

Oracle[®] Retail Data Store Implementation Guide



Release 22.1.302.0

F61324-02

August 2022

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE[®]

Copyright © 2022, Oracle and/or its affiliates.

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, and MySQL 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

Separation of Replicated and Custom Data	1-2
Example	1-2

2 Typical Implementation Events

3 Getting Started

APEX User Management	3-1
Data Visualization Access	3-3

4 Extension

RDS Extension Overview	4-1
RDS Architecture Basics	4-1
Environment Considerations	4-1
Prerequisites	4-2
Implementing a RESTful Service in APEX	4-2
URL Pattern	4-3
Handler	4-4
Security	4-5
Authentication	4-5
ORDS PRE-HOOK	4-5
Invoking a Data Service	4-6
Using RDS to Build Integration	4-6
Outbound Integration using a Data Service	4-8
Outbound Integration using Object Storage	4-9
Hypothetical Outbound Integration Problem	4-9
Retail Home Integrations	4-12
An Asynchronous Approach	4-14
Next Steps	4-14
Monitoring Resource Consumption in RDS	4-14

5 Storage and CPU Usage

6 Version Updates

7 Notes

APEX	7-1
Visual Builder Studio	7-1
APEX and Autonomous Databases	7-1

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

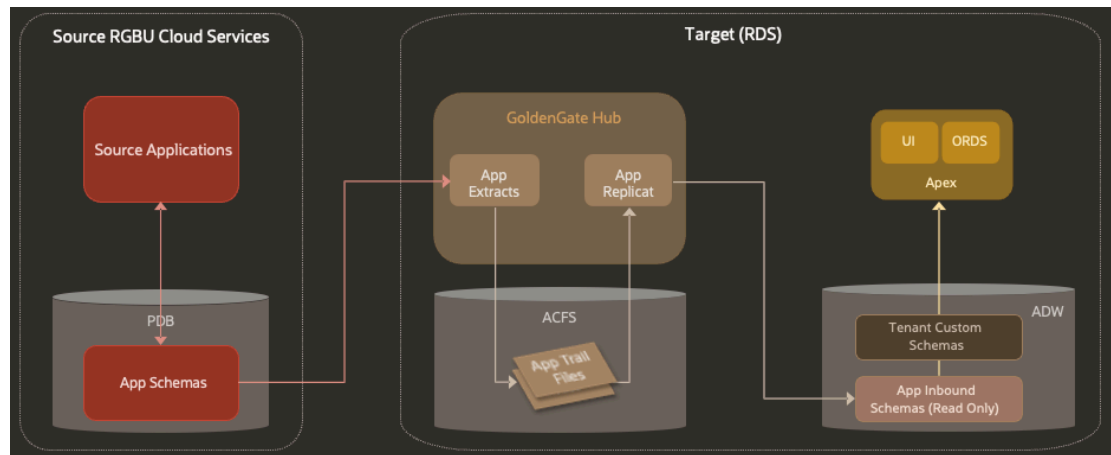
Implementation Overview

Oracle Retail Data Store (RDS) is a set of infrastructure and tools that allows you to build extensions on top of Retail application data without affecting the original Retail applications. These extensions can consist of database objects, web services, and user interfaces. This Implementation Guide describes the solution and provides information about how you can use RDS.

The core of RDS is a data replication implementation that uses Oracle GoldenGate to replicate application data from Retail applications to a centralized Autonomous Data Warehouse (ADW) database. The data is kept in sync with the source application database in near-real-time.

This data is made available through Oracle REST Data Services (ORDS) and Application Express (APEX) workspaces. When a retailer subscribes to RDS, they are given the URLs and credentials to access these workspaces.

Figure 1-1 Data Replication to RDS via GoldenGate



- **PDB** - Pluggable Data Base. The source applications in the RGPU that will be replicating to RDS store their data in pluggable database instances.
- **ACFS** - ASM (Automatic Storage Management) 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.

Separation of Replicated and Custom Data

The replicated application data is held in read-only schemas (one per source application schema). The ORDS and APEX workspaces have access to a read-write schema which can view the read-only schema's database objects. In the read-write schema, you are free to create any database objects you need to create, and you have read privileges to the replicated application data. When new database objects are created in the read-only schema (for example when a patch is applied to the source application), a scheduled database job in the RDS database grants the appropriate read permissions for those objects to the read-write schema. This job runs hourly.

Example

For Merchandising Foundation Cloud Service, an ORDS workspace is available that grants access to the MFCS_RDS_CUSTOM schema. This schema is initially empty, but allows creation of database objects, APEX applications, etc. This schema also has read permissions to database objects in the MFCS_RDS schema, which is where the actual replicated data resides. A customer can use the ORDS workspace to create REST data services that can read the tables with replicated data, or can read and write any custom tables that have been created. A customer can also build APEX applications on top of the custom tables; the read-only replicated tables can be read by the APEX application, but cannot be modified.

Each Retail application controls what data it replicates to the RDS database. Refer to each application's product documentation for details about the data that is made available in RDS.

2

Typical Implementation Events

In any implementation including RDS, there are many steps along the way before a system is running.

- 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.
- Data Seeding via Data Pump
 - The next step is creating an initial data load into RDS from the source application using Oracle Data Pump tools. This step is done by Oracle when the retailer indicates they are ready to move forward.
 - A prerequisite to this step is that the source application must have data ready to be replicated; this may be an involved process depending on the application in question. Refer to documentation for the source application.
 - The result of this step is that a baseline set of data has been replicated from the source application to the RDS read-only schema.
- GoldenGate Hub Configuration
 - A GoldenGate Hub instance is configured to replicate data from the source application'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 applications' database.
- Extension
 - In this step, the retailer uses the tools that are part of RDS to build the custom extensions they need.

3

Getting Started

Once RDS is provisioned, the following APEX workspaces are available to use:

Table 3-1 APEX Workspaces

Workspace Name	Source Cloud Service
MFCS_RDS_CUSTOM	Merchandising Foundation Cloud Service
CECS_RDS_CUSTOM	Customer Engagement Cloud Service
SIOCS_RDS_CUSTOM	Store Inventory Operations Cloud Service



Note:

These workspaces are available even if you have not subscribed to the associated cloud services, but they contain no database objects or replicated data.

You can access these workspaces by navigating to the workspace login page for your environment. The URL for this will be delivered to you after provisioning is complete, and follows the pattern:

`https://<base URL>/<environment ID>/ords/`

For example:

`https://ocacs.ocs.oc-test.com/nryfhvvl5ka2su3imnq6/ords/`

You are then able to log in to one of the workspaces listed above using the credentials that have been supplied to you.

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. Users with Developer role can create and edit APEX applications while Workspace Administrators can do that as well as manage the application lifecycle and workspace settings.

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 20.4 Establishing User Identity Through Authentication.

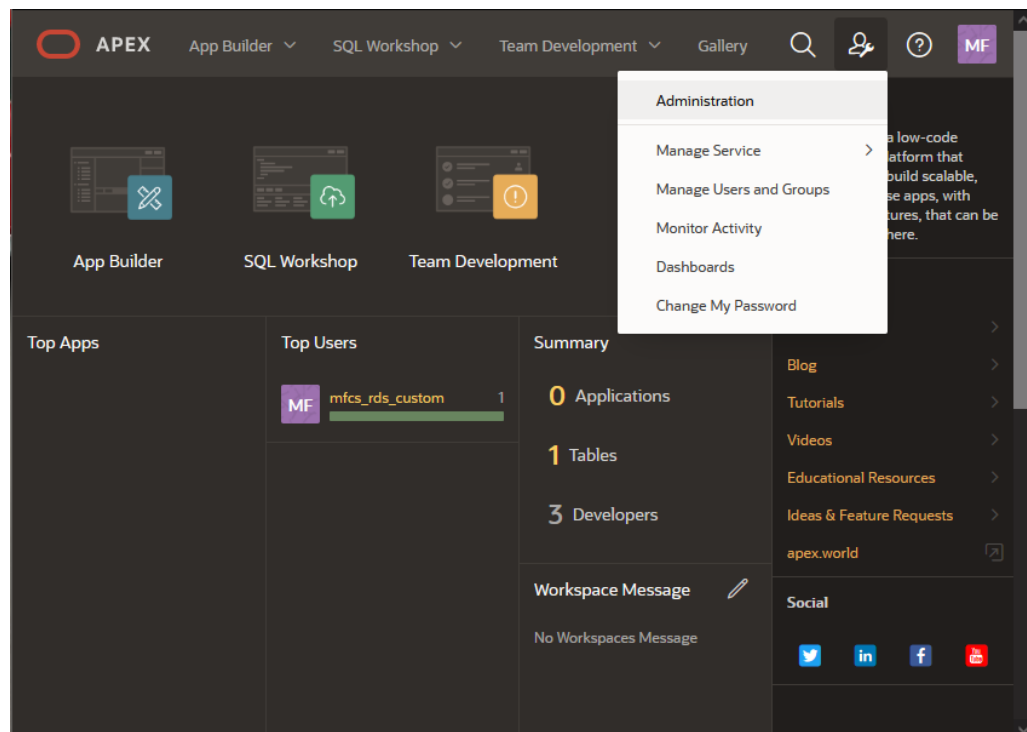
Development user authentication is provided through integration with IDCS. The APEX Workspaces provisioned for RDS are configured to use HTTP Header Variable authentication. For full details on this model, please refer to the *APEX App Builder User's Guide*, section 20.4.3.4 HTTP Header Variable.

Once provisioned, each workspace comes with a single user. This user is the Workspace Administrator for that workspace. For initial access, each Workspace Administrator account must have a matching username in IDCS. The Workspace Administrator account passwords and their lifecycle will then be managed in IDCS going forward. There is no need to synchronize this user with APEX. The only requirement is the usernames match.

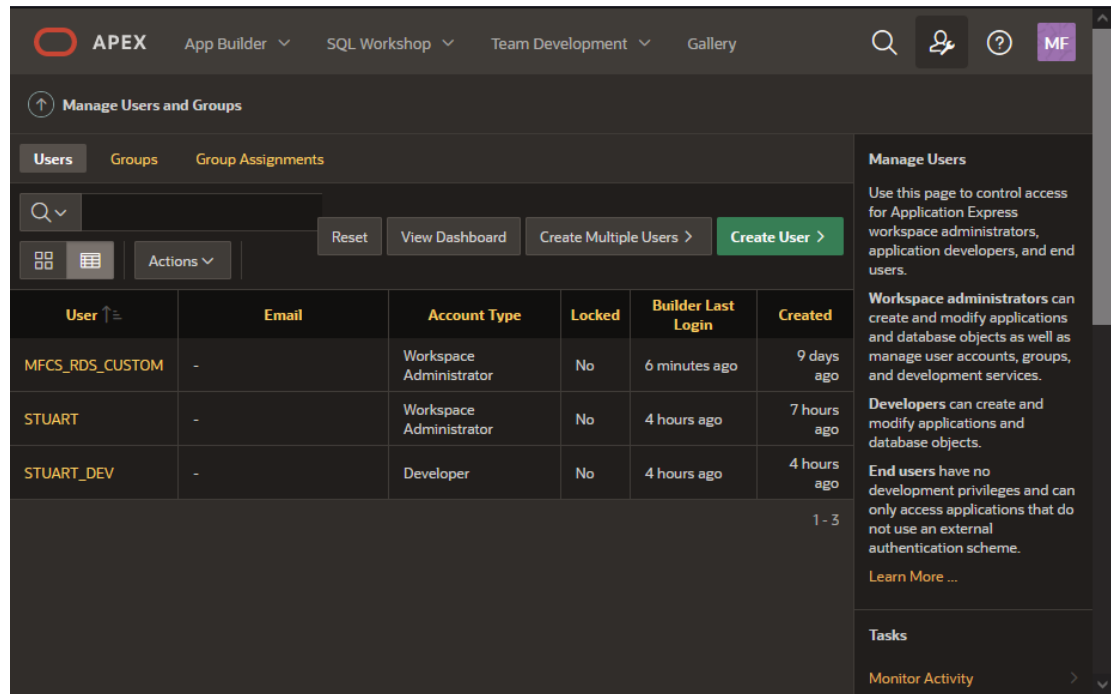
This is the set of Workspace Administrators provided with this release:

- MFCS_RDS_CUSTOM
- CE_RDS_CUSTOM
- SIOCS_RDS_CUSTOM

In most cases, teams will need to create additional development users in these workspaces 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 IDCS. Like the default Workspace Administrator accounts, these new accounts will have their passwords live in IDCS. For the APEX user creation, use the workspace's Administration menu in the top right corner to access *Manage Users and Groups*.



Create the users needed by selecting the *Create User* button and filling in the form.



User	Email	Account Type	Locked	Builder Last Login	Created
MFC5_RDS_CUSTOM	-	Workspace Administrator	No	6 minutes ago	9 days ago
STUART	-	Workspace Administrator	No	4 hours ago	7 hours ago
STUART_DEV	-	Developer	No	4 hours ago	4 hours ago

For full details, please refer to the *APEX Administration Guide*.

Data Visualization Access

RDS is provisioned with Oracle Analytics Server Data Visualization capabilities. You can access these capabilities by navigating to the Data Visualization URL for your environment. The URL for this will be delivered to you after provisioning is complete, and follows the pattern:

`https://<base URL>/<environment ID>/dv/`

For example:

`https://ocacs.ocs.oc-test.com/nryfhvv15ka2su3imnq6/dv/`

This URL is protected by the same IDCS instance and fully supports single sign on.

For full details on taking advantage of Data Visualization in RDS, please refer to the Visualizing Data in Oracle Analytics Server documentation.

4

Extension

RDS Extension Overview

RDS Architecture Basics

The defining feature of RDS is the data of each participating product resides in a single dedicated read-only schema, e.g., MFCS. Product data is made accessible to the customer in a dedicated companion, writeable schema using synonyms. All custom data objects are created in this companion, writeable schema. Management and retention of custom data objects is wholly the responsibility of the customer.

The product data in RDS is a replica of selected data residing in an operations system such as MFCS. RDS is not part of MFCS, but a repository of MFCS data. Moreover, the data exchange is one way from MFCS to RDS. Any data movement, directly or indirectly, from RDS to MFCS is orchestrated by the customer.

Although data from multiple products reside in RDS, there is only an informal guarantee that if there are no updates to a given set of data items, then eventually all accesses will return temporally consistent results. What this statement means is that after sufficient time has passed, RDS accurately reflects the state of the enterprise at some point in time in the recent past (recent could be measured in seconds, minutes, or hours). What qualifies as sufficient time depends on the temporal consistency of the separate subsystems that make up the enterprise, which depends metaphorically speaking on when each system closes its book. Temporal consistency also depends on the replication lag, which varies depending on system loading. This lag, however, is expected to be minimal under normal operating conditions. Temporal consistency may prove decidedly less relevant than semantic and data model differences between the products that reflect the specific problems each product was devised to solve.

Refer to your product data model to determine what data is available in RDS. Bear in mind, the data is a replica of inserts and updates as well as deletes. The point is, the data retention policy in RDS is effectively replicated from the operations system.

Environment Considerations

When embarking on the customization of a product, it is important to understand how the RDS implementation environment, which is a SaaS offering, differs from PaaS and on-premises. First of all, some or all product customization will be accomplished by making modifications to RDS (the product implementation guide will provide details on product customization). Those modifications are achieved using [APEX](#).

APEX is a low code development environment. As a result it does not anticipate the need for (and does not provide) development life cycle tools. Application user interfaces are

composed in an application builder. RESTful services are built in a similar fashion. In fact, one constructs most database objects using a UI rather than by executing code. One can, however, use the [SQL Workshop](#) to compose small amounts of PL/SQL (e.g., 100s to 1000s of lines of code). There is no access to SQL*Developer or SQL*Loader. In fact, most consoles are unavailable. It is an ideal environment for most business savvy users, but may be foreign to the skilled PL/SQL, front end, or back end developer. It is important to note that customizations that require coding will use SQL and PL/SQL. Moreover, most data interchange will rely on JSON formatted messages. All the examples in this document will employ JSON.

When using APEX, SQL command line type activities are performed in the [SQL Commands](#) tool within the [SQL Workshop](#). For SQL script development (for blocks of code where reuse is anticipated), however, one uses the [SQL Scripts](#) tool.

When using APEX, one logs into a workspace and that workspace provides access to a single schema. Specifically, one can have access to the data for a single product within a workspace. In other words, it is not possible to execute a mutli-schema or cross-schema query from within a workspace. If one needs to combine information from multiple products, then one constructs schema specific integrations and then joins that information externally.

Lastly, it is important to remember that since RDS is a SaaS offering, some tools and features may not be available or availability may be provided with some limitations. It is important that one understand the dependencies inherent in customizations that one wishes to migrate. Expect to review these dependencies with an Oracle Representative.

Prerequisites

In order to fully explore the examples below, one will need to meet the following prerequisites:

- Have access to an APEX workspace within an RDS tenant,
- IDCS application credentials
- Have access to a suitable Object Storage instance

APEX is a browser-based application; so, all that is needed is a URL and the necessary authorization to gain access to an APEX workspace. One's RDS APEX workspace admin will be able to provide the necessary details. IDCS application credentials are needed to generate an access token that can be used to authenticate a RESTful service invocation. An object storage instance is needed as a data export destination.

Implementing a RESTful Service in APEX

Oracle RESTful Data Services play a role in both outbound data service and data export integration patterns. For the data service integration pattern, a RESTful service synchronously returns the requested data in whole or part (i.e., through data pagination). For the data export integration pattern, a RESTful service asynchronously initiates a data export and then returns with a suitable response. Much of the RESTful service implementation is the same regardless of data integration pattern. Ultimately, from the perspective of the RESTful service implementation, the two integration

patterns differ only in terms of the actions taken in response to the REST service invocation.

Chapter 7 of the SQL Workshop Guide, Enabling Data Exchange with RESTful Services describes in great detail how one creates a RESTful service in APEX. Bear in mind, the above links are 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. In any case, the following paragraphs will only provide an overview of how one creates the necessary RESTful service. Consult the documentation for additional details and example implementations.

The implementation of a RESTful service as two aspects, a URL pattern and a handler. When designing or composing a URL pattern, one is effectively constructing a dispatch mechanism that ties a given URL pattern to a specific action. This pattern-action pairing represents an API through which external systems access customer data residing in RDS.

URL Pattern

The URL pattern consists of three parts: a base path, a module name, and a URI template. The base path is the same for all RESTful services created within a given workspace, namely:

```
https://<host name>/<tenant_name>/ords/<schema name>
```

where, the host name and tenant name are the same for all a given tenant's RESTful services. The schema name is the workspace schema (remember, there is a one-to-one relationship between workspace and schema). The module name is just a hierarchical organizing feature. For example, one could have a module called po for RESTful services associated with purchase orders in MFCS. If this were the case, the path for all purchase orders services would begin with the following:

```
https://<host name>/<tenant_name>/ords/mfcs/po/
```

The last part of the URL pattern is the URI template. The template consists of 0 or more path components followed by 0 or 1 bind variable. If there are 0 path components and no binding variable, then the URL template is blank or an empty string. If this were the case, then the complete URL is:

```
https://<host name>/<tenant_name>/ords/mfcs/po/
```

In the more typical case, there are one or more path components with and without a bind variable. For example, one might have the following purchase order services:

Service	Base Path + Module	URI Template	Description
active	.../mfcs/po/	active	Return PO numbers for active (not completed or active) purchase orders.
complete	.../mfcs/po/	complete	Return PO numbers for completed purchase orders.
expired	.../mfcs/po/	expired	Return PO numbers for purchase orders that have expired with being fulfilled.

Service	Base Path + Module	URI Template	Description
summary	.../mfcs/po/	summary/:po_num	Return a summary for the purchase order with the given number.
detail	.../mfcs/po/	detail/:po_num	Return the details for the purchase order with the given number.

If one wanted purchase orders within a given date range, a query string specifying a date range can be added. For example, the following would return PO numbers between May 1st and May 15th, 2022.

```
https://<host name>/<tenant_name>/ords/mfcs/po/active?
from=2022-05-01&to=2022-05-15
```

The from and to become bind variables in a query.

Handler

The second aspect of a RESTful service is the handler. In any case, each URL pattern can be associated with up to four handlers, one for each of the following HTTP methods: GET, POST, PUT, and DELETE.

The source is parameterized using bind variables. Explicit bind variables are part of the URL, e.g.: `po_num`, `from`, and `to`. Consider the implementation of the handler that returns the details for a purchase order. The URL pattern might be:

```
https://<host name>/<tenant_name>/ords/mfcs/po/details/:po_num
```

The GET handler for this URI pattern could then be:

```
select ... from purchase_orders where po_num=:po_num;
```

Ideally, `purchase_orders` is a view that is identified as part of the public API of the product data model in RDS. In general, accessing a view versus a table is preferable because the product makes stronger guarantees regarding the immutability of views (i.e., they are less likely to change in inconvenient ways).

Query string parameters can also be used to parameterize the RESTful service handler. The query string parameters are used like the `:po_num` above. For example, one can use the *from-to* range in the URL above to construct a query that returns only purchase orders within a data range, e.g.:

```
select ... from purchase_orders where
supplier_id=:supplier_id and
order_date >= DATE :from and
order_date < DATE :to;
```

It is important that RESTful services respect the semantics of the methods used. The GET method strongly suggests a synchronous operation with no side effects. Whereas

the semantics of POST, PUT, and DELETE methods are expected to have side effects, but could be synchronous or asynchronous.

RESTful data services can be paged. Paging can be useful for interactive operations that only require a subset of data, but it can also be used to return a multi-part result set. In either case, page data returns a subset of data as well as URLs for the previous and next result set.

Lastly, only explicit bind variables have been described. The term explicit is used because there are also a number of implicit bind variables, see [Implicit Parameters](#). These include parameters that provide access to bind variables that implement pagination. They also provide access to the data portion or payload of an endpoint invocation. There is also a `current_user` parameter. This parameter, however, is the schema owner, not the user accessing the RESTful service.

Security

Authentication

Authentication is achieved using OAUTH 2. What this means is that a RESTful service invocation is authenticated using an access token. That access token can be obtained, for example, using curl as shown in Listing 1. The response is a JSON object. This access token then becomes the Bearer token that authorizes the RESTful service invocation.

LISTING 1: ACCESS TOKEN GENERATION

```
curl --location --request POST 'https://<idcs 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:idm:__myscopes__'
```

There is no implicit data or row level security. Any such security that is in effect in MFCS, for example, will not exist in RDS unless explicitly implemented by the customer. An effective approach to implementing data security will employ the ORDS Pre-hook functionality.

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](#).

Invoking a Data Service

A GET data service can be accessed and tested directly from a browser. If one is not already logged into the Oracle Cloud, then one will be redirected to a cloud login screen for authentication. Once authenticated, the data service endpoint will be invoked. The results of the invocation will be displayed in the browser window. If the data service is accessed from Retail Home, for example, the user is already authenticated.

Invoking a POST RESTful service or invoking a RESTful service from an external system is slightly more complicated. Listing 2 illustrates the invocation of a REST service endpoint using curl. Note the use of the access token. The response is dependent on the handler action.

LISTING 2: DATA SERVICE INVOCATION

```
curl --location --request POST 'https://<host name>/<tenant id>/ords/
mfcs/pom/job_start' \
--header 'content-type: application/json' \
--header 'Authorization: Bearer <access token>' \
--data-raw '{"param1": "value1", "param2": "value2"}'
```

Using RDS to Build Integration

There are two approaches to transferring data between RDS and an external system, use a data service or use object storage to mediate the transfer. For an outbound integration, one uses a GET data service or exports data from ADW to object storage. For an inbound integration, one uses a POST (or DELETE if appropriate) data service or imports data from object storage to ADW. In the case where object storage is mediating the transfer, the external system is responsible for uploading or downloading data from object storage. When deciding which approach is best for a given situation, one should consider the latency tolerance, data volume, and tolerance for complexity.

Figure 4-1 Data Integration Option Matrix

		Latency Tolerance	
		Low	High
Data Volume	Low	Data Service	Data Service or Object Storage Mediated Transfer
	High	Data Service with multi-part result	Object Storage Mediated Transfer

Latency tolerance is essentially a measure of how long the consuming process or agent can or will wait for the requested data to be made available. The most obvious low tolerance scenario is an interactive setting where latency tolerances are in the sub-second range. A less obvious scenario is one where the objective is to control some automated process in which responsiveness is critical. On the other hand, the typical retail ecosystem encompasses a multitude of automated processes that have a relatively high latency tolerance so long as the latency does not adversely impact process SLAs.

It is important to note that ORDS has a latency tolerance of 300 seconds. That is, the RESTful service source execution must complete within 300 seconds or a Socket Hung Up error will be returned. The 300 second threshold is a hard limit. Bear in mind that DB execution time jitter, which is the result of DB load variation, can increase typical execution times. Parameterized RESTful services may also exhibit a wide variation in execution time. Building a reliable service may entail introducing a retry component. In some cases, it may be necessary to use an asynchronous execution approach. An asynchronous approach is called for whenever the execution time for representative use cases plus jitter is likely to exceed the 300 second boundary.

The second factor one considers when selecting an integration approach is data volume. Distinctions between high and low volumes are best measured in terms of time rather than bytes. Essentially, the amount of data one can rapidly move from one place to another is ever increasing; however, what one considers long versus short transit times, though subjective, remain somewhat constant. For example, low data volumes are associated with transit times measure in seconds whereas high data volumes are associated with transit times measured in minutes.

Though both latency tolerance and data volume are best measured using time, they are measuring different things and are useful criteria when selecting an integration approach for a given problem. Figure 1 presents a data integration option matrix that provides a conceptual framework for deciding which approach, data service or object store mediated transfer, is best suited to a given problem. Although the figure is for the most part self-explanatory, data service with paged result warrants some discussion. Paged results concern outbound integrations only. These integrations produce JSON formatted results where each page returns one part of a multi-part result as well as URLs for the next and previous page.

The last factor one might consider is complexity. Creating a synchronous data service is relatively simple whereas implementing a data export is quite a bit more complicated. A synchronous data service includes a RESTful service implementation and a query implementation. A data export service (a service is used to initiate the export) includes everything that a data service includes as well as export data, managing credentials, and managing and monitoring export processes. Sometimes, however, complexity is not the most important factor when selecting a data integration pattern.

The remainder of this document will discuss how to implement outbound integration in RDS using a data services pattern and a data export pattern. Bear in mind, the examples provided below are illustrative. They are not robust. They do not implement error handling. The goal is to familiarize the reader with the parts of the problem, the tools one uses, and the relevant Oracle documentation.

Before the general features of the outbound integration problem are introduced, the reader should familiarize themselves with the basics of the RDS architecture and environment as well as prerequisites for implementation.

Outbound Integration using a Data Service

The vast majority of what one needs to know to implement a data integration using a RESTful service is described in the previous section. This short section will discuss some of the finer points of implementing actions. There are several key points to remember:

- A handler's PL/SQL source is not compiled when the handler is saved. Handler source compilation errors reveal themselves when the REST service is invoked. So keep the handler source as simple as possible and avoid an approach that frequently changes to handler source. Essentially, do something simple and then call a procedure or function to do the majority of the work.
- If the result is not returned in the handler source as the direct result of a query, one can compose a JSON object (see APEX_JSON and JSON Data and Oracle Database) and return it using `http.print`.
- A handler's source is not reusable short of copy and paste. If the handler source is likely to be reusable, then encompass it in a procedure or function.

- One can compose procedures, functions, test scripts, and such in the SQL Workshop using the SQL Commands and SQL Scripts tools.

Outbound Integration using Object Storage

As mentioned above, it is substantially easier to meet data integration needs using a data service pattern than it is using a object storage pattern. The data service pattern in the simplest case requires just a data service. Whereas the RESTful service in the object storage pattern moves the data only part way to its destination since the service achieved its end when the data arrives in object storage. Moreover, the data export pattern generally calls for an asynchronous start because data exports are likely to be long running. With the asynchronous start comes the requirement that one now manage and monitor the export — if for no other reason than to trigger the process that moves the data from object storage to its final destination. The remainder of this example, however, will begin with the description of a hypothetical, short running, synchronous data export approach in order to avoid introducing all the complexities of an asynchronous start. The section will conclude with a brief overview of how one would manage long running asynchronous exports.

Hypothetical Outbound Integration Problem

Consider the following hypothetical problem. The customer wants to export changes to the item master (i.e., RDS_WV_ITEM_MASTER). The customer will use the `csn_nbr` column to keep track of what has been exported. Specifically, the customer will select rows whose `csn_nbr` is greater than the max `csn_nbr` of the previously exported rows. The export process is assumed to be idempotent.

Exporting changes to the item master is a task well suited to a synchronous data service. The item master changes slowly and the volume of data needed to capture those changes is relatively small. If, however, the export concerned transactions, a synchronous data service may or may not be appropriate depending on the data volumes.

One would begin by creating an item master module — in the SQL Workshop RESTful Services Tool — with a name such as `item_master`. The presumption is that there may be multiple item master RESTful services and using this approach to naming anticipates that possibility. Next, one creates a URI template with a template of “changes/:last_csn_nbr.” The first part of the template, `changes`, identifies the function of the endpoint. The last part of the template, `:last_csn_nbr`, is a bind variable that will be used in a query. The last step is to create the GET handler for the service.

The most common GET handler has a source type of collection query. If the source type is a collection query, then all one needs to do is provide a query as the source and ORDS will take care transforming the query result into a JSON string. In order to get the most recent changes to the item master one would use the following source for the GET handler:

```
select ... from RDS_WV_ITEM_MASTER where csn_nbr > :last_csn_nbr
```

The endpoint one would use to get recent changes is:

```
https://<host>/<tenant>/ords/mfcs/item_master/changes/26771905065
```

where 26771905065 is the maximum CSN number of the items thus far retrieved. This design makes it the responsibility of the caller to keep track of the maximum CSN number used. With

that in mind, one needs to make sure that the query returns the `csn_nbr`. Without it, it will not be possible to keep track of the maximum CSN number. The initial item master will have items with NULL valued CSN numbers. One could obtain all the changes since the initial load using the following endpoint:

```
https://<host>/<tenant>/ords/mfcs/item_master/changes/0
```

The final part of the GET handler that will be discussed is pagination. Using the pagination, one can retrieve the query result in pages of a specified number of rows. The default pagination size is set at the module level, but can be overridden at the handler level. All results are paginated; however, one can set the page size such that no more than one page is ever returned. The JSON object that is returned has the following form:

```
{
  "items": [...],
  "hasMore": true,
  "limit": 25,
  "offset": 0,
  "count": 25,
  "links": [
    {"rel": "self", "href": ".../item_master/changes/26771905065"},
    {"rel": "describedby", "href": ".../metadata-catalog/item_master/changes/item"},
    {"rel": "first", "href": ".../item_master/changes/26771905065?limit=25"},
    {"rel": "next", "href": ".../item_master/changes/26771905065?offset=50&limit=25"},
    {"rel": "prev", "href": ".../item_master/changes/0"}
  ]
}
```

The `items` value is a list of JSON objects where the keys are column names in the query result. The `hasMore` value indicates whether there is more data. The `limit` specifies the limit used in the query. The `offset` is the row offset of the first row returned. `Count` is the number of rows actually returned. The `links` value provides URLs for the first, next, and previous pages of data. Note that if this is the first query — `offset = 0` — there will be no prev link. If `hasMore` is false, there will be no next link.

If one wanted the items included in the initial item master, then one would need to create a new data service with a URI template of simply “initial”. The get handler source would look like the following:

```
select ... from RDS_WV_ITEM_MASTER where csn_nbr is null
```

An interesting variation would be to also create a POST handler to copy the initial item master into object storage. This approach requires an object storage credential. The creation of a credential is something done infrequently. Bear in mind that credentials do expire and at some point credentials need to be refreshed. Once the credential is created, one implements the data export script. The last part, as implied above, entails creating a RESTful service to initiate the data export. In this case, a POST handler will be added to the “initial” URI template.

The construction of credentials in ADW is described in the [CREATE_CREDENTIAL Procedure](#) section of the [DBMS_CLOUD Subprograms and REST APIs](#) section. Note that the form of the CREATE_CREDENTIAL Procedure one uses is:

```
DBMS_CLOUD.CREATE_CREDENTIAL (  
  credential_name IN VARCHAR2,  
  user_ocid   IN VARCHAR2,  
  tenancy_ocid IN VARCHAR2,  
  private_key IN VARCHAR2,  
  fingerprint IN VARCHAR2);
```

One composes the create_credential script in either the [SQL Scripts](#) or the [SQL Commands](#) tool in the [SQL Workshop](#). A sample create credential script is shown below.

```
BEGIN  
  DBMS_CLOUD.CREATE_CREDENTIAL (  
    credential_name =>'OCI_KEY_CRED',  
    user_ocid=>'ocid1.user.oc1...zdyfhw33ozkwoontjceel7fok5nq3bf2vwetkqpsoa',  
    tenancy_ocid=>'ocid1.tenancy.oc1...gnemmoy5r7xvoypicjqgge32ewnrxcy2a',  
    private_key=>'MIEogIBAACKAQEAtUnxbmrekgwVac6Fd....pESQPD8NM//JEBg=',  
    fingerprint=>'f2:db:f9:18:a4:aa:fc:94:f4:f6:6c:39:96:16:aa:27');  
END;
```

Refer to [Required Keys and OCIDs](#) for details on obtaining credential information. The easiest way to obtain the needed credentials is by navigating 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). Next tap the *API Keys* link in the Resources section on the lower left of the screen. 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. There should be no new lines in the key. These final instructions will make more sense once one goes through the Add API Key process.

The last step is the actual export itself. In order to copy data from ADW to object storage, one uses [DBMS_CLOUD.EXPORT_DATA](#), e.g.:

```
DBMS_CLOUD.EXPORT_DATA (  
  file_uri_list IN CLOB,  
  format       IN CLOB,  
  credential_name IN VARCHAR2 DEFAULT NULL,  
  query       IN CLOB);
```

A sample export script is shown below. Note that a JSON response is composed using [json_object](#) and that response is returned using `htp.print`. To get started with JSON data see [JSON Data and Oracle Database](#). See [HTP](#) for more information on hypertext procedures.

```

declare
response varchar2(4000);
begin
  dbms_cloud.export_data(
    credential_name => 'OCI_KEY_CRED',
    file_uri_list=>",
    query => 'select ... from RDS_WV_ITEM_MASTER where csn_nbr is null',
    format => json_object('type' value 'json', 'compression' value 'gzip'));
select json_object('status' VALUE 'success') into response from dual;
http.print(response);
END;

```

The URI format in `file_uri_list` is the Native URI format, see [DBMS_CLOUD Package File URI Formats](#).

The name of the exported object would have the following form:

```
transactions1_<part>_<timestamp>.json.gz
```

Since this export is not multi-part, `part` is equal to "1." The next step is to consume the export. How this step is accomplished depends on customer requirements. One could use a shell oriented approach with Oracle OCI Command Line Interface, see also [Object Storage Service](#). One could also use a [Java](#) or [Python](#) API.

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.

An Asynchronous Approach

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. The initial item master export described above, however, is both relatively small and infrequent enough (probably once) that it does not warrant addressing it with an asynchronous approach.

One uses the DBMS_SCHEDULER.create_job procedure to create a job that 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 then 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.

Next Steps

The first step is to familiarize oneself with the above concepts. Start with a simple hello world service. First with a GET handler that is invoked from a browser that handles authentication. Next move on to curl or postman where one has generate access tokens. Lastly, build some simple queries. If a data export is anticipated, begin with a synchronous approach before attempting the more complex asynchronous approach.

Monitoring Resource Consumption in RDS

Although ADW is self-tuning, it cannot ensure that one's business priorities and resource consumption are well aligned. For example, the resource consumption of a new, as yet to be tuned, report may adversely impact higher priority tasks. By monitoring the consumption and performance of RDS, one is able to pinpoint which tasks could benefit from additional attention. In some cases, however, even well tuned tasks are long running and resource intensive. If these tasks are of lower priority, the user would like to run them at a lower priority. Ultimately, monitoring allows the user to both effectively employ compute services and determine if resource consumption matches business priorities. Control, on the other hand, gives the user a means to

align resource consumption with priorities. Exerting control will be the subject of another chapter whereas this chapter will focus on monitoring using AWR reports.

Monitoring

AWR reports are used to monitor activity in ADW. This section will discuss how to obtain AWR reports, but it will not discuss how to interpret those reports. Given that each customer's monitoring needs differ, there is no ready to use AWR report access built into RDS. In other words, an AWR report is obtained via a customer implemented data service.

There are two steps to creating an AWR report, obtaining snap ids and generating the report using the appropriate pair of snap ids. Sample code for obtaining snap ids is shown in Listing 1. The code is used as the source for a ORDS GET handler. It illustrates the use of two bind variables. The first is part of the URI template, the `begin_interval_time`. The second optional query string parameter is the end interval time. The times are given as dates for simplicity, but snap ids are based on timestamps. If the query parameter is not given, the value is null.

Note that a procedure, `HTPPRN`, is used to output the response. `HTP.print`, which is ultimately used to output the response, only handles `varchar2` args. `Varchar2` strings have a maximum size of 4000 characters, which is often insufficient. Hence, the output of the query is put into a `CLOB`. The `CLOB` is then output using `HTPPRN`, which is shown in Listing 2.

The second step is generating the report. I hard code the snap ids for simplicity. Using the code in Listing 3 as the source of a GET handler and the URL of the data service, a browser will render the report. It would not be difficult to add additional bind variables to allow one to create a narrower snap id interval and combine the snap id query and the report generation.

LISTING 1: OBTAINING SNAP IDS

```
DECLARE
    from_begin_interval_time date := to_date(:from, 'YY-MM-DD');
    to_end_interval_time date := null;
    db_id NUMBER;
    inst_id NUMBER;
    response clob;
BEGIN
    dbms_output.enable(1000000);
    if :to is not null then to_end_interval_time := to_date(:to, 'YY-MM-DD')
+ 1;
    end if;
    SELECT dbid INTO db_id FROM v$database;
    SELECT instance_number INTO inst_id FROM v$instance;

    SELECT json_arrayagg(
        json_object(
            'snap_id' value snap_id,
            'begin_interval_time' value begin_interval_time,
            'end_interval_time' value end_interval_time
            returning clob format json)
        returning clob) into response
    FROM dba_hist_snapshot
    WHERE dbid = db_id
    AND instance_number = inst_id
    and begin_interval_time >= from_begin_interval_time
    and (to_end_interval_time is null or
```

```

                to_end_interval_time >= end_interval_time)
            ORDER BY snap_id DESC;
    HTPPRN(response);
END;
```

LISTING 2: THE HTPPRN PROCEDURE

```

create or replace PROCEDURE HTPPRN(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.PRNV(V_TEMP);
        V_OFFSET := V_OFFSET + LENGTH(V_TEMP);
    END LOOP;

END;
```

LISTING 3: GENERATING THE AWR REPORT

```

DECLARE
    db_id NUMBER;
    inst_id NUMBER;
    start_id NUMBER;
    end_id NUMBER;
    response clob := null;
BEGIN
    dbms_output.enable(1000000);
    SELECT dbid INTO db_id FROM v$database;
    SELECT instance_number INTO inst_id FROM v$instance;
    start_id := 12133;
    end_id := 12134;

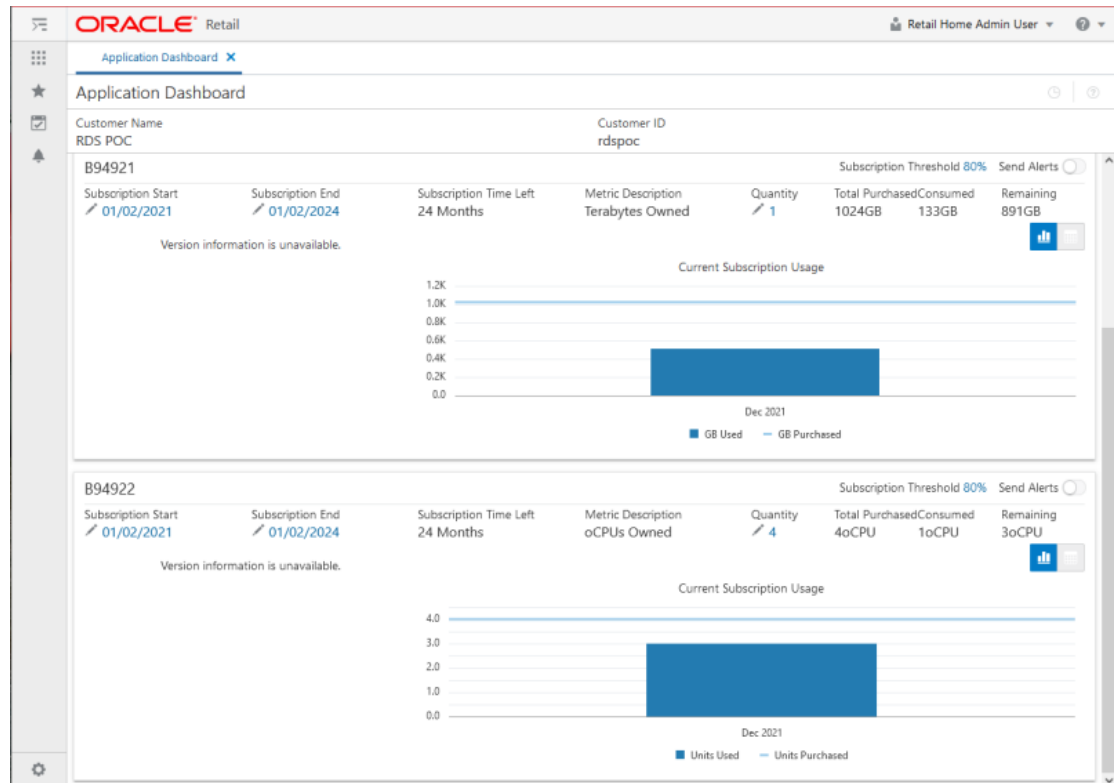
    FOR v_awr IN
        (SELECT output FROM
TABLE(DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_HTML(db_id,inst_id,start_id,e
nd_id)))
    LOOP
        response := response || v_awr.output;
    END LOOP;
    HTPPRN(response);
END;
```

5

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 also 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 5-1 Retail Home Application Dashboard



6

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.

7

Notes

This section provides additional resources when implementing RDS.

APEX

For more information around building performant APEX applications, refer to the [Managing Application Performance](#) section of the *APEX App Builder User's Guide*.

For full details on developing APEX applications, refer to the [APEX documentation](#).

Visual Builder Studio

For full details on developing Visual Builder applications, refer to the [Visual Builder Studio documentation](#).

APEX and Autonomous Databases

Because RDS is built using Oracle Autonomous Data Warehouse (ADW), there are limitations with functionality provided by Oracle Application Express. These limitations are documented at <https://docs.oracle.com/en/cloud/paas/autonomous-database/adbsa/apex-restrictions.html>