Owner	Job Name	Status	Additional Information	Error Number	Required Start Date	Actual Start Date	Run Duration	Instance ID	Session ID	Ag Pro I
3786698	12-SEP-2023 03:16:00	RDS_MANAGEMENT	RDS_RDS_MANAGEMENT_GRANT_JOB	SUCCEEDED		0	12-SEP-2023 03:16:00	12-SEP-2023 03:16:00	00:01	4	6017,58804	104
3786954	12-SEP-2023 03:46:00	RDS_MANAGEMENT	RDS_RDS_MANAGEMENT_GRANT_JOB	SUCCEEDED		0	12-SEP-2023 03:46:00	12-SEP-2023 03:46:00	00:01	4	21661,20884	336
378752	12-SEP-2023 04:16:00	RDS_MANAGEMENT	RDS_RDS_MANAGEMENT_GRANT_JOB	SUCCEEDED		0	12-SEP-2023 04:16:00	12-SEP-2023 04:16:00	00:01	4	28854,42515	257
3787374	12-SEP-2023 04:46:00	RDS_MANAGEMENT	RDS_RDS_MANAGEMENT_GRANT_JOB	SUCCEEDED		0	12-SEP-2023 04:46:00	12-SEP-2023 04:46:00	00:01	4	51697,48192	212
3787598	12-SEP-2023 05:16:00	RDS_MANAGEMENT	RDS_RDS_MANAGEMENT_GRANT_JOB	SUCCEEDED		0	12-SEP-2023 05:16:00	12-SEP-2023 05:16:00	00:01	4	40288,218	72C
3787830	12-SEP-2023 05:46:00	RDS_MANAGEMENT	RDS_RDS_MANAGEMENT_GRANT_JOB	SUCCEEDED		0	12-SEP-2023 05:46:00	12-SEP-2023 05:46:00	00:01	4	19242,9709	175
3788042	12-SEP-2023 06:16:00	RDS_MANAGEMENT	RDS_RDS_MANAGEMENT_GRANT_JOB	SUCCEEDED		0	12-SEP-2023 06:16:00	12-SEP-2023 06:16:00	00:01	4	2426,38565	164
3788268	12-SEP-2023 06:46:00	RDS_MANAGEMENT	RDS_RDS_MANAGEMENT_GRANT_JOB	SUCCEEDED		0	12-SEP-2023 06:46:00	12-SEP-2023 06:46:00	00:01	4	28254,12357	119P

Database Metrics

A database metrics page can be accessed by an 'Administrator' (i.e., someone having RDS_MANAGEMENT_ADMINISTRATOR or RDS_MANAGEMENT_ADMINISTRATOR_PREPROD roles). These metrics describe different aspects of database load. The following table lists the eight database metrics viewable for this environment's Oracle APEX and database instance.

Metric	Description
CPU Utilization	The percentage of processing capacity currently used by active processes
Storage Utilization	The percentage of subscription capacity currently allocated. See the note on storage utilization below.
Session Count	A logical entity in the database instance memory that represents the state of a current user log in to a database. The metric reflects the level of concurrent activity being handled by the database.
Execute Count	The number of SQL statements executed. The metric provides a measure of workload by indicating how many SQL statements have been run during a specific period.
Running Statements	The number of SQL statements currently being executed by the database. The metric indicates the number of active processing tasks that are consuming resources.
Queued Statements	The number of statements that are waiting to be executed, typically due to resource contention or scheduling within the database. This metric indicates the quantity of work that is in the backlog.
APEX Load Time	The time it takes for an APEX page or application to load and render in the user's browser. This metric can be used to assess the overall usability of your APEX applications from a perceived performance point of view.
APEX Page Event	A count of the APEX page loads, submissions, or processes. This metric indicates the user activity intensity.

Note

Storage utilization in both the DB Ops Console and Retail Home report subscription usage. This usage represents high water mark metrics (and in the case of Retail Home, it's the high water mark per month). A more detailed view of storage consumption can be obtained using the following query in the SQL Commands page of the SQL Workshop in the APEX UI.

```
select owner, cast(sum(bytes) / power(1024,2) as integer) MB from
dba_segments where owner like '%_RDS%' group by owner order by owner
```

You can view a number of descriptive statistics over a variety of time intervals. The available statistics are as follows

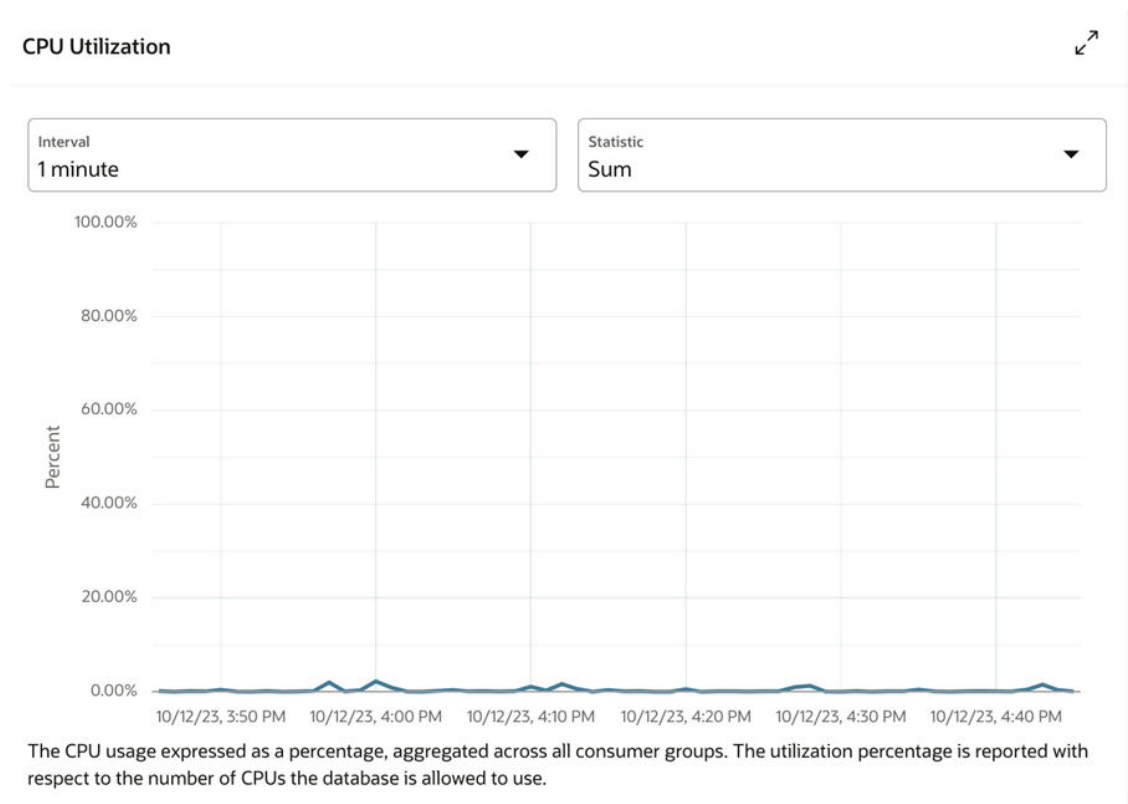
Statistic	Description
Count	Returns the number of observations received in the specified interval.

Statistic	Description
Max	Returns the highest value observed during the specified interval.
Mean	Returns the value of Sum divided by Count during the specified interval.
Min	Returns the lowest value observed during the specified interval.
P50	Returns the estimated value of the 50th percentile during the specified interval.
P90	Returns the estimated value of the 90th percentile during the specified interval.
P95	Returns the estimated value of the 95th percentile during the specified interval.
P99	Returns the estimated value of the 99th percentile during the specified interval.
Rate	Returns the per-interval average rate of change. The unit is per-second.
Sum	Returns all values added together, per interval.

Example screenshots of each metric page are provided in the figures below.

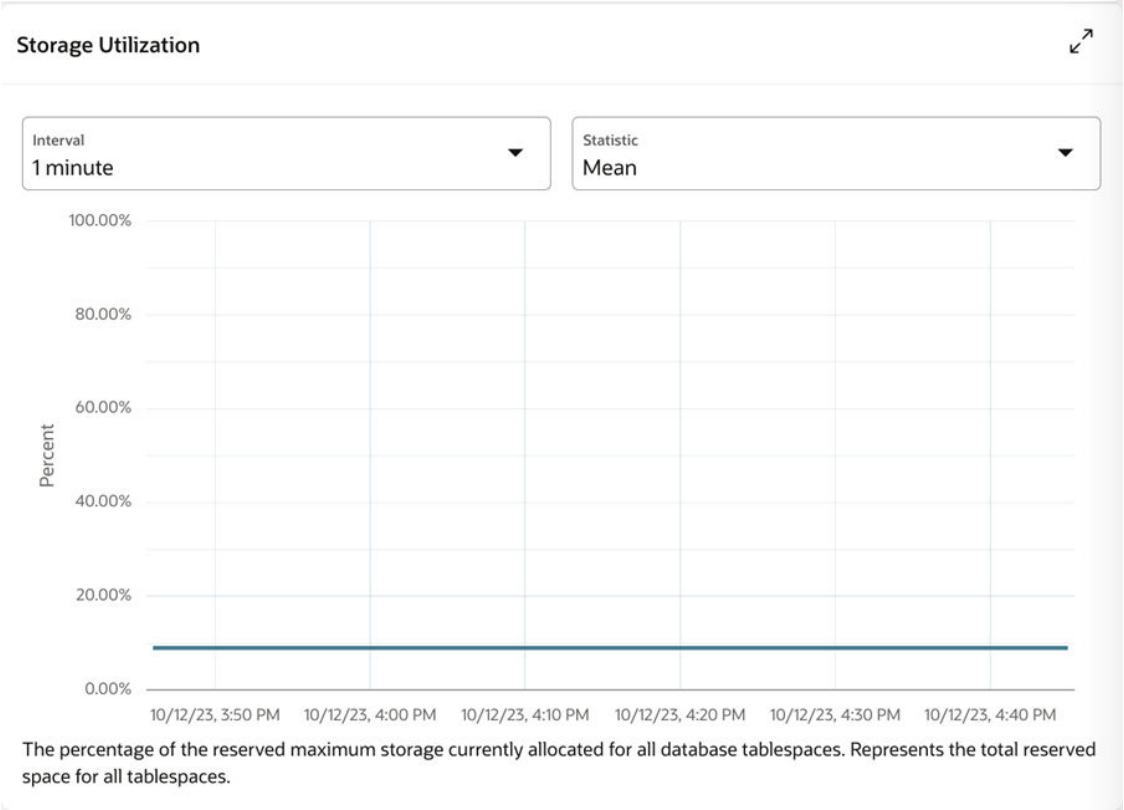
CPU Utilization

Figure 18-12 CPU Utilization



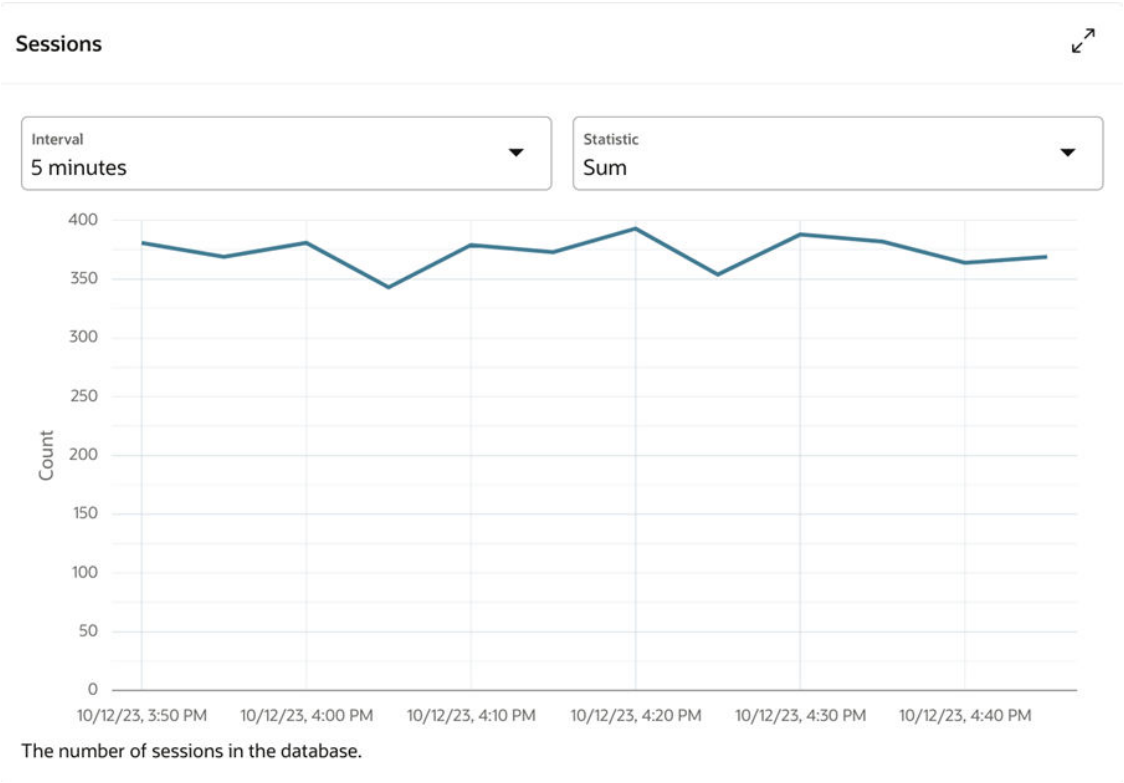
Storage Utilization

Figure 18-13 Storage Utilization



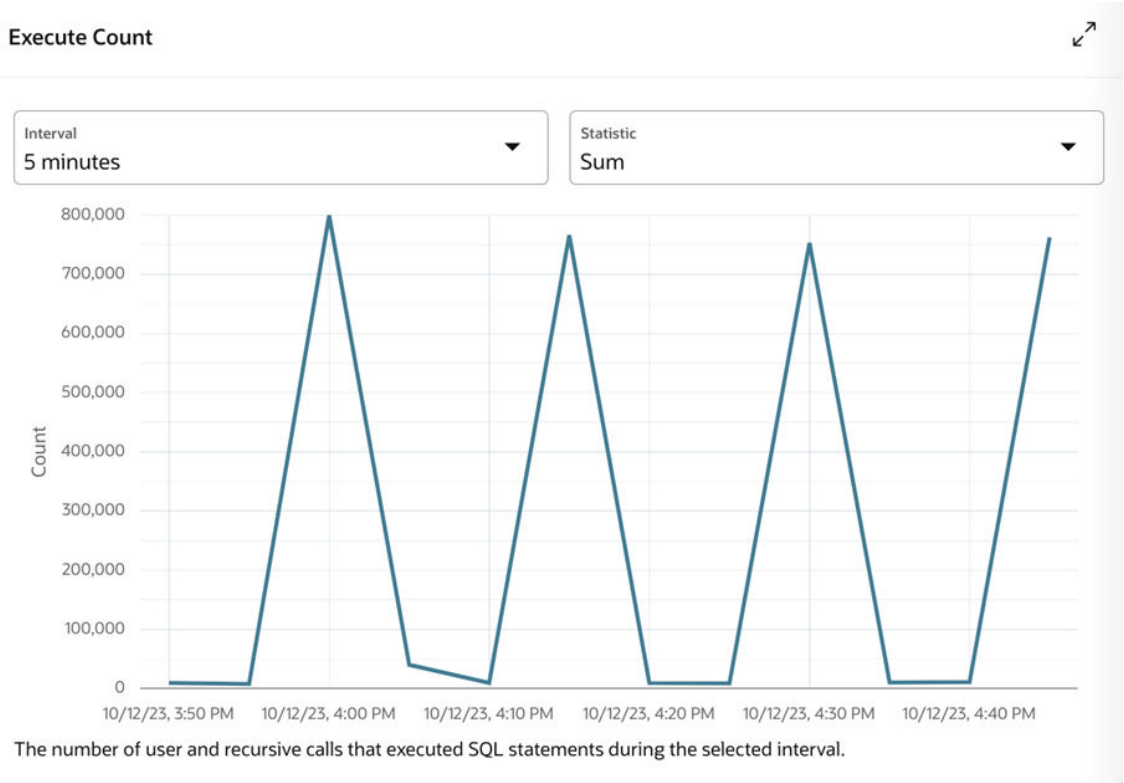
Sessions

Figure 18-14 Sessions



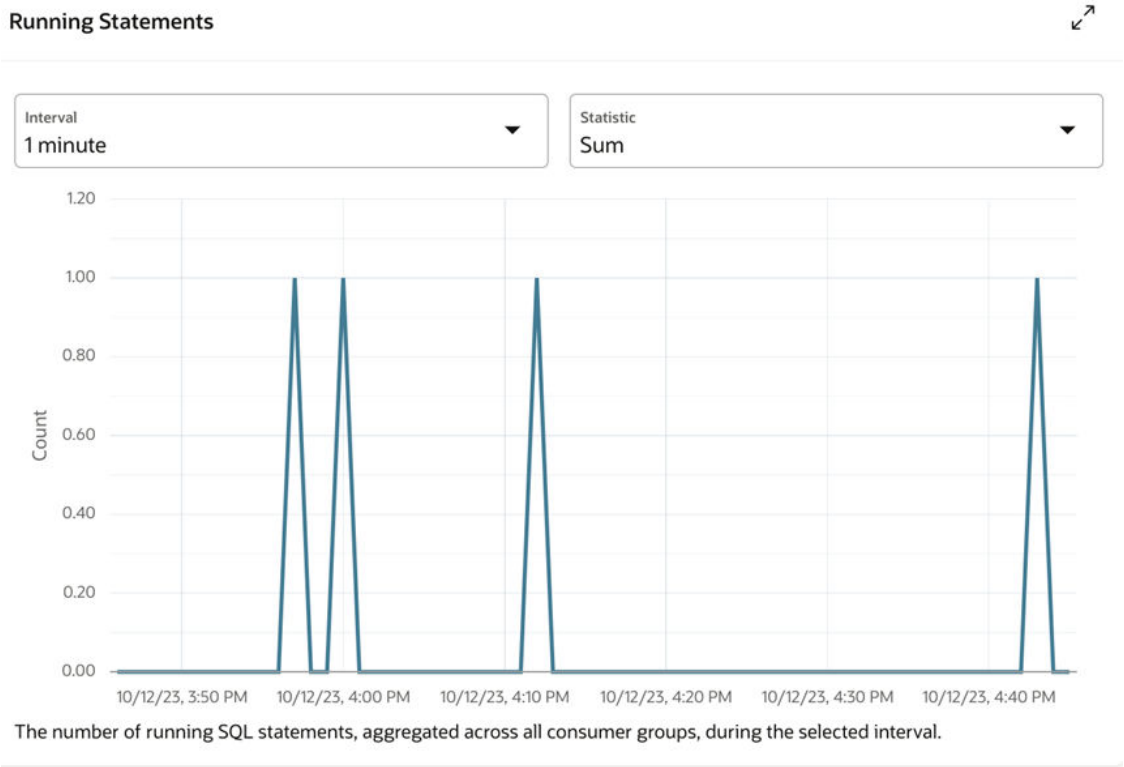
Execute Count

Figure 18-15 Execute Count



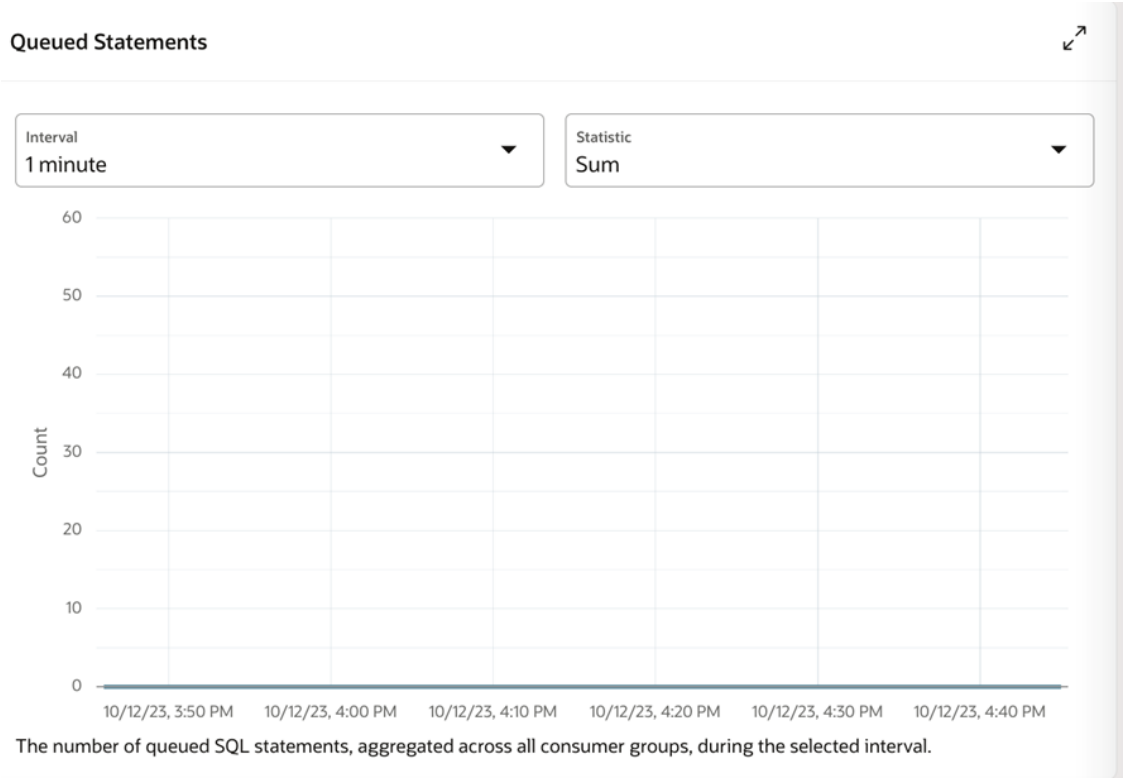
Running Statements

Figure 18-16 Running Statements



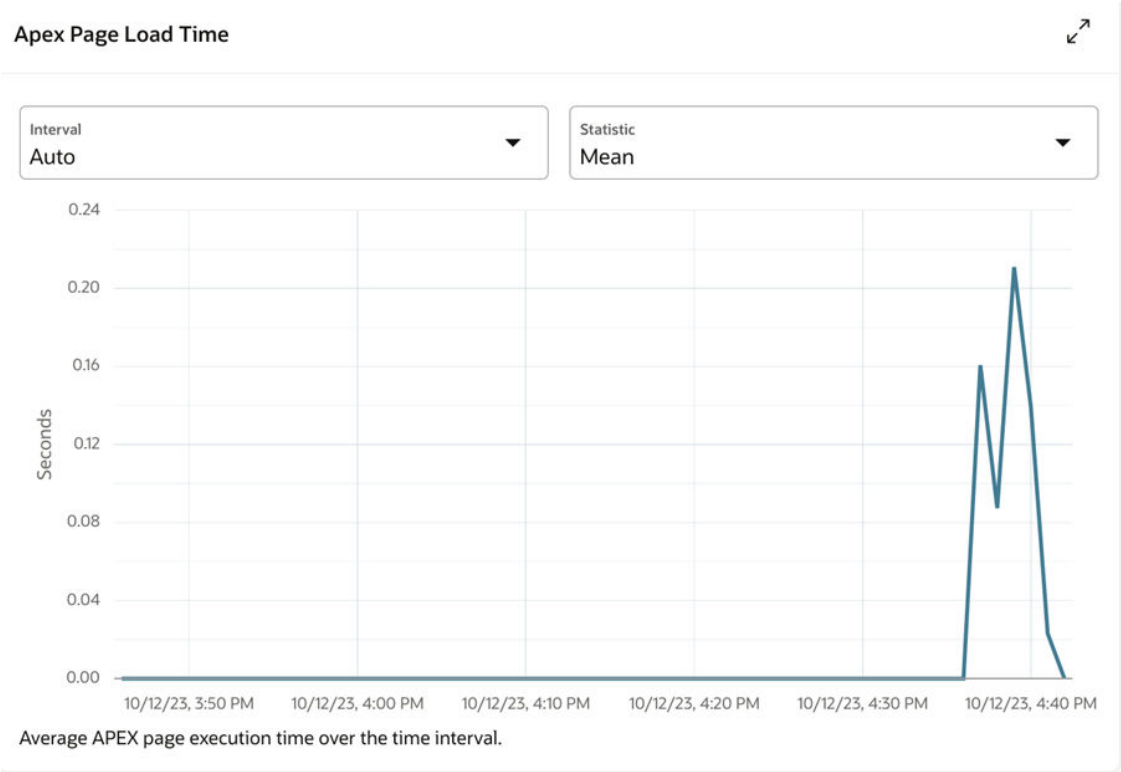
Queued Statements

Figure 18-17 Queued Statements



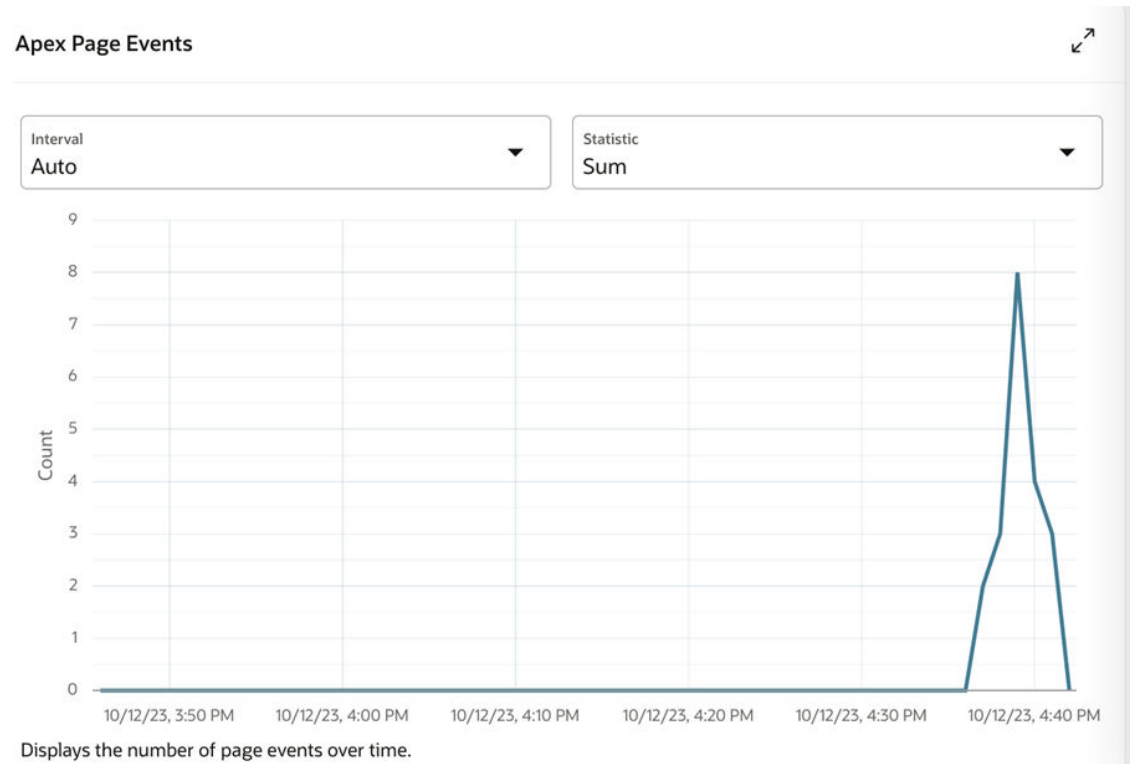
APEX Page Load Time

Figure 18-18 APEX Page Load Time



APEX Page Events

Figure 18-19 APEX Page Events

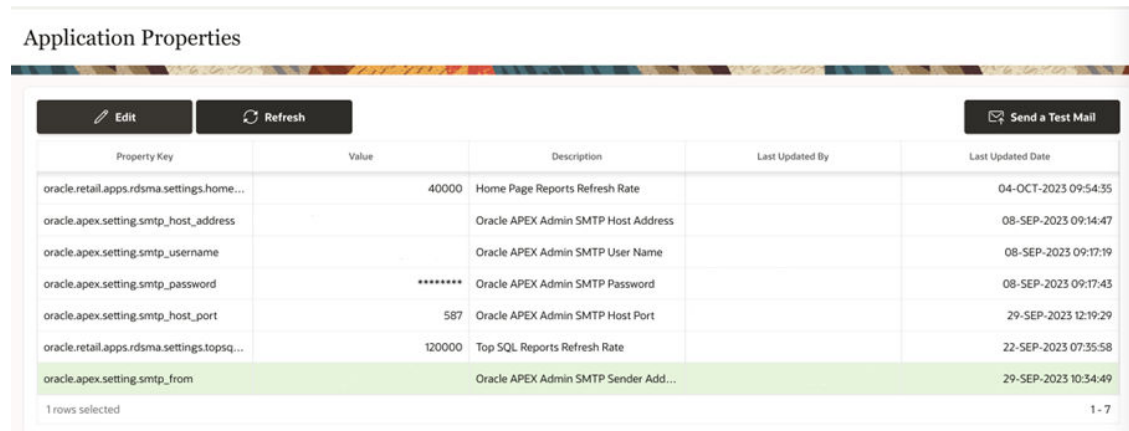


Application Properties

The Application Properties page lists the available application properties including Oracle APEX SMTP settings. The Application Properties are stored as key-value pair, e.g., oracle.apex.setting.smtp_from: me@email.com. This page can be accessed by an 'Administrator' having RDS_MANAGEMENT_ADMINISTRATOR or RDS_MANAGEMENT_ADMINISTRATOR_PREPROD roles.

Note

Roles are synonymous with OCI groups. An assignment to an OCI group may take up to 1 hour to propagate.

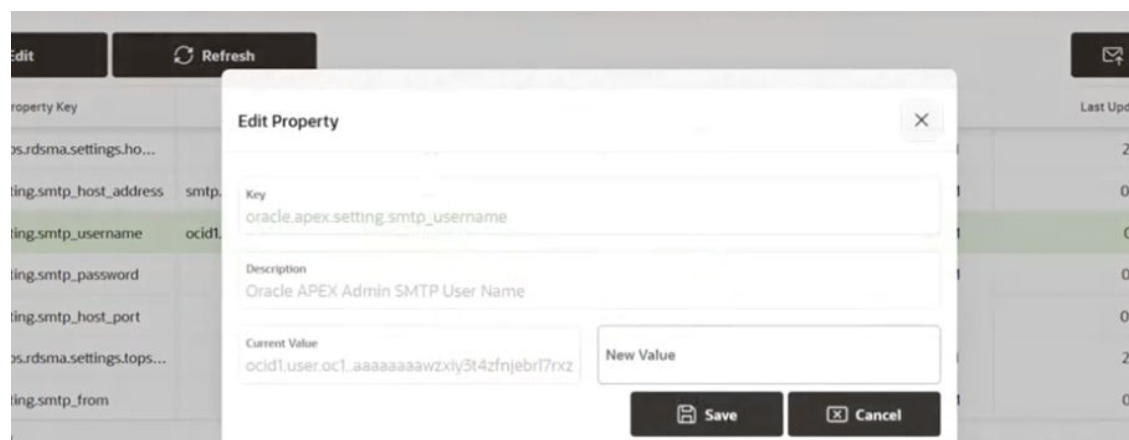
Figure 18-20 Application Properties


The screenshot shows the 'Application Properties' page with a table of application settings. The table has columns for Property Key, Value, Description, Last Updated By, and Last Updated Date. The 'oracle.apex.setting.smtp_from' row is highlighted in green.

Property Key	Value	Description	Last Updated By	Last Updated Date
oracle.retail.apps.rdsma.settings.home...	40000	Home Page Reports Refresh Rate		04-OCT-2023 09:54:35
oracle.apex.setting.smtp_host_address		Oracle APEX Admin SMTP Host Address		08-SEP-2023 09:14:47
oracle.apex.setting.smtp_username		Oracle APEX Admin SMTP User Name		08-SEP-2023 09:17:19
oracle.apex.setting.smtp_password	*****	Oracle APEX Admin SMTP Password		08-SEP-2023 09:17:43
oracle.apex.setting.smtp_host_port	587	Oracle APEX Admin SMTP Host Port		29-SEP-2023 12:19:29
oracle.retail.apps.rdsma.settings.topsq...	120000	Top SQL Reports Refresh Rate		22-SEP-2023 07:35:58
oracle.apex.setting.smtp_from		Oracle APEX Admin SMTP Sender Add...		29-SEP-2023 10:34:49

1 rows selected

The value for the key can be updated using the Edit window.

Figure 18-21 Edit Property


The screenshot shows the 'Edit Property' dialog box. It contains fields for Key, Description, Current Value, and New Value. The 'Key' field is populated with 'oracle.apex.setting.smtp_username' and the 'Description' field is populated with 'Oracle APEX Admin SMTP User Name'. The 'Current Value' field is populated with 'ocid1.user.oc1.aaaaaaaawzxiy3t4zfnjebri7nrxz'. The 'New Value' field is empty. There are 'Save' and 'Cancel' buttons at the bottom.

Session Management

Session management provides a mechanism for managing custom database sessions interactively. The session management UI allows the user to list and terminate database sessions based on predefined criteria – that is, sessions that are nominally (but not guaranteed to be) safe to terminate. Specifically, sessions where the username is ORDS_PUBLIC_USER_DIS and the schemaname is NULL, or where the username or schemaname contains the substring _RDS_CUSTOM. These criteria are formulated to minimize the risk of terminating essential, non-custom sessions. These criteria may, nonetheless, identify essential custom sessions as candidates for termination. Thus, it remains the responsibility of the user to ensure that the termination of a session will yield the desired outcome. It is possible that you may wish to terminate additional sessions that do not fit the above criteria. If so, submit a support request to terminate one or more sessions that are beyond the scope of the session management UI. Note that, even when terminating one or more sessions through a support request, you are responsible for the consequences of termination.

Refer to the “Post Installation Configuration” chapter of the *Retail Data Store Security Guide* for details on the OCI groups needed to unlock Session Management capabilities.

1. To start managing sessions, tap the Database Sessions button in the navigation side bar (see [Figure 18-22](#)). Database sessions will open on the right side of the parent panel (see [Figure 18-23](#)). Session Management enables the user to:
 - View a list of running database sessions (see [Figure 18-24](#))
 - End a running database session from the list by clicking end Database Session button:
 - enabled/visible for users with the RDS_MANAGEMENT_ADMINISTRATOR role
 - available on each row at the last column of the interactive report
 - Perform a “Refresh on Demand” with the Refresh Button of the Active Database Sessions
 - disabled/not visible for users without administrator role

Figure 18-22 Navigation to Database Sessions Page

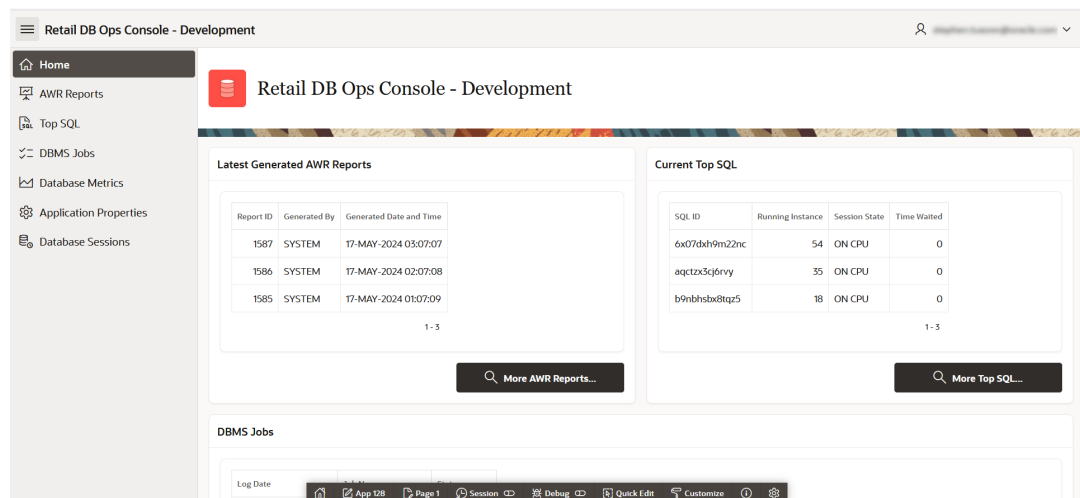


Figure 18-23 Database Sessions Page

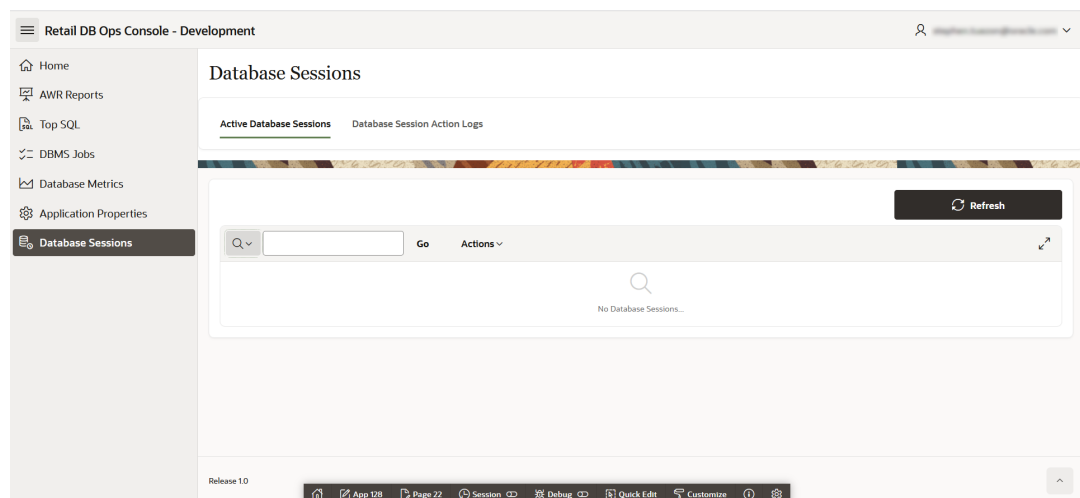
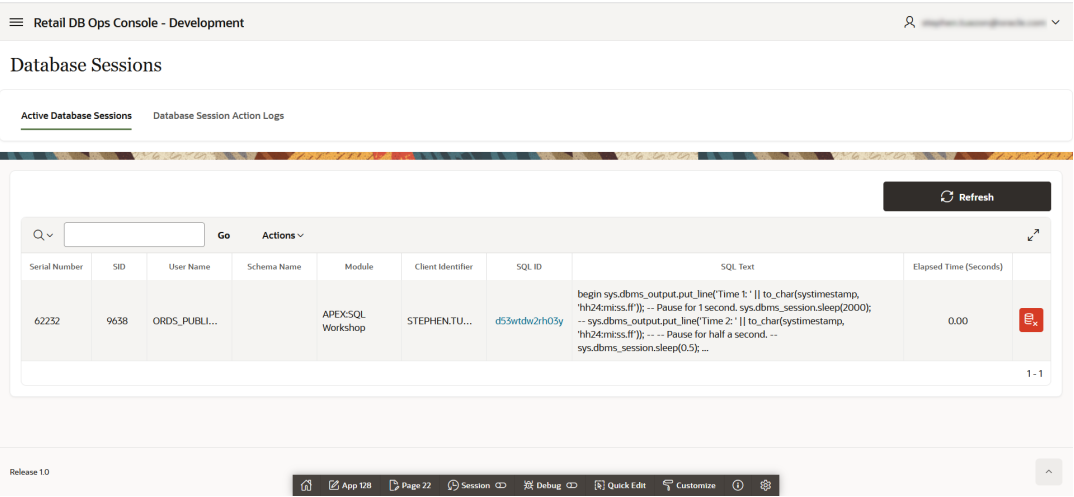


Figure 18-24 Active Sessions



- 2. Use the filter functionality under the Actions Menu on the interactive report to filter the results presented on the page (see [Figure 18-25](#) and [Figure 18-26](#)). Available filters are: Serial Number, SID, User Name, Schema Name, Module, Client Identifier, SQL ID and SQL Text.

Figure 18-25 Filtering

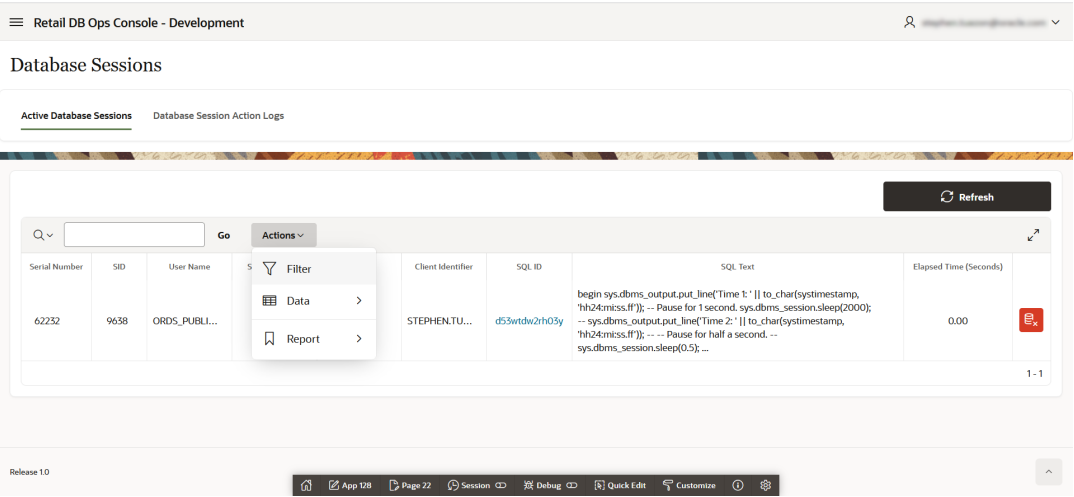
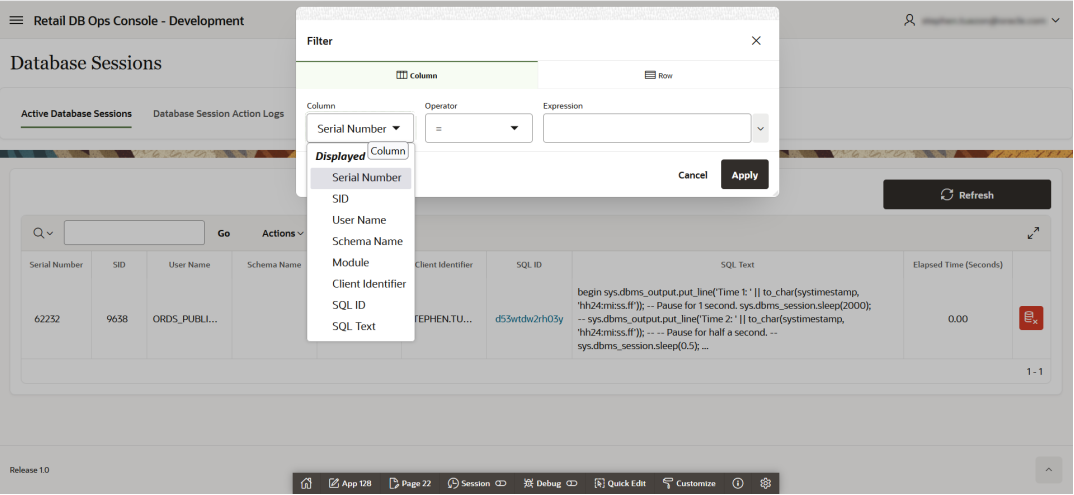


Figure 18-26 Filtering Example



3. To end a database session, tap the End Database Session button (see [Figure 18-27](#)) in the list of database sessions (in the last column of the row). Then tap the End Session button (or tap Cancel) on the pop-up dialog (see [Figure 18-28](#)). This will end the database session and reload the Active Database Sessions Interactive Report. The ended session is removed from the list of Active Database Sessions in the Interactive Report when reloaded.

Figure 18-27 End Database Session

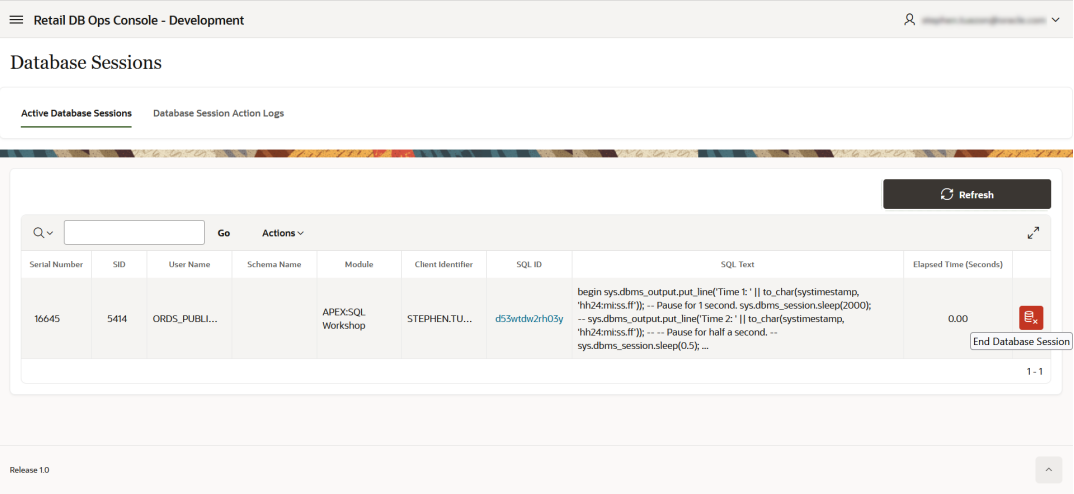
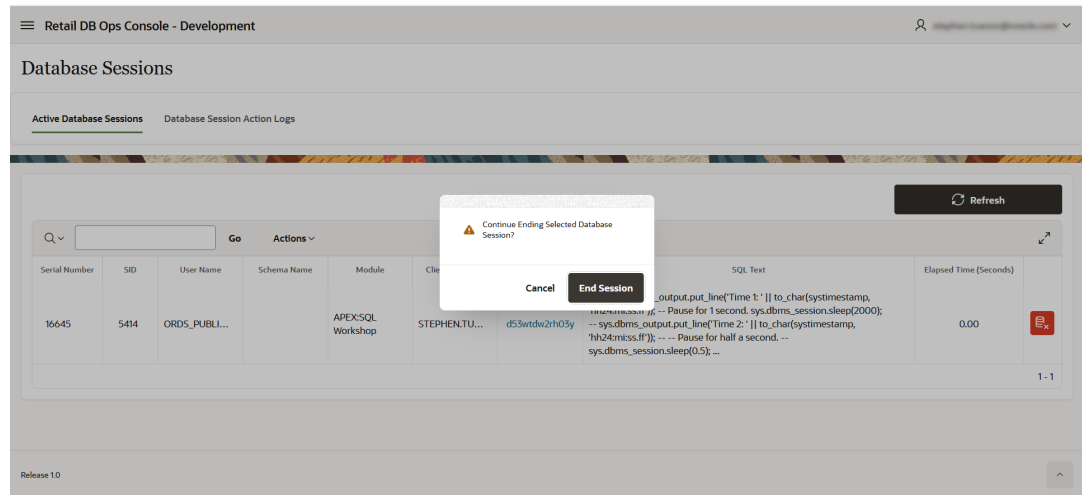
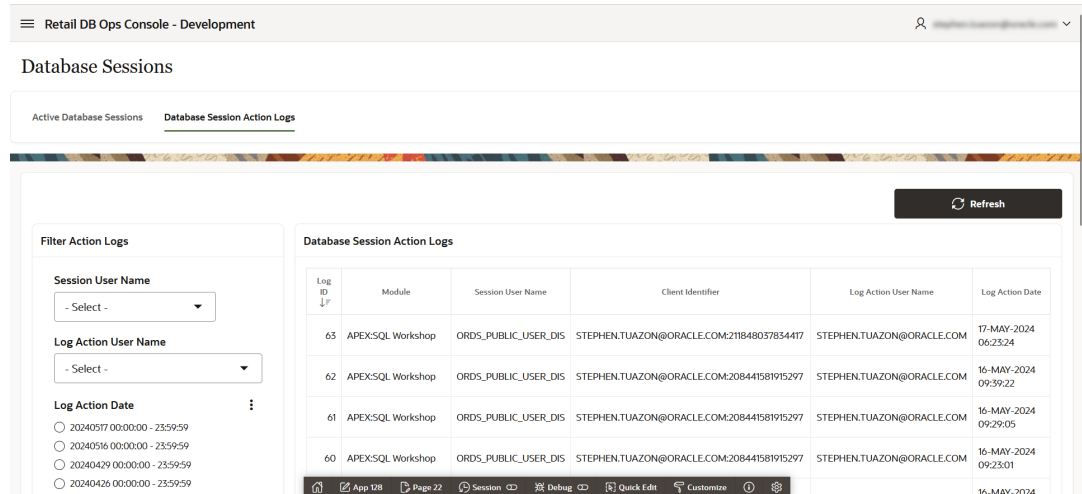


Figure 18-28 End Database Session Pop-Up

4. The Database Session Action Logs (see [Figure 18-29](#)) enables a user with the administrator role to:
- View the logs with details of ended sessions
 - Refresh the action log with the Refresh Button
 - Filter logs using Session User Name, Log Action User Name, and Log Action Date.

Figure 18-29 Action Logs

Session Management API

RDS also provides a session management API for managing custom database sessions programmatically. Programmatic termination of sessions may be called for when the number of sessions that need to be terminated would make interactive termination burdensome.

This API consists of a package containing procedures and functions to list and terminate database sessions based on predefined criteria – that is, sessions that are nominally (but not guaranteed to be) safe to terminate. Specifically, sessions where the username is `ORDS_PUBLIC_USER_DIS` and the schemaname is `NULL`, or where the username or schemaname contains the substring `_RDS_CUSTOM`. These criteria are formulated to minimize

the risk of terminating essential, non-custom sessions. These criteria may, nonetheless, identify essential custom sessions as candidates for termination. Thus, it remains the responsibility of the API user to ensure that the termination of a session will yield the desired outcome.

It is possible that you may wish to terminate additional sessions that do not fit the above criteria. If so, submit a Support Request to terminate one or more sessions that are beyond the scope of the session management API. Bear in mind, even when terminating one or more sessions through a support request, you are responsible for the consequences of termination.

Package Overview

The package `MANAGEMENT.RDS_SESSION_MGMT` contains the following procedures and functions:

- `LIST_RDS_SESSIONS` (`O_session_list` OUT SYS_REFCURSOR): This procedure returns a list of database sessions that meet the criteria described above.
- `KILL_RDS_SESSION` (`I_sid` IN NUMBER, `I_serial` IN NUMBER, `O_status` OUT VARCHAR2): This procedure terminates a session based on the input SID (System Identifier) and SERIAL. It first validates that the session meets the RDS criteria and then looks up the instance ID to terminate the session.
- `KILL_RDS_SESSION` (`I_audsid` IN NUMBER, `O_status` OUT VARCHAR2): This procedure terminates sessions based on the input AUDSID (Audit Session Identifier). It retrieves the SID, SERIAL, and instance ID based on the AUDSID and then validates and terminates the session if it meets the RDS criteria.

Procedure and Function Details

LIST_RDS_SESSIONS

This procedure opens a cursor to return sessions that meet the RDS criteria.

It selects sessions from the `gv$session` view, filtering based on the username, schemaname, and custom patterns.

KILL_RDS_SESSION (SID and SERIAL)

This procedure first validates that the session with the given SID and SERIAL meets the RDS criteria. If the session is valid, it retrieves the instance ID for the session and then executes an `ALTER SYSTEM` statement to terminate the session.

The status of the operation is returned in the `O_status` OUT parameter.

KILL_RDS_SESSION (AUDSID)

This procedure retrieves the SID, SERIAL, and instance ID based on the input AUDSID. It then validates the session and, if valid, terminates the session using an `ALTER SYSTEM` statement. The status of the operation is returned in the `O_status` OUT parameter.

19

Sending Email from APEX

To enable sending email from APEX, you must have access to an SMTP server. Only Oracle's OCI SMTP server is supported. To send mail, you will need to provide values for the following Application Properties (see Section 7.19.6 "Application Properties" for details on setting property values).

To enable sending email from APEX, you must have access to an SMTP server. Only Oracle's OCI SMTP server is supported. To send mail, you will need to provide values for the following Application Properties (see the [Application Properties](#) in the [Retail DB Ops Console](#) chapter for details on setting property values).

```
oracle.apex.setting.smtp_host_address  
oracle.apex.setting.smtp_username  
oracle.apex.setting.smtp_password  
oracle.apex.setting.smtp_host_port  
oracle.apex.setting.smtp_from
```

Your settings can be verified by sending a test email by clicking the **Send a Test Mail** button (see the figure below). Once you have verified your settings, you can use the APEX_MAIL package to send emails from Oracle APEX applications.

The APEX_MAIL_QUEUE will contain any queued email messages. The APEX_MAIL_LOG will contain the disposition of sent mail. Specifically, it will indicate whether an email message was sent successfully or not. See APEX_MAIL package documentation for additional details.

Note

To send email from your RDS ADW instance using the APEX_MAIL package, you must provision an OCI SMTP server. Only Oracle's OCI SMTP server is supported for sending email from RDS ADW.

Figure 19-1 Sending a Test Email

The image shows a web application interface with a modal dialog box titled "Send Test Mail". The dialog has a close button (X) in the top right corner. It contains four input fields: "Sender", "Recipient", "Subject", and "Message Body". The "Message Body" field is a larger text area. At the bottom right of the dialog, there are two buttons: "Send" (with a paper plane icon) and "Cancel" (with an X icon). The background of the web application is partially visible, showing a table with columns like "Key", "a.settin", "p_hos", "p_use", "p_pas", "p_hos", "a.settin", and "p_fron".

Monitoring Replication Lag

RDS uses Oracle GoldenGate to replicate application data from Retail applications to its ADW instance. The data is kept in sync with the source application as changes are made to the source. You can monitor the lag between source (application) and target (RDS) systems using the following GoldenGate heartbeat tables and views:

- GGADMIN.GG_HEARTBEAT_SEED
- GGADMIN.GG_HEARTBEAT
- GGADMIN.GG_HEARTBEAT_HISTORY
- GGADMIN.GG_LAG
- GGADMIN.GG_LAG_HISTORY

Refer to [Oracle GoldenGate Integrated Heartbeat](#) for details on how to use these tables and views to assess replication lag. Note that GoldenGate heartbeat has already been enabled in RDS.

Incoming Path	Product	Schema	Workspace
BC_* ==> BCREP	Brand Compliance Cloud Service	BC_RDS_CUSTOM	Yes
CE_* ==> CEREP	Customer Engagement Cloud Service	CE_RDS_CUSTOM	Yes
I1_* ==> I1REP	Retail Integration Cloud Service	RICS_BDI	No
I2_* ==> I2REP	Retail Integration Cloud Service	RICS_RFI	No
I3_* ==> I3REP	Retail Integration Cloud Service	RICS_RIB_EXT1	No
I4_* ==> I4REP	Retail Integration Cloud Service	RICS_RIB_LGF1	No
I5_* ==> I5REP	Retail Integration Cloud Service	RICS_RIB_ROB1	No
I6_* ==> I6REP	Retail Integration Cloud Service	RICS_RIB_RWMS1	No
I7_* ==> I7REP	Retail Integration Cloud Service	RICS_RIB_SIM1	No
I8_* ==> I8REP	Retail Integration Cloud Service	RICS_RIB_TAFR1	No
I9_* ==> I9REP	Retail Integration Cloud Service	RICS_USM1	No
IN_* ==> INREP	Retail Integration Cloud Service	XO_RDS_CUSTOM	Yes
MF_* ==> MFREP	Merchandising Foundation Cloud Service	MFCS_RDS_CUSTOM	Yes
OB_* ==> OBREP	Order Broker Cloud Service	OB_RDS_CUSTOM	Yes
OM_* ==> OMREP	Order Administration Cloud Service	OM_RDS_CUSTOM	Yes
SI_* ==> SIREP	Store Inventory Operations Cloud Service	SIOCS_RDS_CUSTOM	Yes
XO_* ==> XOREP	Xstore Office Cloud Service	XO_RDS_CUSTOM	Yes

Notification-Based Monitoring

POM can be used in combination with the data in RDS to establish automated process monitoring. A POM batch job can be setup to call a RDS data service to generate and send the Alert to the associated application such as Merchandising.

This document describes the steps needed to setup such an automated process. These steps are:

1. Setup a notification type in Retail Home
2. Implement a RESTful service in RDS
3. Setup a job in POM to invoke that service

RDS-POM integration will be illustrated for an alert that returns the counts of stock counts that are open for more than seven days.

Setting up the Notification Type

Typically, an alert/notification results from the monitoring activity. As a first step, a notification type associated with this monitoring activity needs to exist or be setup in Retail Home. Refer to Retail Home documentation for details on how to create a notification type. For the example at hand the notification type is MerchStockCountAlert and is setup for the MFCS application.

Implementing a RESTful Service in RDS

The first step is to create the data service boiler plate, which consists of the following:

- creating a module
- creating a URI template, and lastly
- creating a POST handler

The steps below describe the implementation of an open stock count RESTful service suitable for integration with POM.

The alert service must conform to the POM specification. There are two relevant specifications. The first concerns the format of the JSON body in the POST, which is shown in the following table. The alert service need not use any of the details in the body of the POST message.

Endpoint to start a job.
Method: **POST**
Body:

Attribute	Description
cycleName	Name of the Cycle - Nightly, Hourly_1, Adhoc etc.

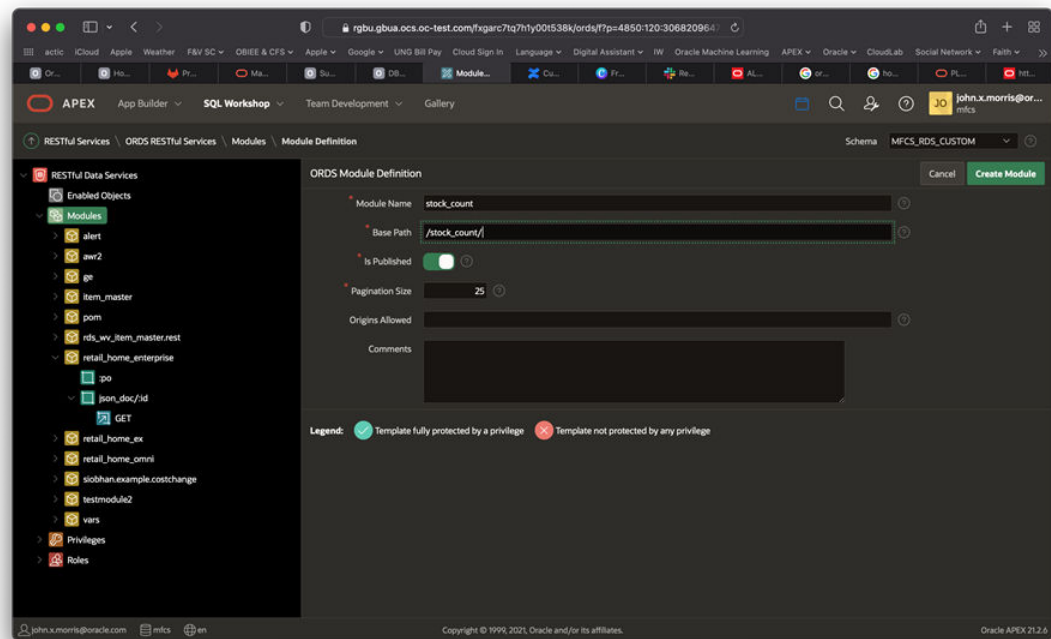
Attribute	Description
flowName	Name of the Flow.
processName	POM process name
jobName	POM job name
agentExecutionId	A unique ID assigned by POM for every job run (Re-run of a failed job would have a different id compared to the initial run)
parameters	Job Parameters (Pipe delimited key-value pairs -- hey1=value1 key2=value2)

The second specification concerns the response returned. If the *notification* object is not null, the content will be sent as a notification by POM.

Attribute	Description
executionId	Unique Id returned by the target app to POM for status tracking
executionInfo	Any additional info the target app would like to share with POM
notification	This is an optional entry that can contain the following attributes. The <i>url</i> , <i>type</i> , and <i>severity</i> fields are optional. <pre>"notification": { "info": "<message>", "url": null, "type": "ErrorNotification", "severity":1 }</pre>
status	Submitted/ Running/Error/Completed

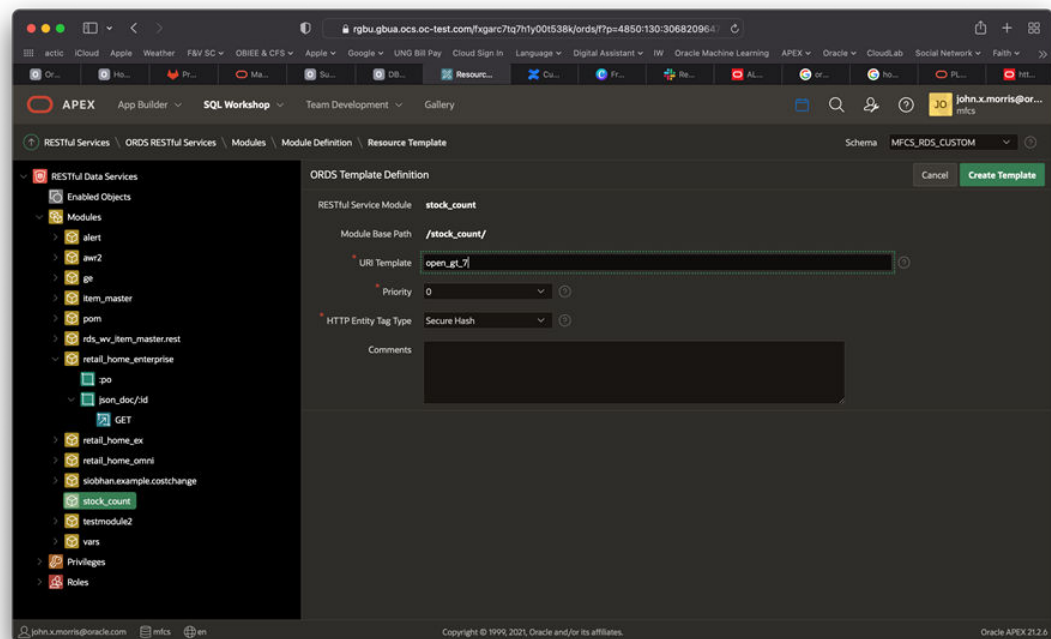
Create a Stock Count Module

The first step is to create a module. Module is a hierarchical organizing construct. The user may have multiple services associated with it or just one. The following screen shot illustrates what the create module screen looks like prior to creating the module. Note the module name is `stock_count` and the base path is `/stock_count/`.



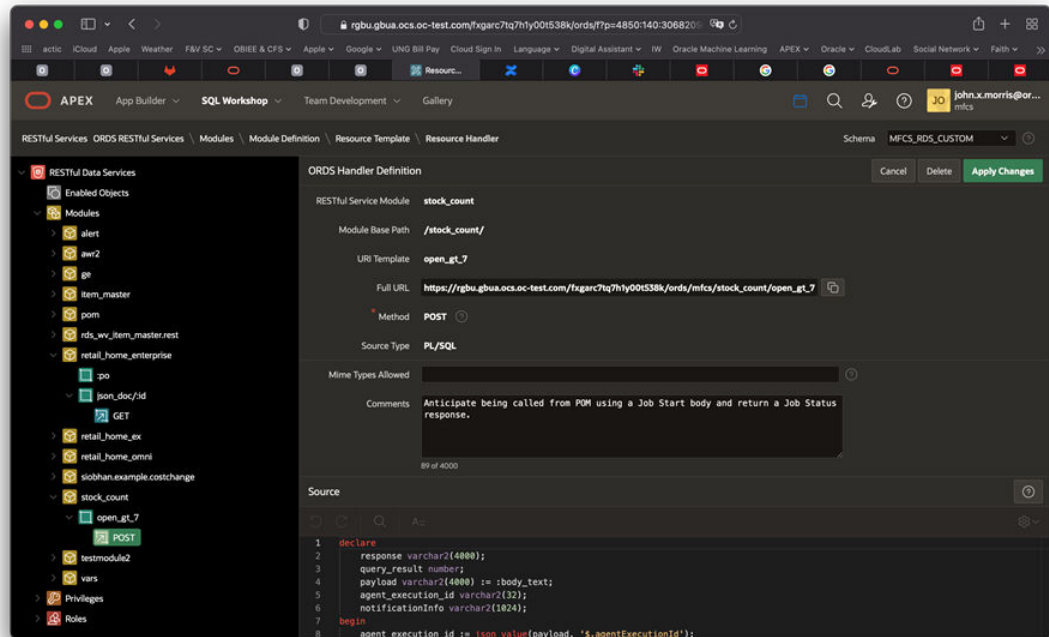
Create a URI Template

Once the module is created, create the URI template. Note the URI template is set to `open_gt_7`, i.e., stock counts open for more than seven days.



Create a POST Handler

The last step is to create a POST handler. Note the method is POST and the source type is PL/SQL.



The source of the example alert service is in Listing 1. The body, a sample of which is shown in Listing 2, of the alert service invocation is accessed using the `:body_text` implicit parameter -- a bind variable. This variable can only be read once; so, retrieve it and store it in a variable. Unpacking the payload and crafting a response will be common tasks for most RESTful services. The example below illustrates some but not all that one might encounter. Bear in mind, the example below does not show any error handling. Experimentation and experience will determine what level of error handling is warranted. Listing 3 shows a response.

```
declare
  -- The :body_text bind variable is an implicit parameter.
  -- It can only be read once, so it is captured in a variable
  -- called payload.
  payload varchar2(4000) := :body_text;
  -- The result of the stock count query
  query_result number;
  -- Note that varchar2 is set to the maximum. If the response
  -- could exceed 4000 characters, a CLOB would be needed and
  -- one could not use http.print directly.
  response varchar2(4000);
  -- The response will return the agentExecutionId in
  -- response as the executionId.
  agent_execution_id varchar2(32);
  -- The notification info destination.
  notificationInfo varchar2(1024);
begin
  -- Get the agent execution id from the payload (i.e., body text).
```

```

agent_execution_id := json_value(payload, '$.agentExecutionId');
-- Get a count of stock counts that have been open for more than
-- 7 days and put it in query result.
SELECT count(1)
    into query_result
    FROM rds_wv_stake_prod_loc spl,
         rds_wv_stake_head s
    WHERE s.stocktake_date BETWEEN sysdate - 7 AND sysdate
    AND s.delete_ind = 'N'
    AND spl.cycle_count = s.cycle_count
    AND s.stocktake_type = 'B'
    AND spl.processed != 'S'
    AND spl.cycle_count = s.cycle_count;
-- Craft notificationInfo as a human readable string.
notificationInfo := 'Number of stock counts that have been ' ||
    'open for more than 7 days is ' || to_char(query_result) || '.';
-- Craft the required JSON response. There is no executionInfo.
select json_object('status' value 'success',
    'executionInfo' value '',
    'executionId' value agent_execution_id,
    'notificationInfo' value notificationInfo) into response from
dual;
-- Output the response - note http.print is used here. http.print
-- only supports varchar2 so another approach would be needed
-- if the response is likely to exceed 4000 characters. This could
-- if error handling (not shown) were to return a stack trace.
http.print(response);
end;
```

Listing 1: An Alert service conforming to POM Requirements

A sample payload for the above job is:

```

{
  "cycleName": "cycle1",
  "flowName": "flow1",
  "processName": "process1",
  "jobName": "job1",
  "agentExecutionId": "agentExecutionId1234",
  "parameters": ""
}
```

Listing 2: A Sample Payload

The response of the above service, given the payload in Listing 2 is:

```

{
  "status": "success",
  "executionInfo": null,
  "executionId": "agentExecutionId1234",
  "notificationInfo": "Number of stock counts that have been open for more
than 7 days is 0."
}
```

Listing 3: A Sample Response

Setting up the POM Job

The user now needs to setup an Ad Hoc batch job in POM which calls the RESTful service described above. Adding such a job is done through the spreadsheet as described in the next section. The user then uploads the spreadsheet to POM then schedules to run at the desired time.

When the job executes, it will invoke the RESTful service which will return a response with notification content. POM will then send a notification to the associated application based on the designated notification type. The notification will contain the notification content returned from the RESTful service.

Note that based on the 300 second limitation on service duration described in [Limits on Service Initiated Queries and PL/SQL Blocks](#), the POM job setup according to the specification in this section has a 300 second execution limit. When this limit is reached, the Job in POM will fail with a timeout error.

Adding the Ad hoc Job in Batch Schedule Spreadsheet

Entries as shown below as an example need to be added in the specified tabs of the batch schedule spreadsheet for every Ad hoc job created for an alert.

Figure 21-1 Process Tab

ProcessName	Description	ApplicationName	DependencyType	AdhocInd
MERCH_STOCK_COUNT_ALERT_PROCESS	Merch Stock Count Alert Process	RMS	Time	Y

- **ProcessName** – Add a unique process name in upper case with no spaces. Use an underscore if needed. It should end with XXX_PROCESS.
- **Description** – Short description of the process without any special characters.
- **Application Name** – Mention the application name where the batch process belongs to. For example, MFCS
- **DependencyType** – This needs to be set to 'Time'.
- **AdhocInd** – It will be 'Y' as we are creating an ad hoc job here.

Figure 21-2 Job Tab

JobName	Description	RmsBatch	ScriptFolder	RmsWrapper	ParameterValue	ApplicationName	Module	FixedPa	ParameterU	SkipOnError	JobType
MERCH_STOCK_COUNT_ALERT_JOB	Merch Stock Count Alert job				notificationType=MerchStockCountAlert[[jobrestfulServiceModule=mfcs/stock_cou nt/open_gt_7	RMS		Y	N	Y	RDS

- **JobName** – Unique job name for each alert in upper case only with no spaces. Use an underscore if needed. It should end with XXX_JOB.
- **Description** – Short description of the job without any special characters.
- **RmsBatch, ScriptFolder and RmsWrapper** – Irrelevant for alerts and can be left empty.
- **ParameterValue** – This holds the two parameters needed to identify the notification type and the RDS endpoint. For the example at hand, these are:

- notificationType: This is the notification type setup in Retail Home as described in the Setting Up a Notification Type section at the top of this document: MerchStockCountAlert.
- restPath : This points to the endpoint defined above: mfcs/stock_count/open_gt_7.
- These need to be separated by a double pipe as depicted in the Job tab screen shot above.
- ApplicationName – Mention the same application name entered in Process tab. Here, it's RMS.
- Modules – Job can be associated with a module of the application.
- Job Type – The job type associated with RDS alerts is RDS.

Figure 21-3 ProcessJobMapping

ProcessName	JobName	DayOfTheWeek
MERCH_STOCK_COUNT_ALERT_PROCESS	MERCH_STOCK_COUNT_ALERT_JOB	

- To map the new process and jobs (it's one-to-one for adhoc jobs), enter the created ProcessName, JobName and the day(s) of the week on which the specific Process/Job needs to be run. If the Job will run on daily basis, leave 'DaysOfTheWeek' column blank.

Figure 21-4 Schedule

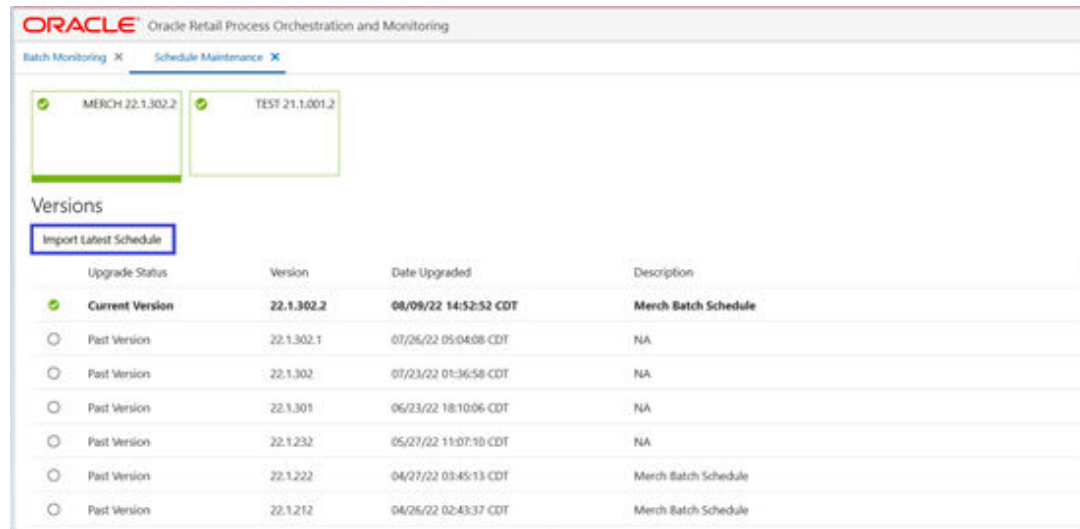
ScheduleName	Description	Version
MERCH	Merch Batch Schedule	22.1.302.2

Ensure that the 'Version' is updated to a version greater than the 'Current Version' in POM Application. In this example, the version should be changed on the spreadsheet to 22.1.302.2.1 or 22.1.302.3.

Uploading the Batch Schedule Spreadsheet in POM

1. Log in to POM UI and navigate to Tasks -> Schedule Maintenance.
2. Select the Application Scheduler tile and click 'Import Latest Schedule' button.
3. Upload the spreadsheet containing the new Adhoc job for alert.

Figure 21-5 Upload Batch Schedule

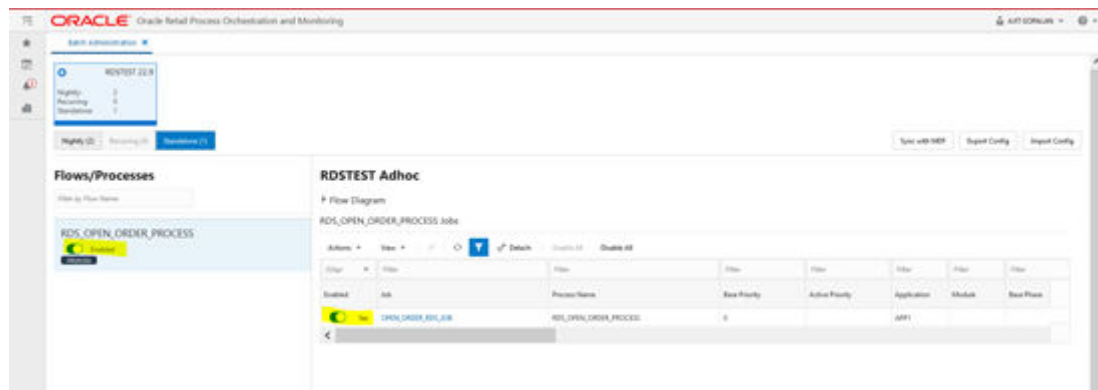


Upgrade Status	Version	Date Upgraded	Description
☒ Current Version	22.1.302.2	08/09/22 14:52:52 CDT	Merch Batch Schedule
☐ Past Version	22.1.302.1	07/06/22 05:04:08 CDT	NA
☐ Past Version	22.1.302	07/23/22 01:36:58 CDT	NA
☐ Past Version	22.1.301	06/23/22 18:10:06 CDT	NA
☐ Past Version	22.1.232	05/27/22 11:07:10 CDT	NA
☐ Past Version	22.1.222	04/27/22 03:45:13 CDT	Merch Batch Schedule
☐ Past Version	22.1.212	04/26/22 02:43:37 CDT	Merch Batch Schedule

Enable the Newly Added Process/Job

Note that any new job introduced through the spreadsheet will be added in the disabled state. Enable it using the Batch Administration screen.

Figure 21-6 Batch Administration Screen



Enabled	Job	Process Name	Base Priority	Active Priority	Application	Module	Base Phase
☒	RDS_OPEN_ORDER_PROCESS	RDS_OPEN_ORDER_PROCESS	0		JOFF		

Starting/Restarting the Scheduler Day

You will need to start a new scheduler day or restart the existing scheduler day on the Batch Monitoring screen for the new changes to take effect in the next batch run. See the Batch Monitoring screen shot below.

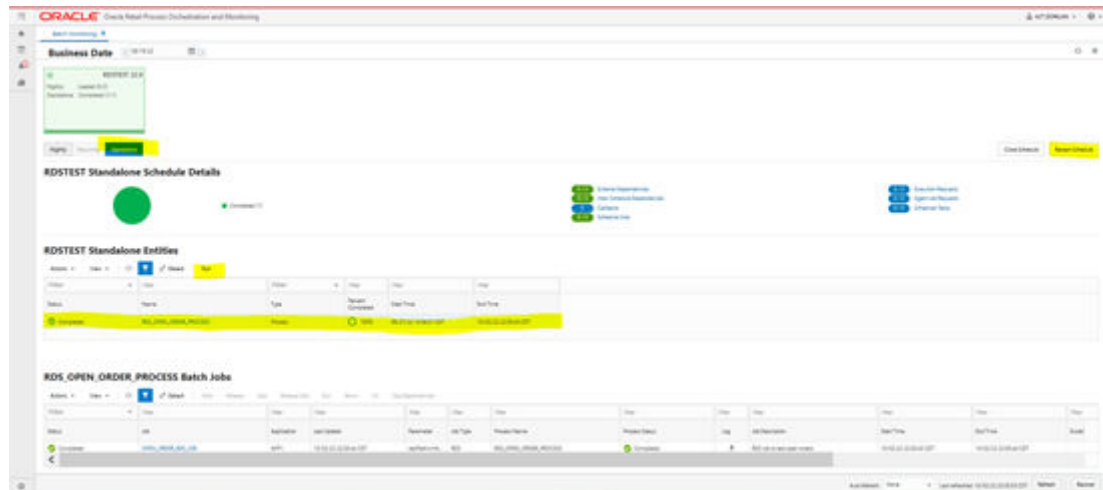
You can now run the Alert job in POM in one of two ways:

1. Direct run through the Batch Monitoring screen or
2. Schedule it to run using the POM Scheduler Administration screen.

Direct Run

On the Batch Monitoring screen, select the Standalone tab below the tile area. Then select the previously added RDS Alert process in the Standalone Entities table and click on the run action button above the table.

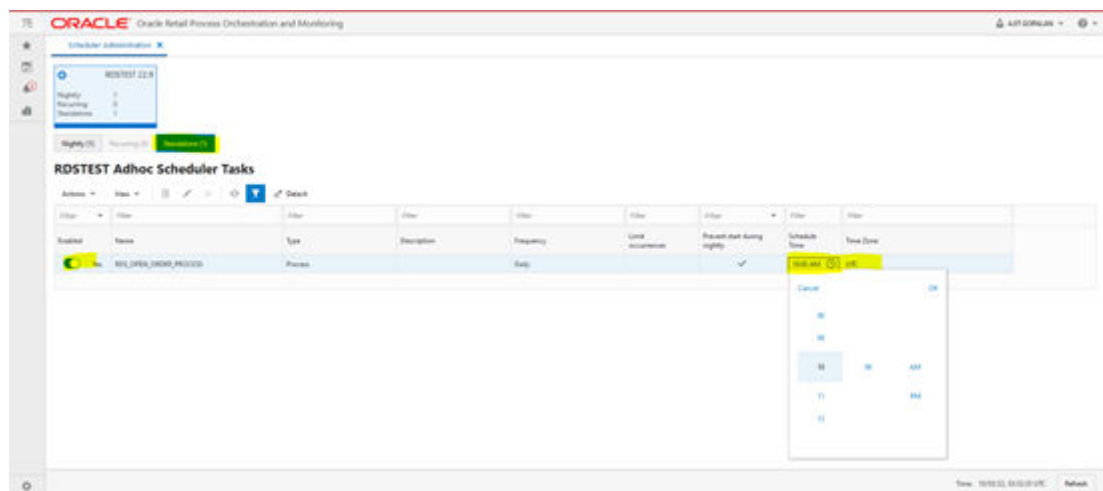
Figure 21-7 Batch Monitoring



Schedule through the Scheduler Administration Screen

You can use the POM Scheduler Administration screen to schedule the newly added job to run as frequently as needed.

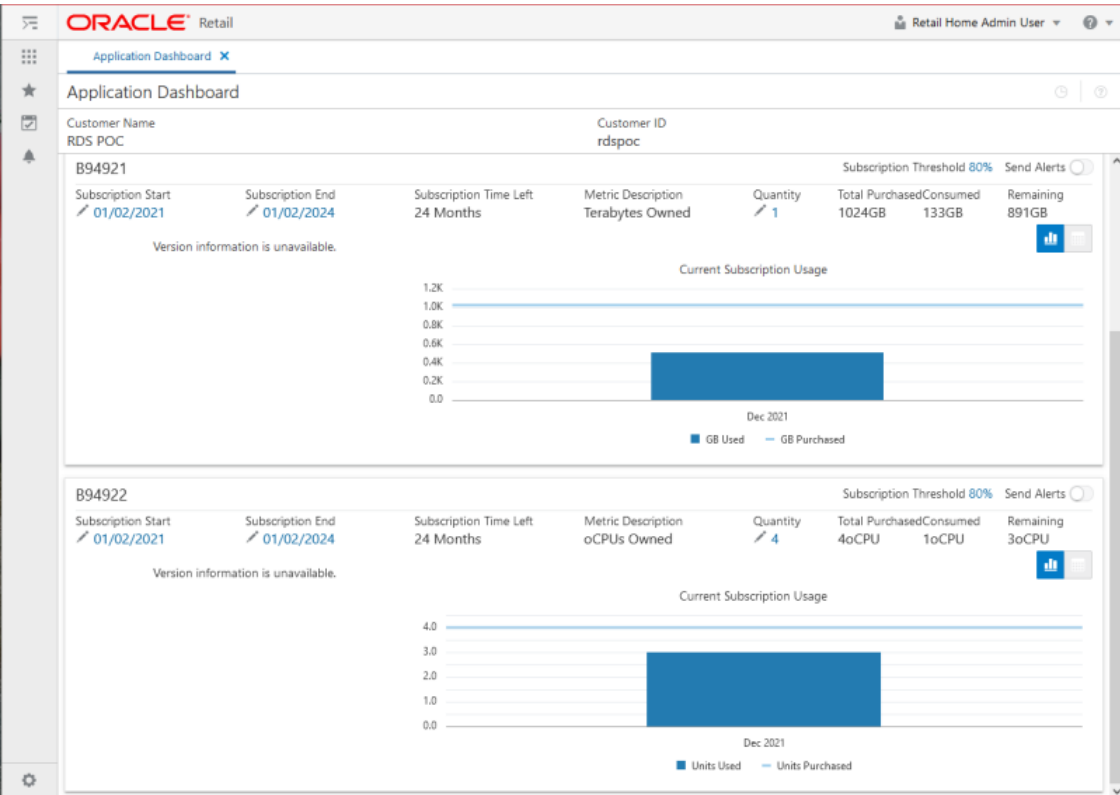
Figure 21-8 Scheduler Administration



Storage and CPU Usage

For each RDS instance, the database disk storage and CPU usage is tracked. Usage can be seen by logging in to Oracle Retail Home and viewing the Application Dashboard. On the list are two entries: one for RDS CPU Usage, and one for RDS Disk Usage. The entries show current usage and display the currently subscribed amounts for CPU and storage, so a customer can see if they are nearing their subscription limits. The usage is tracked on a weekly basis, so updates to these charts happen about four times a month. This UI can only be viewed by Retail Home administrator users. Refer to the Retail Home product documentation for more information.

Figure 22-1 Retail Home Application Dashboard



23

Version Updates

Software updates are critical to keeping an environment secure and functioning well. Critical patch updates are installed on a quarterly basis, for example to the database, APEX/ORDS, and other tools being used in RDS. These updates may require downtime. If this is the case, the planned downtime is communicated in advance according to Oracle Retail standards.

Please note that for APEX patching, the only notifications for when this will occur are shown on the APEX workspace login page and the APEX UI home page; no email notification is sent. Log in at regular intervals to ensure you know when patching will occur, because the ADW instance will be unreachable during patching and all customer integration using RDS (ORDS) will fail. Likewise, APEX interaction should be suspended during the patch period.

You can defer major upgrades to APEX (for example, 22.1 to 23.1) for up to 90 days. See the following information on that process: <https://docs.oracle.com/en/cloud/paas/autonomous-database/serverless/adbsb/apex-apply-defer-updates.html>

For ADW patching, you can find the status of the current patch level, as well as the scheduled date for any future patching by running this query:

```
SELECT * FROM DB_NOTIFICATIONS WHERE TYPE = 'MAINTENANCE';
```

You can find more information on ADW patching and maintenance scheduling here: <https://docs.oracle.com/en/cloud/paas/autonomous-database/serverless/adbsb/maintenance-windows-patching.html#GUID-C4F488BA-C2ED-4890-A411-9F99C69CD8DF>

Additional Notes

This chapter groups implementation guidance under the following categories:

- **Not Supported in RDS:** Capabilities that are not available in the RDS environment.
- **Limited in RDS Environment:** Capabilities that exist but have important constraints.
- **Workarounds:** Features that do not work as expected but can be supported with additional effort.
- **Internal Use Only:** Implementation details visible to developers but not intended for use.

Not Supported/Available in RDS

APEX Roles and Privileges Not Supported

APEX roles and privileges are not available for RDS services. Any attempt to attach privileges to a service will result in that service becoming inaccessible. When invoking the service, the ORDS services container will respond with a 401, authorization required.

APEX Workspace Administration Disabled

APEX Administration services are not available. For example, you will not be able to create or remove workspaces. You will, however, be able to manage users as described in [APEX User Management](#).

GoldenGate Source/Target Use Not Supported

Use of RDS as a Golden Gate source is not supported. The use of RDS as a Golden Gate target beyond pre-configured retail cloud services is not supported.

PWA-Enabled APEX Apps Not Supported

PWA-enabled APEX applications are not supported in RDS due to a known APEX bug that results in a 401 Authorization error. Disable the PWA setting in your application as a workaround.

No Support for SFTP

There is no support for using SFTP. The workaround is to use Object Store.

Preconfigured Schema and Workspace Model

There is no support for the creation of new workspaces, users, or schema. The workspaces and schema are pre-configured and immutable.

Immutable Privilege Model for Replicated Objects

The baseline privilege model for replicated database objects cannot be changed. Select access to replicated data is granted automatically within **primary custom schemas**.

Individuals with access to a **primary custom schema** through the APEX UI are automatically granted select access to all replicated data in RDS. It is not possible to further restrict view- or row-level access within a primary custom schema.

Individuals with access only to an **auxiliary custom schema** (RDS_CUSTOM_1/2/3) must be explicitly granted select on the desired replicated views. Without such grants, they will not have access to any replicated data.

OAuth Scope Enforcement Not Supported

RDS does not support OAuth **scope-based access control** for IDCS client apps. Scopes in the token are not enforced.

Limited in RDS Environment

Service-Initiated Queries Limited to 300 Seconds

Service-initiated queries and PL/SQL blocks (that is, the service source) must complete in less than 300 seconds. This limitation is a standard timeout and not configurable. Queries and blocks of longer durations must be run asynchronously and report results using other approaches (that is, an output table populated by one service and queried by another, output to object storage, and so on).

Service Source Length Limited to 4000 Characters

Service source is limited to 4000 characters. Character count for GET oriented queries can be reduced using views without sacrificing the automatic to-JSON translation.

Only OCI SMTP Server Supported

To send email from your RDS ADW instance using the APEX_MAIL package, you must provision an OCI SMTP server. Only Oracle's OCI SMTP server is supported for sending email from RDS ADW.

APEX Limitations in Autonomous Database

Because RDS is built using Oracle Autonomous Data Warehouse (ADW), there are limitations with functionality provided by Oracle Application Express. These limitations are documented in [Oracle APEX Limitations on Autonomous Database](#).

Workarounds

Explain Plan via APEX UI Requires Workaround

Explain Plan in the APEX UI does not work as expected. Explain plan requires select access on replicated TABLES. The APEX user, however, only has access to replicated views by design. See the Explain Plan prerequisites in [SQL Language Reference: Explain Plan](#). Nevertheless, you can still access the explain plan functionality using a workaround.

Steps for this workaround:

1. Perform your query against the database - it may be helpful if you include a hint so that you can later find the SQL_ID for the query when performing step 2. For example we've added a hint with DEMO_WORKAROUND so that it's easy to find our query:

```
select /* +DEMO_WORKAROUND */ item, item_desc from rds_wv_item_master
where status = 'A';
```

2. Next, find the SQL_ID for the query you ran in step 1. In our example, this returned two rows that had the actual query from step 1 and the other the one we executed against v\$sqlarea. We want the non-v\$sqlarea one of those listed:

```
select sql_id, sql_text from v$sqlarea where upper(sql_text) like
'%DEMO_WORKAROUND%';
```

3. Next issue the following command to get the execution plan:

```
select * from table(dbms_xplan.display_cursor(<your SQL_ID>));
```

Exporting Query Results in APEX

One can export query results using **APEX > SQL Workshop > SQL Commands**. There is, however, a known issue where the resulting export is a zero-length file. This problem can occur when there are duplicate column names in the output file. For example, downloading the results of this query, which has two columns named DEPTNO, gives a zero-length file:

```
select e.deptno, d.deptno from emp e, dept d where d.deptno = e.deptno
```

The export thinks DEPTNO is duplicated in the output and fails. Downloading a variant of the query that has all unambiguous column names works as expected:

```
select e.deptno as emp_deptno, d.deptno from emp e,dept d where d.deptno =
e.deptno
```

Note that the following query is also problematic and will fail:

```
select * from emp e, dept d where d.deptno = e.deptno
```

More generally, when joining results, you will need to use AS <your-output-column-name> to manually disambiguate output column names.

Enhancing OAuth Security with ORDS Prehook

ORDS services are authenticated through OAUTH, but if your deployment has stage and production environments that are both protected with the same identity domain, then OAUTH tokens generated for stage also work for production. If you determine that this degree of accessibility poses a security concern, you may mitigate this risk by implementing a custom ORDS_PREHOOK function to perform additional authentication. See [ORDS PRE-HOOK](#) for additional details. Host names for production and non-production environments have distinct signatures.

Retrieving RDS Outbound IP Address

If the RDS outbound IP address is required for ingress allowlisting, it can be found with the following query:

```
select json_value(cloud_identity, '$."OUTBOUND_IP_ADDRESS"[\*]') FROM v$pdb;
```

Internal Use Only

Reserved Application ID Range

Oracle has reserved the APEX application ID range from 30001-39999 for future enhancements. Use of APEX application IDs within this range in custom code may result in undefined behavior.

GoldenGate Metadata Fields Are Not CDC

You might notice TARGET_COMMIT_DATETIME, LAST_DM, and CSN_NBR columns on replicated views. These columns are hidden on the source but visible on the target. These columns are for internal use only by GoldenGate. They are not to be confused with change data capture support.

A

Query Tuning

Introduction

When a query slows down, you're not powerless—and you're not the first person to encounter a slow query. Most SQL performance problems are fixable, and this guide gives you a way to start. This isn't about blame. It's about control. If you start with the premise that it is something you did that slowed down the query, then you follow up with the premise there is something you can do to speed it up. And by doing so, you'll gain something even more valuable: the ability to diagnose and avoid problems earlier next time. Even if this is your first slow query, it won't be your last. The process you follow here—understanding how SQL executes, capturing a plan, identifying where things go off track—will make you faster and better at writing efficient queries in the future. You'll stop performance issues before they start. Taking ownership doesn't mean going it alone—it means being in control. It means asking better questions, testing smarter ideas, and sometimes solving the issue before it becomes a roadblock.

This guide helps you do that. It gives you the tools, mindset, and examples you need to take control of performance and build that skill for good.

How SQL Executes - A Conceptual Overview

Most performance problems begin with a SQL statement. Understanding *how SQL runs* in a relational database gives you the foundation to troubleshoot when things go wrong.

This overview helps you:

- Grasp the core moving parts behind SQL performance.
- Understand where problems originate.
- Decide what to check first.

Relational Databases: What You Need to Know

Relational databases like Oracle are *declarative, set-based engines*. You tell them *what* data you want (using SQL), and the database figures out *how* to retrieve it. The execution is driven by a component called the **optimizer**, which chooses the most efficient plan it can—based on statistics, indexes, and data volume.

Every SQL statement is turned into a **plan** composed of a small number of operations:

- Table access (full scan or index)
- Join (nested loop, hash, merge)
- Filter
- Aggregate
- Sort

The plan is what determines performance.

A Mental Model You Can Use

Here's a simplified way to think about SQL execution, especially if you come from an algorithm or systems background:

- **SQL is a request.**
- **The optimizer is a planner.**
- **The execution plan is a strategy.**
- **Execution is a parallel, resource-aware run of that strategy.**

When performance suffers, one or more of these steps failed to line up with reality.

Where Problems Show Up

Problem Area	What's Happening Under the Hood
Query runs forever	Bad access path (for example: full table scan)
Query never returns	Execution stuck on lock, contention, or temp space
Query used to be fast	Execution plan changed (due to stats, bind peeking)
Query runs fast once	Plan got cached or parallelism kicked in by chance
System is overloaded	Too many concurrent queries, bad query patterns

Start Here: Performance Triage

Use this decision tree to get oriented quickly:

Is the problem with one query or the whole system?

- **One query is slow**
 - Capture the **SQL_ID**
 - View the **execution plan**
 - Look for bad access paths, joins, temp usage
 - See: [Diagnosing a Slow Query](#)
- **Multiple queries are slow**
 - Check **system resource usage** (CPU, I/O, concurrency)
 - Identify heavy queries in **AWR or SQL Monitor**
 - See: [Diagnosing System-Wide Performance Issues](#)
- **Query performance regressed recently**
 - Check **plan history** (DBA_HIST_SQL_PLAN)
 - See if **optimizer stats changed** or **binds differ**
 - See: [Understanding Plan Instability](#)

Next Step

Pick the path that matches your situation and go straight there. You don't have to understand everything about Oracle SQL internals—but the more you understand, the better you'll be able to reason about performance.

To start that process, you need a baseline.

Baselining Your Query

Before you ask “*Is my query slow?*” you need to ask a different question:

How does this query normally behave?

A **baseline** gives you a point of comparison. It's the only way to tell whether performance is degrading, improving, or holding steady. Without one, everything is just a hunch.

Why Baselining Matters

Most performance issues don't come out of nowhere. They evolve—slowly, then suddenly. Baselining helps you:

- Detect regressions early
- Justify changes or rollback decisions
- Compare across environments (for example, dev vs. prod)
- Communicate clearly with support

Baseline Tuning

A functional query is not necessarily a fast query. You run your first baseline once you have formulated a functional query: a query that returns the expected result. You run your baseline against representative data volumes on a mostly quiet system. Your baseline is the fastest you can ever expect your query to run. Higher system loading will only slow your query down. In any case, your first baseline may be your first indication that you have a problem. Specifically, your baseline fails to meet performance expectations. So your first baseline collection may evolve into a tuning effort.

What to Capture

For any business-critical or frequently executed query, record the following:

Metric	How to Get It (GV\$SQL)
SQL_ID	SELECT sql_id FROM gv\$sql WHERE sql_text LIKE '%...%'
Elapsed time (avg)	elapsed_time / executions
Rows processed	rows_processed / executions
Buffer gets	Logical I/O (work done)
Disk reads	Physical I/O (slower)
Plan hash value	Unique ID for execution plan

If the query is already running, use `DBMS_SQL_MONITOR` or `DBMS_XPLAN.DISPLAY_CURSOR` to capture **real-time** stats.

A SQL ID is a unique identifier for a SQL statement in Oracle. It is constructed by hashing the text of the SQL statement.

The SQL ID is generated using a hash function that takes the SQL statement text as input. The resulting hash value is then converted to a 13-character string, which is the SQL ID.

The hashing algorithm used to generate the SQL ID is not publicly documented, but it is designed to produce a unique identifier for each distinct SQL statement.

Here are some key points to note about SQL IDs:

1. **Case sensitivity:** SQL IDs are case-sensitive, so the same SQL statement with different casing will produce different SQL IDs.
2. **Whitespace and formatting:** SQL IDs are sensitive to whitespace and formatting, so the same SQL statement with different formatting will produce different SQL IDs.
3. **Literal values:** SQL IDs are sensitive to literal values, so the same SQL statement with different literal values will produce different SQL IDs.
4. **Bind variables:** If a SQL statement uses bind variables, the SQL ID will be the same regardless of the values bound to the variables.

When to Baseline

- Before go-live
- After a major schema or stats change
- After tuning
- Periodically for high-impact queries

Storing baselines in a spreadsheet or performance table gives you history that support or DevOps can reference later.

Baseline Template (Example)

SQL_ID	Date	Avg Elapsed (s)	Avg Rows	Avg Buff Gets	Avg Disk Reads	Plan Hash
abcd1234	2025-05-02	2.8	14,200	28,000	64	872349876



Tip

Automate this for key queries with a scheduled job.

If you care about a query, baseline it. You'll thank yourself later when something changes—and you need to prove it.

Baseline SQL Script (for Recent Performance Snapshot)

Here's a sample SQL script to baseline a query by SQL_ID, returning key performance metrics that you can copy into Excel, store in a logging table, or track over time:

```
SELECT
    sql_id,
    plan_hash_value,
    TO_CHAR(SYSDATE, 'YYYY-MM-DD') AS baseline_date,
    ROUND(elapsed_time/1000000, 2) AS total_elapsed_sec,
    executions,
    ROUND((elapsed_time/1000000) / NULLIF(executions, 0), 2) AS
avg_elapsed_sec,
    rows_processed,
    ROUND(rows_processed / NULLIF(executions, 0)) AS avg_rows,
    buffer_gets,
    ROUND(buffer_gets / NULLIF(executions, 0)) AS avg_buffer_gets,
    disk_reads,
    ROUND(disk_reads / NULLIF(executions, 0)) AS avg_disk_reads
FROM
    gv$sql
WHERE
    sql_id = '<your SQL_ID>'
ORDER BY
    last_active_time DESC
FETCH FIRST 1 ROWS ONLY;
```

Output Columns

Column	Description
sql_id	Identifier of the query
plan_hash_value	Unique ID for the current execution plan
baseline_date	Date when the baseline was captured
total_elapsed_sec	Total time across all executions
executions	Number of times the query ran
avg_elapsed_sec	Average execution time per run
avg_rows	Average rows returned
avg_buffer_gets	Logical I/O (how much work Oracle did)
avg_disk_reads	Physical I/O (slowest part of query access)

Baselining Procedure

Here's how to automate nightly baseline logging for important SQL statements by maintaining a list of SQL_IDs and looping through them in a job.

Step 1: Create a Tracking Table

```
CREATE TABLE sql_baseline_log (
    log_id          NUMBER GENERATED ALWAYS AS IDENTITY,
    log_date        DATE DEFAULT SYSDATE,
```

```

        sql_id          VARCHAR2(13),
        plan_hash_value NUMBER,
        executions      NUMBER,
        total_elapsed_s NUMBER(10,2),
        avg_elapsed_s   NUMBER(10,2),
        rows_processed  NUMBER,
        avg_rows        NUMBER,
        buffer_gets     NUMBER,
        avg_buffer_gets NUMBER,
        disk_reads      NUMBER,
        avg_disk_reads  NUMBER,
        PRIMARY KEY (log_id)
    );

```

Step 2: Create the Baselining Procedure

```

CREATE OR REPLACE PROCEDURE log_sql_baseline(p_sql_id IN VARCHAR2) AS
BEGIN
    INSERT INTO sql_baseline_log (
        sql_id, plan_hash_value, executions,
        total_elapsed_s, avg_elapsed_s,
        rows_processed, avg_rows,
        buffer_gets, avg_buffer_gets,
        disk_reads, avg_disk_reads
    )
    SELECT
        sql_id,
        plan_hash_value,
        executions,
        ROUND(elapsed_time / 1e6, 2),
        ROUND(elapsed_time / 1e6 / NULLIF(executions, 0), 2),
        rows_processed,
        ROUND(rows_processed / NULLIF(executions, 0)),
        buffer_gets,
        ROUND(buffer_gets / NULLIF(executions, 0)),
        disk_reads,
        ROUND(disk_reads / NULLIF(executions, 0))
    FROM
        gv$sql
    WHERE
        sql_id = p_sql_id
    ORDER BY
        last_active_time DESC
    FETCH FIRST 1 ROWS ONLY;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Baseline logging failed: ' || SQLERRM);
END;

```

Step 3: Example Call

```
BEGIN
    log_sql_baseline('abcdef1234567'); -- Replace with your SQL_ID
END;
```

Optional: Automate It Nightly for Key Queries

You can maintain a list of important SQL_IDs in a table and loop through them in a scheduled job. Here's how to automate nightly baseline logging for important SQL statements by maintaining a list of SQL_IDs and looping through them in a job.

Step 1: Create a Tracking Table

```
CREATE TABLE tracked_sql_ids (
    sql_id          VARCHAR2(13) PRIMARY KEY,
    description     VARCHAR2(200)
);
```

Insert the SQL_IDs of the queries you care about:

```
INSERT INTO tracked_sql_ids (sql_id, description)
VALUES ('abcdef1234567', 'Critical dashboard query');

COMMIT;
```

Step 2: Create a Batch Baselining Procedure

```
CREATE OR REPLACE PROCEDURE baseline_all_tracked_sql IS
BEGIN
    FOR r IN (SELECT sql_id FROM tracked_sql_ids) LOOP
        BEGIN
            log_sql_baseline(r.sql_id);
        EXCEPTION
            WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('Error logging ' || r.sql_id || ': ' ||
SQLERRM);
        END;
    END LOOP;
END;
```

This wraps your log_sql_baseline procedure and runs it for each tracked SQL_ID.

Step 3: Optional: Schedule the Job (DBMS_SCHEDULER)

```
BEGIN
    DBMS_SCHEDULER.create_job (
        job_name          => 'baseline_tracked_sql',
        job_type           => 'PLSQL_BLOCK',
```

```
job_action      => 'BEGIN baseline_all_tracked_sql; END;',
start_date      => SYSTIMESTAMP,
repeat_interval => 'FREQ=DAILY;BYHOUR=2', -- runs at 2 AM
enabled         => TRUE
);
END;
```

You now have:

- A log of query performance over time.
- A foundation for spotting regressions.
- A process that runs nightly with minimal overhead.

Is My Query Slow?

Before jumping into diagnosis, it's worth asking: **is the query actually slow—or just slower than expected?** The goal of this section is to help you assess whether a query meets your requirements or needs tuning.

Start with your Requirements

Not all queries need to be fast. Some run in the background or during batch windows. Others support REST services, dashboards, or interactive user flows—and those **do** have time limits. If you're running in a **REST service** (including Oracle REST Data Services, APEX APIs, or external integrations), Oracle enforces a **300-second (5 minute) timeout**. That includes SQL or PL/SQL blocks.

Rule of thumb:

If your query supports a REST service or user interaction, it must complete in under 300 seconds. Otherwise, it will fail.

How to Think About "Fast Enough"

Ask yourself:

- **Is this query blocking a user?**
 - Yes - Aim for sub-second to low-second response times.
- **Is this query supporting a dashboard or interactive page?**
 - Yes - Sub-5 seconds preferred; sub-30 seconds may be acceptable with feedback (for example, spinner).
- **Is it a background report or batch process?**
 - Yes - Performance is relative. It may be fine as-is unless it's consuming too many resources.

Measuring Execution Time

Even if the user reports that “it’s slow,” confirm it:

```
SELECT elapsed_time/1000000 AS seconds, executions
FROM gv$sql
WHERE sql_id = '<your-sql-id>';
```

This shows average time per execution. One run taking 90 seconds may not be a problem—unless it’s being called hundreds of times per hour or runs in a timeout-sensitive context.

Don't Tune Without a Target

Tuning without knowing your goal leads to wasted time. Instead, start with:

- **Functional context:** Who/what depends on this query?
- **Frequency:** How often does it run?
- **Failure mode:** Has it ever hit a timeout or caused a system issue?

If the answer to any of those is yes—or if the user experience is impacted—then move on to [Diagnosing a Slow Query](#).

Diagnosing a Slow Query

When a query is slow, the key is to work from facts—not guesses. You don’t need to be an expert in Oracle internals, but you do need to capture what the database is doing and compare that to what you expected. This section walks you through a simple and repeatable process.

Step 1: Get the SQL_ID

Before you do anything else, identify the query using its SQL_ID. You can get it:

- From the application or APEX session logs
- Using a query on recent activity:

```
SELECT sql_id, sql_text
FROM gv$sql
WHERE sql_text LIKE '%target_table%'
AND last_active_time > SYSDATE - 1/24
ORDER BY last_active_time DESC;
```

Step 2: Get the Execution Plan

Once you have the **SQL_ID**, pull the most recent execution plan. This tells you *how* Oracle is trying to run the query.

```
SELECT *
FROM table(DBMS_XPLAN.DISPLAY_CURSOR('<your-sql-id>', NULL, 'ALLSTATS LAST'));
```

This gives you the final plan from the most recent run, including row estimates vs. actual rows, I/O, and temp usage.

Step 3: Ask These Questions

Review the execution plan and look for red flags:

Question	What to Look For
Are there full table scans?	Look for TABLE ACCESS FULL on large tables
Are joins using hash or nested loops?	Hash joins are usually good for large inputs. Nested loops on big tables may be slow.
Are estimated rows way off from actual rows?	That means bad stats or data skew.
Is there temp usage?	Indicates a sort, hash join, or aggregation that didn't fit in memory.
Is the plan parallel?	Look for PX steps. If it's not using parallel and it should, the query may be underutilizing the ADW architecture.

Step 4: Check for Waits

If the plan looks okay, check what the query is **waiting on**:

Current Waits

```
SELECT event, wait_class, time_waited, sid
FROM gv$session
WHERE sql_id = '<your-sql-id>';
```

Recent Waits

```
SELECT event, wait_class, time_waited, session_id
FROM gv$active_session_history
WHERE sql_id = '<your-sql-id>';
```

Look for:

- db file sequential read: Index access (can be okay)
- db file scattered read: Full table scan
- direct path read temp: Temp usage, probably from a spill
- enq: TX: Locking issues

What's a Spill

A **spill** occurs when operations like sorts, joins, or aggregations require more memory than is available, causing intermediate data to be written to temporary disk storage instead. This typically happens in the temporary tablespace and can significantly slow down performance due to increased disk I/O. Spills indicate that a query exceeded its in-memory processing capacity and may benefit from SQL tuning.

Step 5: Cross Check Stats

Bad estimates can lead to bad plans. Check the table and column stats:

```
SELECT table_name, last_analyzed, num_rows, owner
FROM dba_tables
WHERE table_name = '<target-table>;
```

Also check for histograms:

```
SELECT column_name, histogram, num_distinct, owner
FROM dba_tab_col_statistics
WHERE table_name = '<target-table>;
```

If LAST_ANALYZED is old or there are no histograms on skewed columns, you may want to gather fresh stats.

What's a Skewed Column

A **skewed column** is a column where some values appear **much more frequently** than others—creating an **uneven distribution** of data. In a well-distributed column, values are fairly balanced. In a skewed column, a few values dominate the data.

Why Skew Matters

The **Oracle optimizer relies on statistics** to estimate how many rows a query will return. If it thinks all values are equally likely (uniform distribution), it may pick a bad plan:

- Use a full table scan instead of an index.
- Choose the wrong join method.
- Under- or over-allocate memory.

How Oracle Handles Skew

When Oracle detects skew, it can collect a **histogram** to model the actual frequency of column values. This helps the optimizer make better decisions, especially in **bind-sensitive** or **filter-heavy** queries.

You can check for skewed columns using:

```
SELECT column_name, histogram, num_distinct
FROM dba_tab_col_statistics
WHERE table_name = '<target-table>' AND owner = '<my-owner>;
```

Look for FREQUENCY or TOP-FREQUENCY histograms—that means Oracle found skew and is tracking it.

Step 6: Consider Plan History

Plans can change. To see what changed and when:

```
SELECT *  
FROM dba_hist_sql_plan  
WHERE sql_id = '<your-sql-id>'  
ORDER BY plan_hash_value, timestamp;
```

Look for:

- Plan changes over time
- Elapsed time or row differences across plans

Summary: What to do Next

If you found...

- **Bad access paths:** Add indexes or rewrite joins.
 - You won't be able to add indexes to replicated objects
- **Mismatched estimates:** Check stats and consider histograms.
- **Temp spills:** Filter earlier.
- **Wrong join type:** Add hints or use a different query shape.
- **Stale plans:** Consider SQL plan baselines or profiles.

Diagnosing System Wide Performance Issues

When your entire application or environment feels sluggish—slower pages, delayed jobs, or timeouts—you need to look beyond individual queries and assess **system-wide health**. In the Oracle Retail Data Store (RDS) environment, you have limited infrastructure visibility, but the **DB Ops Console** and **AWR reports** give you powerful tools to diagnose problems effectively.

This section explains where to look and what to interpret using the capabilities provided in the *RDS DB Ops Console*.

Step 1: Use AWR Reports to Understand System Load

The **Automated Workload Repository (AWR)** report is your most powerful tool for understanding what the system was doing during a specific period. In the RDS DB Ops Console, you can:

- View automatically generated AWR reports (hourly snapshots)
- Filter and search for reports by date/time
- View full reports directly in the UI or download them for further analysis

To access:

Retail Home -> Application Navigator -> DB Ops Console -> AWR Reports

What AWR Tells You

An AWR report answers the following key questions:

- What was the **database doing most** during the time window?
- What were the **top resource consumers**?
- Was the database **waiting on something**?
- Did any **SQL dominate** CPU, I/O, or elapsed time?

Focus Areas in the AWR Report

Top 10 Foreground Events by Total Wait Time

- Start here. It tells you what the database spent time *waiting on*.
- Events to watch:
 - db file sequential read or scattered read: I/O-bound
 - direct path read/write temp: spills to disk due to memory limits
 - enq: TX - row lock contention: blocking transactions
 - log file sync: commit contention
 - resmgr:cpu quantum: Resource Manager throttling (ADW)

Look at both **total time** and **waits per second** to distinguish chronic from bursty issues.

Load Profile

- Gives a high-level view of system intensity.
- Important metrics:
 - **DB Time(s)** – Total time spent on queries and waits
 - **DB CPU(s)** – Portion of DB time spent on CPU
 - **Executes (SQL)** – Query activity volume
 - **Redo size (bytes)** – Change rate and workload size

If **DB Time is much higher than CPU**, most time was spent *waiting*, not computing.

SQL Statistics

Sections include:

- **Elapsed Time** – Longest running queries
- **CPU Time** – CPU-bound queries
- **Buffer Gets** – Logical I/O volume (data movement)
- **Executions** – Most frequently run statements

Identify:

- One-off queries that were expensive
- Repeated queries that may be chatty or inefficient
- SQL with high cost but low rows returned: signs of inefficiency

Buffer Cache Hit Ratio

Hit ratio should be high. Low means excessive disk I/O

Deriving Buffer Cache Hit Ratio from AWR

The **buffer cache hit ratio** is not listed directly in the AWR report, but you can compute it using values from the **Instance Activity Stats** section:

$$\text{Buffer Cache Hit Ratio} = 1 - (\text{physical reads} / (\text{db block gets} + \text{consistent gets}))$$

Step-by-Step

1. Open your AWR report.
2. Find the **“Instance Activity Stats”** section.
3. Locate the following three metrics:
 - physical reads
 - db block gets
 - consistent gets
4. Plug the values into the formula above.
5. Multiply by 100 for a percentage.

A high ratio may look healthy but can mask inefficient access patterns. Use this metric as a supporting clue, not a primary diagnostic.

Parse to Execute Ratio

Deriving the Parse to Execution Ratio from AWR

- **Parse to Execute Ratio** – High = too many hard parses (bad reuse)
- **Redo per Txn** – Spikes can indicate heavy DML or poor batching

$$\text{Parse to Execute Ratio} = \text{parse count} / \text{execute count}$$

Step-by-Step

1. Open your AWR report.
2. Find the **“Instance Activity Stats”** section.
3. Locate the following two metrics:
 - parse count (total)
 - execute count
1. Plug the values into the formula above.
2. Multiply by 100 for a percentage.

Understanding Temporary Table Space Usage

If direct path read temp or direct path write temp are among the top wait events, it may indicate a problem. Check the total wait time for these events. If the wait time for these events is a significant portion of the total wait time, then **sorts, hash joins, or group by** operations may be spilling to disk.

You will find path read temp and direct path write temp in the Foreground Wait Events table. One or both may be missing from the table. If they are missing, temporary table space usage is unlikely to be a problem.

Blocking and Concurrency

When investigating blocking and concurrency, you are looking for wait events. If the locks and latches described below are among the top wait events, it may indicate a problem. Check the total wait time for these events. If the wait time for these events is a significant portion of the total wait time, you will need to investigate further.

Explicit Locks

Explicit locks are created with `LOCK TABLE` or `SELECT ... FOR UPDATE` statements. If a session is blocked because of a `LOCK TABLE`, you will see it associated with a `enq: TM - contention` wait event. If a session is blocked because of a `SELECT ... FOR UPDATE` statement, you will see it associated with a `enq: TX - row lock contention` wait event. You can find these statements in SQL text of running sessions.

Implicit Locks

`INSERT`, `UPDATE`, or `DELETE` implicitly acquire locks as well. You will see a blocked session associated with a `enq: TX - row lock contention` wait event, if it is blocked by an `INSERT`, `UPDATE`, or `DELETE` operation.

Latch Locks

Latch locks are lightweight synchronization mechanisms used by Oracle to protect shared data structures in memory. Here are some common latch locks and their causes:

1. **cache buffers chains latch:** This latch protects the buffer cache chains, which are used to manage the buffer cache. Contention on this latch can occur due to:
 - High buffer cache activity (for example, frequent reads and writes).
 - Poor buffer cache sizing or configuration.
 - High concurrency (for example, many sessions accessing the same data).
2. **cache buffers lru chain latch:** This latch protects the LRU (Least Recently Used) chain, which is used to manage the buffer cache. Contention on this latch can occur due to:
 - High buffer cache activity (for example, frequent reads and writes).
 - Poor buffer cache sizing or configuration.
3. **redo allocation latch:** This latch is used to allocate space in the redo log buffer. Contention on this latch can occur due to:
 - High redo log activity (for example, frequent commits or high DML activity).
 - Poor redo log buffer sizing or configuration.
4. **redo copy latch:** This latch is used to copy redo log data from the log buffer to the redo log files. Contention on this latch can occur due to:
 - High redo log activity (for example, frequent commits or high DML activity).
 - Poor redo log buffer sizing or configuration.
5. **library cache latch:** This latch protects the library cache, which is used to store parsed SQL statements and other metadata. Contention on this latch can occur due to:
 - High SQL parsing activity (for example, frequent execution of dynamic SQL).
 - Poor cursor management (for example, not using bind variables).
6. **shared pool latch:** This latch protects the shared pool, which is used to store shared SQL and PL/SQL objects. Contention on this latch can occur due to:
 - High shared pool activity (for example, frequent parsing or loading of SQL and PL/SQL objects).

- Poor shared pool sizing or configuration.

Common Causes of Latch Contention

1. **High concurrency:** Many sessions accessing the same data or resources simultaneously.
2. **Inefficient SQL:** SQL statements that cause high parsing or execution overhead.
3. **Lack of indexing or poor indexing:** Insufficient or poorly designed indexing can lead to high buffer cache activity and latch contention.
4. Mismatch between index design and query goals.

Replicate Objects

Only replication will create locks on replicated objects. Read operations (for example, SELECT statements) on replicate objects will generally not be blocked by locks. In other words, locking on replicated objects should not be an issue for replicate objects. You are also unable to create indexes for replicated objects. As a result, you may find yourself in a position where you lack the indexing to support your query. You can list indexes for views using the DBA_INDEXES and DBA_IND_COLUMN tables if you are uncertain about which columns are indexed in each view.

What to Compare

If this is not your first AWR review, compare against:

- A previous report from the **same time of day** (for example, yesterday 2–3 PM)
- A known **healthy period** (for example, baseline from last successful run)

Look for:

- New top SQLs
- Increased wait time or database time
- Different execution patterns (for example, hash joins vs. nested loops)

Report Retention Notes

- AWR reports in the RDS DB Ops Console are **retained for 30 days**.
- Use the “**Refresh Report Logs**” button to get the latest list.
- Reports can be **downloaded** for local storage or comparison.
- Custom reports (if you have RDS_MANAGEMENT_OWNER role) can be generated for any snapshot range. A snapshot range longer than several hours is not recommended.

When to Escalate or Log a Baseline

If the AWR shows a shift in workload or a clear new bottleneck, **capture and store**:

- The report ID
- Top SQLs (SQL_ID and plan hash)
- Wait events
- Load profile

This gives you a complete snapshot to compare after changes or to include when engaging support.

AWR Review Checklist

Snapshot Range Reviewed (Begin Snapshot ID - End Snapshot ID):**Time Window (Start Time – End Time):** _____**1. SYSTEM LOAD**

- **DB Time per Second:**
 - Value: _____ — Is it high for this system?
- **CPU Usage per Second:**
 - Value: _____ — Is DB Time >> CPU time? Suggests waits.
- **Executions per Second:**
 - Value: _____ — Is there a spike or dip?

2. TOP WAIT EVENTS

Wait Event	Time (s)	Notes [e.g., temp spill, I/O, locking]
1.		
2.		
3.		

3. TOP SQL STATEMENTS

SQL_ID	Metric	Value	Comment (slow/overused/etc.)
	Elapsed Time		
	Buffer Gets		
	Executions		

4. TEMPORARY SPACE USAGE

- **Temp Waits (for example, direct path read/write temp):**
 - Observed? Yes / No
 - Notes: _____
- **Operations likely causing spills (sorts, joins, and so on):**

5. INSTANCE EFFICIENCY RATIOS

- **Buffer Cache Hit Ratio:** _____ %
- **Parse to Execute Ratio:** _____ (High = too many hard parses?)

6. NOTABLE CHANGES COMPARED TO BASELINE

Area	What Changed	Details
Workload	Increased / Decreased	
Waits	New / Increased	
Top SQL	Changed / Same	

7. INITIAL CONCLUSION / NEXT STEP

(for example, Investigate temp usage, tune top SQL, compare plans, escalate to support)

Step 2: Use Database Metrics (DB Ops Console)

Navigate to:

Retail Home -> Application Navigator -> DB Ops Console -> Database Metrics

Table A-1 Key Metrics to Watch

Metric	What It Indicates
CPU Utilization	High = heavy load; may align with performance drop
Storage Utilization	If near capacity, could be causing issues
Session Count	Spike in active sessions = concurrency issue
Execute Count	High = heavy workload; baseline expected volume
Running Statements	Indicates how many statements are using resources
Queued Statements	Indicates backlog due to contention
APEX Load Time	Perceived slowness for end users
APEX Page Events	Intensity of user interaction with APEX apps

Look at trends using 1-minute or 5-minute intervals. Spikes or sustained elevation may confirm a user-reported slowdown.

Step 3: Triage with "Top SQL"

Navigate to:

DB Ops Console -> Top SQL

- Identify SQLs that are frequently active or waiting.
- Filter by **session state**: ON CPU vs. WAITING.
- Click into a SQL ID to view execution details and full text.

This is your quickest path to **confirm which queries are stressing the system now**, even outside the AWR snapshot window.

Check DBMS Jobs (If Relevant)

Navigate to:

DB Ops Console -> DBMS Jobs

- View failed or long-running jobs.
- Check job history for trends or specific failures during slow periods.

Step 5: Session Management (Admin Role)

If you have admin rights, you can also:

- View active sessions (filter by username, SQL ID, and so on).
- Identify stuck or long-running sessions.
- Use the provided interface or API to terminate safe-to-kill sessions (based on naming patterns like `_RDS_CUSTOM`).

Table A-2 Interpretation Tips

Observation	Likely Cause / Next Step
High wait time on temp I/O	Queries spilling to disk → check joins/sorts
One query dominates Top SQL	Regressed plan or unindexed access → tune it
Lots of queued statements	Resource bottleneck (e.g., CPU) → throttle, tune
Sharp spike in APEX load time	Backend slowness or DB resource saturation
Session count far above normal	Application surge, leaking sessions, or blocking

Takeaway

Even without OS access or OEM, you can do meaningful diagnosis using:

- **AWR reports** for time-window analysis
- **Database metrics** for real-time and trending load
- **Top SQL and jobs** to find active bottlenecks

You don't need to solve everything at once. Start by identifying **what changed, what's dominant, or what's waiting**, and then follow the evidence.

Understanding Plan Instability

You tune a query, it runs well, and then—suddenly—it doesn't. This is often due to **plan instability**, where Oracle's optimizer picks a different execution plan for the same SQL text. Plan changes are normal. But when the new plan performs worse, it becomes a **problem**.

You should not encounter a plan instability problem. The Oracle Autonomous Database should keep plan instability from becoming a problem. The steps below will help you make a case and submit a ticket if you think you have an instability problem.

What is Plan Instability?

Oracle's optimizer generates a plan each time a query is parsed. If the inputs to the optimizer change—even slightly—it might produce a different plan. Most of the time, that's fine. But sometimes:

- The new plan is much slower.
- The old plan would still be better.
- You didn't change anything, but the plan changed anyway.

Why Plans Change

Cause	Description
Stats changed	Table, index, or column stats were refreshed. Cardinality estimates changed.
Bind peeking	The first bind value caused a plan that's bad for later values.
No histograms	Optimizer assumes uniform data when actual data is skewed.
New index or object	The optimizer sees a new access path and tries it.
Adaptive optimization	Oracle switched plans mid-execution based on early row counts.
SQL text changed slightly	Even whitespace differences result in new SQL_IDs and plans.
Cursor aged out	The plan aged out of the shared pool and was regenerated.

How to Detect Plan Instability

Use AWR or SQL history views to spot plan changes:

```
SELECT sql_id, plan_hash_value, COUNT(*)
FROM dba_hist_sql_plan
WHERE sql_id = 'your_sql_id'
GROUP BY sql_id, plan_hash_value
ORDER BY COUNT(*) DESC;
```

- If there are **multiple plan_hash_values** for the same sql_id, you've had a plan change.
- Look at performance metrics for each plan: which one was faster? More consistent?

```
SELECT is_bind_sensitive, is_bind_aware, is_shareable
FROM v$sql
WHERE sql_id = 'your_sql_id';
```

If is_bind_sensitive = Y but is_bind_aware = N, you may be suffering from **bind peek instability**.

When Plan Changes are Bad

A plan change is only a problem when the **new plan is worse**. Symptoms include:

- Increased runtime
- Increased temp usage (due to spills)
- Higher CPU or I/O
- Lower row estimates than actuals
- Drastically different join orders

What To Do

If you believe you have encountered an unstable plan, submit your research in a ticket. Generally, you will not have access to the tools that will allow you to fix the problem yourself.

B

Using FTS: An Example

Overview

This chapter provides an example of an API that provides a set of functions and procedures to facilitate data export from a database query to object store using FTS. The API consists of four main components:

1. `get_idcs_token`: Obtains an IDCS token for authentication.
2. `generate_par`: Generates a Pre-Authenticated Request (PAR) for uploading data to an object store.
3. `put_object_in_store`: Uploads a BLOB to object store using a PAR.
4. `export_query_to_blob`: Exports the result of a database query to a BLOB.

Functions and Procedures

`get_idcs_token`

- **Purpose:** Obtain an IDCS token for authentication.
- **Parameters:**
 - `p_idcs_url`: IDCS URL.
 - `p_idcs_client_id`: IDCS client ID.
 - `p_idcs_client_secret`: IDCS client secret.
 - `p_scope_suffix`: IDCS scope suffix.
- **Return Value:** IDCS token (VARCHAR2).
- **Raises:** `INVALID_IDCS_CREDENTIALS` if the IDCS credentials are invalid.

`generate_par`

- **Purpose:** Generate a Pre-Authenticated Request (PAR) for uploading data to an object store.
- **Parameters:**
 - `p_base_url`: Base URL of object store.
 - `p_tenant`: Tenant ID.
 - `p_idcs_token`: IDCS token obtained using `get_idcs_token`.
 - `p_os_prefix`: Object store prefix.
 - `p_file_name`: File name to be uploaded.

- **Return Value:** PAR access URI (VARCHAR2).
- **Raises:** PAR_GENERATION_FAILED if PAR generation fails.

put_object_in_store

- **Purpose:** Upload a BLOB to object store using a PAR.
- **Parameters:**
 - p_access_uri: PAR access URI obtained using generate_par.
 - p_blob_data: BLOB data to be uploaded.
- **Raises:** UPLOAD_FAILED if the upload fails.

export_query_to_blob

- **Purpose:** Export the result of a database query to a BLOB.
- **Parameters:**
 - p_sql_query: SQL query to be executed.
- **Return Value:** BLOB containing the query results.
- **Raises:** QUERY_EXECUTION_FAILED if the query execution fails.

Example Usage

```
DECLARE
    l_idcs_token  VARCHAR2(4000);
    l_par_uri     VARCHAR2(32767);
    l_blob_data   BLOB;
BEGIN
    l_idcs_token := get_idcs_token('https://idcs-url.com', 'client-id',
    'client-secret', 'scope-suffix');
    l_par_uri := generate_par('https://object-store-url.com', 'tenant-id',
    l_idcs_token, 'os-prefix', 'file-name.csv');
    l_blob_data := export_query_to_blob('SELECT * FROM my_table');
    put_object_in_store(l_par_uri, l_blob_data);
END;
```

Error Handling

The API raises specific exceptions for each function/procedure, allowing the caller to handle errors accordingly. The exceptions are:

- INVALID_IDCS_CREDENTIALS
- PAR_GENERATION_FAILED
- UPLOAD_FAILED
- QUERY_EXECUTION_FAILED

These exceptions can be caught and handled using standard PL/SQL error handling mechanisms.

Securely Obtaining an IDCS Token

Overview

The `get_idcs_token_for_planning_app` function provides a secure way to obtain an IDCS token for the Planning App. This function is designed to be used by authorized users and applications, and it abstracts away the underlying complexity of obtaining an IDCS token.

Security Considerations

- The `get_idcs_token_for_planning_app` function is a privileged function that has access to sensitive information, such as the IDCS client ID and client secret. Therefore, execute permission on this function is restricted to authorized users and applications.
- The function does not expose the underlying IDCS client ID and client secret, ensuring that these sensitive values are not compromised.

Using the `get_idcs_token_for_planning_app` Function

To obtain an IDCS token securely using the `get_idcs_token_for_planning_app` function, follow these steps:

1. Ensure that you have the necessary execute permission on the `get_idcs_token_for_planning_app` function.
2. Call the `get_idcs_token_for_planning_app` function using a secure connection (for example, over a secure database connection).
3. The function will return a valid IDCS token that can be used for authentication and authorization purposes.

Example Usage

```
DECLARE
    l_idcs_token VARCHAR2(4000);
BEGIN
    l_idcs_token := get_idcs_token_for_planning_app();
    -- Use the obtained IDCS token for authentication and authorization
    purposes
END;
```

Using the `export_query_to_object_store` Procedure

The `export_query_to_object_store` procedure is a template that exports the result of a SQL query to object store. To use this procedure, you need to replace the hardcoded placeholders with your actual values.

Prerequisites

- The `get_idcs_token_for_planning_app`, `export_query_to_blob`, `generate_par`, and `put_object_in_store` functions/procedures are available and properly configured.

- The object store is properly set up and accessible.

Usage

To use the `export_query_to_object_store` procedure, simply call it with the SQL query you want to export as an argument.

```
BEGIN
    export_query_to_object_store('SELECT * FROM my_table');
END;
```

Customization

Before using this procedure, you need to modify the `export_query_to_object_store` procedure to replace the hardcoded placeholders with your actual values. These placeholders include:

- Object store prefix
- Base URL
- Tenant ID
- IDCS URL
- File name

You should update the procedure to use your specific values for these placeholders.

Example

Suppose you want to export the result of a query to an object store with a prefix of `my_prefix`, base URL of `https://example.com`, tenant ID of `my_tenant`, and file name of `my_file.csv`. You would need to modify the `export_query_to_object_store` procedure to use these values.

Once you have customized the procedure, you can use it to export your query results to the object store.

get_idcs_token

```
CREATE OR REPLACE FUNCTION get_idcs_token(
    p_idcs_url          IN VARCHAR2,
    p_idcs_client_id    IN VARCHAR2,
    p_idcs_client_secret IN VARCHAR2,
    p_scope_suffix      IN VARCHAR2
) RETURN VARCHAR2
IS
    l_response CLOB;
    l_token    VARCHAR2(4000);
    idcs_base_64_identity VARCHAR2(2000);
BEGIN
    idcs_base_64_identity :=
```

```

REPLACE(REPLACE(REPLACE(UTL_ENCODE.TEXT_ENCODE(p_idcs_client_id || ':' ||
p_idcs_client_secret, 'WE8ISO8859P1', UTL_ENCODE.BASE64), CHR(9)), CHR(10)),
CHR(13));

APEX_WEB_SERVICE.G_REQUEST_HEADERS.DELETE;
APEX_WEB_SERVICE.G_REQUEST_HEADERS(1).NAME := 'Authorization';
APEX_WEB_SERVICE.G_REQUEST_HEADERS(1).VALUE := 'Basic ' ||
idcs_base_64_identity;
APEX_WEB_SERVICE.G_REQUEST_HEADERS(2).NAME := 'Content-Type';
APEX_WEB_SERVICE.G_REQUEST_HEADERS(2).VALUE := 'application/x-www-form-
urlencoded; charset=UTF-8';

l_response := APEX_WEB_SERVICE.MAKE_REST_REQUEST(
    p_url          => p_idcs_url,
    p_http_method => 'POST',
    p_parm_name    => APEX_UTIL.STRING_TO_TABLE('grant_type:scope'),
    p_parm_value   =>
APEX_UTIL.STRING_TO_TABLE('client_credentials,rgbu:rpas:psraf-' ||
p_scope_suffix, ',')
);

IF l_response IS NULL THEN
    RAISE_APPLICATION_ERROR(-20001, 'Failed to retrieve IDCS token.
Response is null.');
```

```

END IF;

BEGIN
    APEX_JSON.PARSE(l_response);
    l_token := APEX_JSON.GET_VARCHAR2(p_path => 'access_token');
```

```

EXCEPTION
    WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20002, 'Failed to parse IDCS token
response: ' || SQLERRM);
END;
```

```

IF l_token IS NULL THEN
    RAISE_APPLICATION_ERROR(-20003, 'IDCS token is null.');
```

```

END IF;

RETURN l_token;
EXCEPTION
    WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20004, 'Error getting IDCS token: ' ||
SQLERRM);
END get_idcs_token;
```

generate_par

```

CREATE OR REPLACE FUNCTION generate_par(
    p_base_url          IN VARCHAR2,
    p_tenant            IN VARCHAR2,
    p_idcs_token        IN VARCHAR2,
    p_os_prefix         IN VARCHAR2,
    p_file_name         IN VARCHAR2
) RETURN VARCHAR2
```

```

IS
    l_response      CLOB;
    l_req_body       CLOB;
    l_access_uri     VARCHAR2(32767);
BEGIN
    -- Create the JSON request body for FTS invocation
    l_req_body := create_par_request_body(p_os_prefix, p_file_name);

    -- Get PAR from FTS service
    l_response := apex_web_service.make_rest_request(
        p_url          => p_base_url || '/' || p_tenant || '/
RetailAppsReSTServices/services/private/FTSWrapper/upload',
        p_http_method => 'POST',
        p_body         => l_req_body,
        p_headers      => apex_web_service.g_request_headers
    );

    -- Check if the response is not null
    IF l_response IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Failed to generate PAR: null
response');
    END IF;

    -- Parse the response
    BEGIN
        APEX_JSON.parse(l_response);
    EXCEPTION
        WHEN OTHERS THEN
            RAISE_APPLICATION_ERROR(-20002, 'Failed to parse PAR response: '
|| SQLERRM);
    END;

    -- Get the access URI from the response
    l_access_uri := APEX_JSON.get_varchar2(p_path => 'parList[%d].accessUri',
p0 => 1);

    -- Check if the access URI is not null
    IF l_access_uri IS NULL THEN
        RAISE_APPLICATION_ERROR(-20003, 'Failed to generate PAR: null access
URI');
    END IF;

    RETURN l_access_uri;
EXCEPTION
    WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20004, 'Failed to generate PAR: ' ||
SQLERRM);
END generate_par;

```

put_object_in_store

```

CREATE OR REPLACE PROCEDURE put_object_in_store(
    p_access_uri  IN VARCHAR2,
    p_blob_data   IN BLOB
) AS

```

```

        l_error_msg VARCHAR2(4000);
BEGIN
    BEGIN
        DBMS_CLOUD.PUT_OBJECT(
            object_uri => p_access_uri,
            contents   => p_blob_data
        );
    EXCEPTION
        WHEN OTHERS THEN
            l_error_msg := 'Error uploading object to store: ' || SQLERRM;
            -- Log the error
            DBMS_OUTPUT.PUT_LINE(l_error_msg);
            -- Re-raise the error
            RAISE_APPLICATION_ERROR(-20001, l_error_msg);
    END;
END put_object_in_store;

```

export_query_to_blob

```

CREATE OR REPLACE FUNCTION export_query_to_blob(
    p_sql_query IN VARCHAR2
) RETURN BLOB
IS
    l_context apex_exec.t_context;
    l_export apex_data_export.t_export;
    l_blob_data BLOB;
BEGIN
    BEGIN
        l_context := apex_exec.open_query_context(
            p_location => apex_exec.c_location_local_db,
            p_sql_query => p_sql_query
        );
    EXCEPTION
        WHEN OTHERS THEN
            RAISE_APPLICATION_ERROR(-20001, 'Failed to open query context: '
|| SQLERRM);
    END;

    BEGIN
        l_export := apex_data_export.export (
            p_context => l_context,
            p_format => apex_data_export.c_format_csv
        );
    EXCEPTION
        WHEN OTHERS THEN
            apex_exec.close(l_context);
            RAISE_APPLICATION_ERROR(-20002, 'Failed to export data: ' ||
SQLERRM);
    END;

    l_blob_data := l_export.content_blob;

    apex_exec.close(l_context);

    RETURN l_blob_data;

```

```
EXCEPTION
  WHEN OTHERS THEN
    IF l_context IS NOT NULL THEN
      apex_exec.close(l_context);
    END IF;
    RAISE;
END export_query_to_blob;
```

C

Transferring Table Data

This chapter outlines the steps to move table data from one Oracle Autonomous Data Warehouse (ADW) database to another using Oracle Cloud Object Storage and a pre-authenticated request (PAR) URI with read/write permissions.

Assumptions

- You have a list of table names to export.
- You have a **read/write prefix-level PAR URI** for an Oracle Object Storage bucket.
- You want to export each table as **JSON** – **CSV** has issues with null in the last column.
- Each table's data will be stored under a URI path like <PAR_URI>/table_name.
- You will use DBMS_CLOUD.EXPORT_DATA for export and DBMS_CLOUD.COPY_DATA for import.
- Multi-part and timestamped files will be generated during export.
- You are importing into existing tables with matching structures in the destination ADW.

Step 1: Export Tables from Source ADW

Use DBMS_CLOUD.EXPORT_DATA to export each table to the Object Store.

Example PL/SQL Block

```
DECLARE
  c_base_uri VARCHAR2(4000) := '<PAR URI>/<prefix>'; -- without trailing slash
  c_table_list SYS.ODCIVARCHAR2LIST :=
    SYS.ODCIVARCHAR2LIST('ITEM_LOC_SOH_EOD', 'ITEM_LOC_SOH', 'ITEM_LOC');
  l_columns CLOB;
  l_query CLOB;
BEGIN
  FOR i IN 1 .. c_table_list.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('outputing ' || c_table_list(i) || ' to uri: ' ||
      c_base_uri || '/' || LOWER(c_table_list(i)));

    -- To guarantee a specific ordering of columns, you can use the ORDER BY
    -- clause within the JSON_ARRAYAGG function.
    SELECT LISTAGG(column_name, ', ')
      WITHIN GROUP (ORDER BY column_id)
      into l_columns
    FROM user_tab_columns
    WHERE table_name = c_table_list(i);

    -- build query
    l_query := 'SELECT ' || l_columns || ' FROM ' || c_table_list(i);
```

```

DBMS_OUTPUT.PUT_LINE('columns: ' || l_columns);
DBMS_OUTPUT.PUT_LINE('query: ' || l_query);
DBMS_CLOUD.EXPORT_DATA(
    credential_name => NULL, -- PAR requires no credential
    file_uri_list   => c_base_uri || '/' || LOWER(c_table_list(i)),
    format          => JSON_OBJECT('type' VALUE 'json'),
    query           => l_query
);
END LOOP;
END;

```

This generates files in object store like:

```

<PAR_URI>/orders_1_<timestamp>.json
<PAR_URI>/orders_1_<timestamp>.json
... and so on

```

Step 2: Import Tables to Destination ADW

Use `DBMS_CLOUD.COPY_DATA` to import each exported table using wildcards to match multi-part, timestamped files. The destination tables already exist. Data is inserted into the destination table.

Example PL/SQL Block

```

DECLARE
    c_base_uri VARCHAR2(4000) := '<PAR URI>/<prefix>'; -- same PAR URI used for
    export
    c_table_list SYS.ODCIVARCHAR2LIST :=
    SYS.ODCIVARCHAR2LIST('ITEM_LOC_SOH_EOD', 'ITEM_LOC_SOH', 'ITEM_LOC');
    l_columnpath CLOB;
BEGIN
    FOR i IN 1 .. c_table_list.COUNT LOOP
        -- To guarantee a specific ordering of columns, use the ORDER BY
        -- clause within the JSON_ARRAYAGG function. Column order in
        -- columnpath must match table column ordering. First column in
        -- columnpath matches first column in table.
        SELECT JSON_ARRAYAGG('$.' || column_name)
            WITHIN GROUP (ORDER BY column_id)
            INTO l_columnpath
        FROM user_tab_columns
        WHERE table_name = c_table_list(i);

        DBMS_OUTPUT.PUT_LINE('Importing ' || c_table_list(i) || ' to uri: ' ||
c_base_uri || '/' || LOWER(c_table_list(i)));
        DBMS_CLOUD.COPY_DATA(
            table_name      => c_table_list(i),
            credential_name => NULL,
            file_uri_list   => c_base_uri || '/' || LOWER(c_table_list(i)) || '*',
            format          => JSON_OBJECT('type' VALUE 'json', 'columnpath' VALUE
l_columnpath)
        );
    END LOOP;
END;

```

```
END LOOP;  
END;
```

Notes

- The file_uri_list uses a wildcard * to match all exported file parts.
- Ensure the target tables already exist in the destination ADW with compatible structure.
- This process requires that the PAR remains valid and unexpired during export and import.
- Use LOWER(r.table_name) to maintain consistency in file naming.

Optional Enhancements

- Add error handling and logging.
- Add support for column.
- Add a control table to track exports and imports.

D

Data Access Schema (DAS) vs. Retail Data Store (RDS)

Why, What, How

When customers move from the legacy **Data Access Schema (DAS)** to the **Retail Data Store (RDS)**, they are shifting responsibility for infrastructure, replication, and database operations. This document helps two audiences:

- **Oracle Staff** who support customer transitions.
- **Customers** who currently use DAS and are evaluating or planning moving to RDS.

What is the Same

Area	DAS & RDS
Replication technology	Both use Oracle GoldenGate to replicate data from Retail applications.
Data freshness model	Replication is generally near real time; lag grows with large transactions.
Read-only access to application data	Customers cannot update source application data in either model.
Schema change buffering	Views insulate customers from most base schema changes.
Customer responsibility for extensions	Customers own and maintain their custom objects and code.
Environments	Prod + Stage included, additional '1 NPE's are optional.

What is Different

Area	DAS	RDS
DB Platform	ADW (recommended) or ATP	ADW
Service Level	SLA determined by Customer and OCI PaaS SLA	TSA, RTO, RPO based on SaaS SLA
Data Schemas	MFCS, SIOCS, CE	MFCS, SIOCS, CE, OB, XO, SE, BC, RICS, RSCH, OM
Database Ownership	Customer provisions and manages the target database.	Oracle provisions and manages an Autonomous Data Warehouse.
Users & Schemas	Customer can create new users/schemas.	Schema set is predefined and immutable . Customers cannot create new users or schemas.
Subscription	PaaS (storage, compute), GoldenGate additional	SaaS (storage, compute), GoldenGate included

Area	DAS	RDS
Included Tools	APEX, ORDS, SQL Dev Web, SQL Dev Desktop	APEX, ORDS
DB Connection	Web Access, others with supported database connection	Web Access Only, Private Endpoint (additional)
Integration	ORDS, APEX, ADW Adapter, others with supported database connection.	ORDS, APEX, Export/Import through Object Storage
GoldenGate Management	Customer manages GoldenGate replication on the target, including the initial load.	Oracle manages GoldenGate end-to-end, including the initial load and synchronization of source and target.
Object Set	Customers can configure which objects to replicate.	Fixed set of objects replicated; not configurable.
Data Access Model	Tables and views are both exposed.	Only views are exposed, with naming like RDS_WV_<view>. The precise naming convention is cloud service specific but expect the names to differ from those in DAS.
Schema Structure	Replicated data and extensions may coexist in the same schema.	Replicated data and extensions are segregated: read-only application schemas vs. read/write extension schemas. Three auxiliary schemas (RDS_CUSTOM_1 , RDS_CUSTOM_2 , RDS_CUSTOM_3) are also provided with no default access .
Security & Privileges	Customer controls user accounts, roles, and grants. Security posture varies by implementation.	Oracle defines a static, least-privilege model . Customers have limited security responsibilities . Custom objects are not accessible outside the owning schema unless privileges are explicitly granted.
Move Support	N/A.	Oracle may provide recommendations, but customers are responsible for moving from DAS to RDS.
Capacity Planning	Fully managed by customer.	Oracle manages infrastructure, but customers must monitor storage and CPU consumption and request capacity increases.
Replication Scope & Integration	Customer-owned database can act as a GoldenGate source or target ; customers can also integrate other pipelines (ETL, database links, flat files).	RDS cannot be a GoldenGate target or source . External data can be brought in only through Object Store + DBMS_CLOUD, REST services, or OIC with Private Endpoint , and only into the extension schema.
Customer Role	Customers act as both DBA and developer — responsible for patching, backups, replication management, security model.	Customers act primarily as developers . Oracle handles DBA functions: patching, upgrades, replication, backups, security baseline.

Considerations When Moving from DAS to RDS

- **Expect renaming of views** — All replicated objects are exposed as views. The precise naming convention is cloud service specific but expect the names to differ from those in DAS.
- **Extensions must be separated** — Move custom tables, packages, procedures, and other database objects into the extension schema.

- **Plan for fixed data set** — Customers cannot replicate arbitrary objects into RDS.
- **Monitor consumption** — Oracle manages ADW, but customers must keep an eye on storage and CPU usage.

Key Takeaways

- RDS simplifies operations by providing a safe, stable foundation, but removes customer flexibility. The privilege model is static, replication is not configurable, and customers must adapt to being *developers* in a managed world rather than *DBAs*.
- Oracle will manage replication, initial loads, patching, and the security baseline.
- Customer responsibility is to:
 - build and manage access to needed extensions,
 - and monitor capacity.

E

CPU Utilization Metric Tile Example

This document provides a full working example of a PL/SQL-based REST service that exposes Oracle CPU metrics as a JSON response suitable for Oracle Retail Home dashboard tiles. It includes a line chart of CPU usage over time and a summary of average CPU utilization for the past hour.

Overview

The implementation draws on the `v$metric_history` and `v$parameter` views to retrieve and normalize CPU usage. It uses PL/SQL functions to generate the required JSON structure for visualization. The data is exposed through an anonymous block suitable for use in an APEX RESTful service.

Function: `get_cpu_count`

Purpose

This function queries the `v$parameter` view to determine the number of CPU cores available to the database instance. This count is used to normalize CPU usage metrics from Oracle's internal monitoring views (which report usage across all CPUs). By dividing usage by the CPU count, the result is expressed as a per-CPU percentage, making it easier to interpret across systems with different configurations.

Error Handling

If the parameter is not found or an unexpected error occurs, the function returns 0 to ensure downstream logic can continue gracefully.

```
CREATE OR REPLACE FUNCTION get_cpu_count RETURN NUMBER IS
    l_cpu_count NUMBER;
BEGIN
    SELECT TO_NUMBER(value)
        INTO l_cpu_count
        FROM v$parameter
        WHERE name = 'cpu_count';
    RETURN l_cpu_count;
EXCEPTION
    WHEN no_data_found THEN
        RETURN 0;
    WHEN OTHERS THEN
        RETURN 0;
END;
/
```

Function: get_rh_chart_data_json

Purpose

This function builds a JSON-formatted line chart showing CPU utilization over time. It uses v\$metric_history, which contains recent time-series metrics (typically at one-minute intervals), and converts each sample's end time into a human-readable format (HH24:MI). CPU usage values are normalized using get_cpu_count.

Key Features

- Uses JSON_ARRAYAGG to collect multiple data points into a single array.
- Wraps the data in a chart structure expected by Oracle Retail Home tiles.
- Includes metadata such as axis titles, tooltip rendering, and value formatting.

Design Note

Because the metric CPU Usage Per Sec returns usage across all CPUs in centiseconds (that is, 1/100 second), dividing by the CPU count yields per-core utilization (0.00–1.00 range, often rendered as a percentage with two decimal places).

```
CREATE OR REPLACE FUNCTION get_rh_chart_data_json RETURN CLOB IS
    l_json CLOB;
BEGIN
    WITH chart_data AS (
        SELECT TO_CHAR(m.end_time, 'HH24:MI') AS sample_time,
               ROUND((m.value / 100) / get_cpu_count(), 2) AS cpu_utilization_pct
        FROM v$metric_history m
        WHERE m.metric_name = 'CPU Usage Per Sec'
    )
    SELECT JSON_OBJECT(
        'type' VALUE 'line',
        'items' VALUE JSON_ARRAYAGG(
            JSON_OBJECT(
                'name' VALUE cd.sample_time,
                'series' VALUE 'CPU',
                'value' VALUE cd.cpu_utilization_pct
            ) ORDER BY cd.sample_time
        ),
        'renderTooltips' VALUE 'true',
        'valueFormat' VALUE 'PC',
        'renderXAxisLabels' VALUE 'true',
        'renderYAxisLabels' VALUE 'true',
        'xAxisTitle' VALUE 'Time (hours and minutes)',
        'yAxisTitle' VALUE 'CPU Utilization (%)'
    )
    INTO l_json
    FROM chart_data cd;

    RETURN l_json;
```

```
END;  
/
```

Function: get_rh_summary_json

Purpose

This function computes a single-value summary of CPU usage: the average CPU utilization across all data points in the last hour. Like the chart data, values are normalized using the number of CPUs.

Output Format

The function produces a JSON array with a single item:

```
{  
  "name": "Avg CPU Utilization (Last Hour)",  
  "value": <normalized average>,  
  "valueFormat": "PC"  
}
```

Use Case

This value appears prominently at the top of the metric tile as a high-level indicator of system load.

```
CREATE OR REPLACE FUNCTION get_rh_summary_json RETURN CLOB IS  
  l_json CLOB;  
BEGIN  
  WITH summary AS (  
    SELECT ROUND(AVG((m.value / 100) / get_cpu_count()), 2) AS  
      avg_cpu_utilization  
    FROM v$metric_history m  
    WHERE m.metric_name = 'CPU Usage Per Sec'  
  )  
  SELECT JSON_ARRAYAGG(  
    JSON_OBJECT(  
      'name' VALUE 'Avg CPU Utilization (Last Hour)',  
      'value' VALUE avg_cpu_utilization,  
      'valueFormat' VALUE 'PC'  
    )  
  )  
  INTO l_json  
  FROM summary;  
  
  RETURN l_json;  
END get_summary_json;  
/
```

Function: get_rh_cpu_utilization_json

Purpose

This function combines the output of the previous two functions into a single JSON object with two keys:

- items: Contains the summary.
- chart: Contains the line chart data.

Design Pattern

The approach ensures separation of concern: each sub-function is responsible for one part of the structure, and this final function composes them.

Why FORMAT JSON is Used

It preserves the embedded JSON structure when building the outer object, avoiding character-escaping issues.

```
CREATE OR REPLACE FUNCTION get_rh_cpu_utilization_json RETURN CLOB IS
    l_json CLOB;
    l_items CLOB;
    l_chart CLOB;
BEGIN
    l_chart := get_rh_chart_data_json();
    l_items := get_rh_summary_json();
    SELECT JSON_OBJECT(
        'items' VALUE l_items FORMAT JSON,
        'chart' VALUE l_chart FORMAT JSON
        RETURNING CLOB
    )
    INTO l_json
    FROM dual;

    RETURN l_json;
END get_rh_cpu_utilization_json;
/
```

Anonymous Block (REST Service Source)

Purpose

This anonymous PL/SQL block serves as the source code for an APEX RESTful service. When the REST endpoint is invoked, this block executes and returns a complete JSON response representing CPU usage data.

Key Elements

- `owa_util.mime_header('application/json')`: Sets the HTTP content type to JSON.
- `http.p('Cache-Control: no-cache')`: Disables caching to ensure fresh data.
- `http.prn(l_json)`: Prints the resulting JSON to the response body.

Context

This block is typically configured as the source of a GET method in Oracle APEX RESTful services. It bridges the backend metric data with the frontend visual tile.

```
DECLARE
    l_json CLOB;
BEGIN
    l_json := get_rh_cpu_utilization_json;

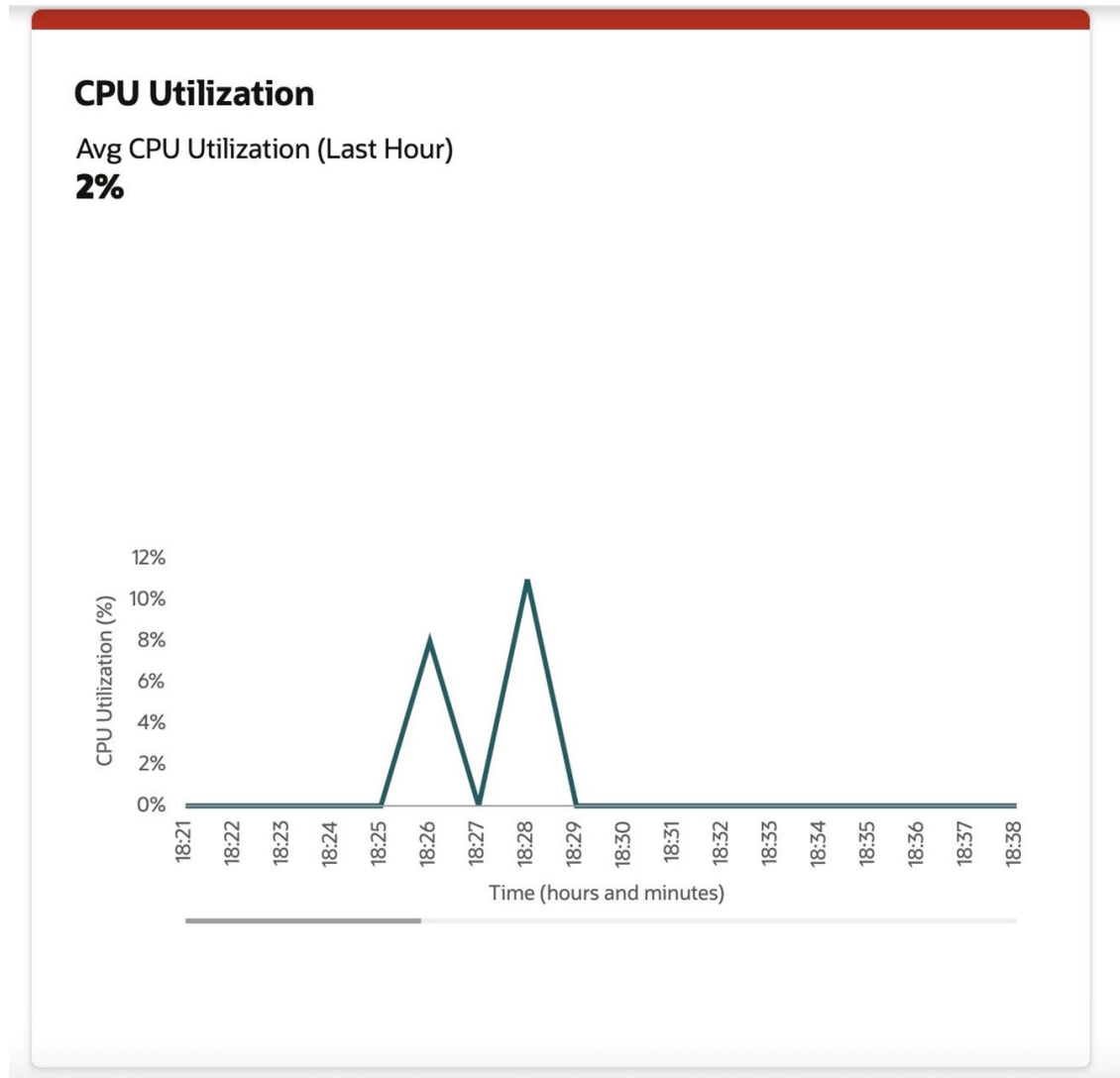
    owa_util.mime_header('application/json', FALSE);
    http.p('Cache-Control: no-cache');
    owa_util.http_header_close;

    http.prn(l_json);
END;
```

Example REST Service Response

```
{
  "items": [
    {
      "name": "Avg CPU Utilization (Last Hour)",
      "value": 0.02,
      "valueFormat": "PC"
    }
  ],
  "chart": {
    "type": "line",
    "items": [
      { "name": "18:26", "series": "CPU", "value": 0.08 },
      { "name": "18:27", "series": "CPU", "value": 0 },
      { "name": "18:28", "series": "CPU", "value": 0.11 },
      ...
    ],
    "renderTooltips": "true",
    "valueFormat": "PC",
    "renderXAxisLabels": "true",
    "renderYAxisLabels": "true",
    "xAxisTitle": "Time (hours and minutes)",
    "yAxisTitle": "CPU Utilization (%)"
  }
}
```


Figure E-1 CPU Utilization Tile Displaying Average and Time-Series Values Derived from the REST Service



Summary

This example illustrates how to:

- Extract and normalize CPU usage metrics from Oracle system views.
- Generate JSON structures compatible with Oracle Retail Home tiles.
- Implement an APEX RESTful service using PL/SQL to expose monitoring data.