ORACLE

Extreme Oracle Database Connection Scalability with Database Resident Connection Pooling (DRCP)

Optimizing Oracle Database resource usage for applications and mid-tier services

October 2023, Version 2.1

Disclaimer

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without the prior written consent of Oracle. This document is not part of your license agreement, nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

Table of contents

ORACLE

List of images

List of tables

Introduction

Database Resident Connection Pooling (DRCP) is an Oracle Database feature developed for environments requiring multiple connections with optimal database resource usage. DRCP is typically suitable for web application scenarios where the application obtains a database connection, works on it for a relatively short time, and then releases the connection. DRCP provides a pool of "dedicated" server processes (known as ***pooled servers***) to the database that can be shared by multiple applications running on the same or several hosts. These pooled servers handle the database connections/sessions with the client applications. A connection broker process controls the pooled servers at the database instance level. As DRCP is a configurable feature chosen at application runtime, both traditional and DRCP-based connection architectures can be used concurrently by client applications.

DRCP boosts the scalability of the database and the application tier because connections to the database are held at a minimal cost. Database memory is used only by the pooled servers.

In Oracle Database's traditional dedicated connection model, each process creates and destroys database servers when connections are opened and closed. Applications with idle dedicated connections will hold on to database resources, such as the server process, memory storage, etc.

DRCP HELPS GOL TICK

"GOL selected Oracle Cloud Infrastructure based on its security, cost and superior performance compared to other vendors. If an application experiences a rise in simultaneous connections, GOL's IT uses the Database Resident Connection Pool from the Autonomous Cloud Database"

GOL Linhas Aéreas, Brazil
Leading Brazilian Airline
Oracle Customer Success Story

The DRCP implementation creates a pool of server processes on the database host that can be shared across multiple applications. The DRCP pool substantially decreases memory consumption on the server thanks to the multiplexing of client connections over a reduced number of database server processes. This removes the overheads of creating and destroying database servers and boosts the scalability of application deployments involving Oracle Database. As explained later in this document, applications with idle connections in DRCP do not consume database resources.

DRCP is available with all the editions of Oracle Database from version 11g and is usable on-premises and in Oracle Cloud. DRCP can be used by any applications running JDBC, ODP.NET, and Oracle Call Interface (OCI[1]) libraries to connect to Oracle Database. Applications using the Oracle Database drivers for Python, Node.js, PHP, Ruby, and Go also support DRCP.

This document covers the architecture, configuration settings, commands, benefits, system views, and examples to get you started and running with DRCP.

Please check the latest Oracle Database Administrator Guide: DRCP section for a more detailed overview of DRCP.

Overview of DRCP architecture

DRCP enables applications to scale up to tens of thousands of simultaneous database connections. The DRCP architecture plays a vital role in achieving this scalability.

DRCP uses pooled server processes, which are essentially the combination of dedicated server processes and database sessions. This model avoids the overhead of allocating a dedicated server for every client connection that only needs the server for a brief amount of time.

Client applications (hereon referred to as *clients* in this section) requesting connections from DRCP communicate with an Oracle background process known as a ***connection broker***. The connection broker multiplexes the pooled server processes among the inbound connection requests from the clients.

[1] Oracle's native C-based library for connecting to Oracle Database

Initially, the connection broker authenticates the connection requests from the client by using a reserved set of DRCP processes called ***authentication servers***. Usually, around 5% of the current pooled servers are reserved for authentication.

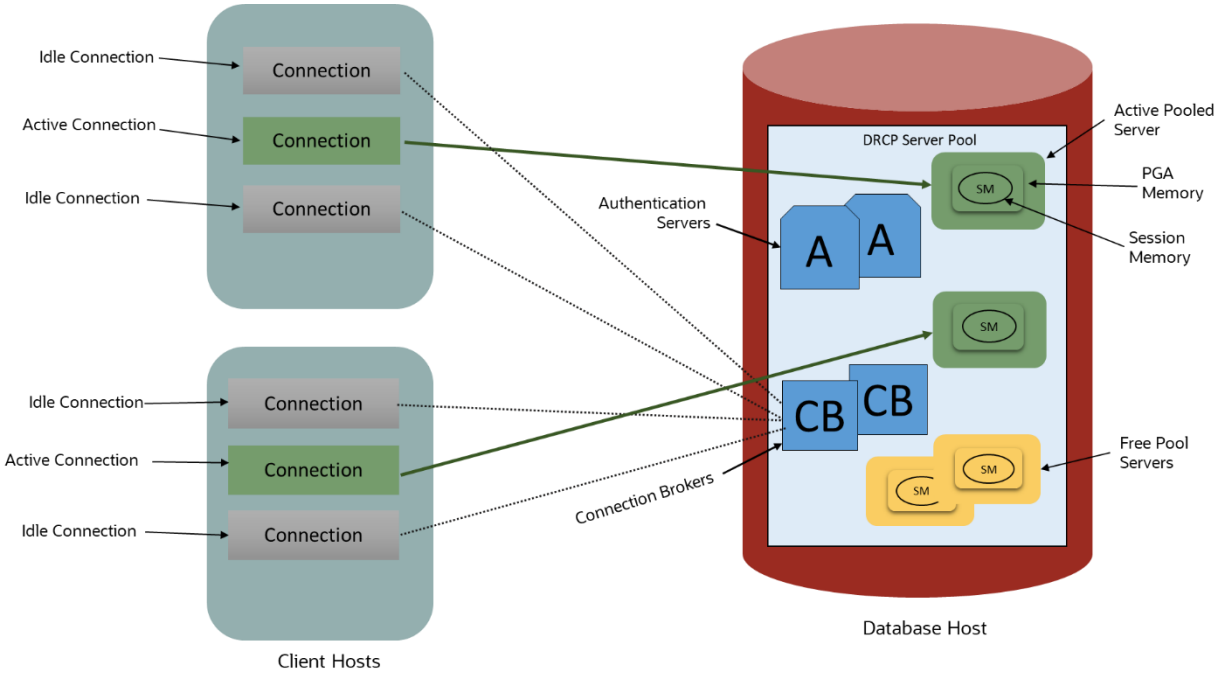

Image 1: DRCP Architecture between Client and Database hosts

Whenever the client process requires database access, the connection broker selects a pooled server process from the free pool and passes it to the client. The client will now be directly connected to the server process (dubbed the ***'active' pooled server***) until the database activity is completed.

After the server completes the database activity, the client application must release the active pooled server process back to the DRCP pool and re-establishes its link to the connection broker. The link to the connection broker remains open until the client process stops running or another client request comes in.

Also, in the case of DRCP, the session memory is stored in the Program Global Area (PGA) of the database, which is physically private to a process (the database connection here). This implementation enables the client to re-establish a connection quickly when any database activity is required.

DRCP Quickstart

The pool can be started, configured, and stopped by running the appropriate procedures in the PL/SQL DRCP package *DBMS_CONNECTION_POOL*. Only users with SYSDBA privileges (i.e., *SYS* user) or users granted EXECUTE access for the PL/SQL DRCP package by the SYS user can run it.

In Oracle Cloud Autonomous Database, DRCP is started by default. Otherwise, log in to Oracle Database as the user with the required privileges and run the *dbms_connection_pool.start_pool()* PL/SQL procedure.

Once the procedure runs successfully, any client application can access Oracle Database through DRCP using the Easy Connect string syntax with '*:pooled*' string or setting `(SERVER=POOLED)` in the TNS connect string. Applications that do not change the connect string will continue to use traditional dedicated server processes.

More details and examples of configuring DRCP and other applications using DRCP are available in the later sections of this technical brief.

How DRCP works

With DRCP, the database listener initially hands the new connection requests from a client to the DRCP connection broker (CB). These connection requests must initially be authenticated before running any database transactions on the connection.

The broker gets the authentication done using one of the pool's reserved authentication servers. Logon triggers fire once per authentication and once per user session. Logoff triggers fire on every logoff and session release. Once authenticated, the broker keeps the client connection persistent till the client closes the connection. Further pooled server requests and releases happen on this persistent and authenticated connection and are coordinated by the broker.

On a session request for a database activity, the connection broker assigns a server from the free pool of server processes for the client and hands over the authenticated client connection to this server. The client carries all its database interactions on this assigned server process, referred to as the *'busy'* server. After the client closes the session explicitly through an API call or when the client application ends, the busy server is released back to the free server pool with the session. Then, the client restores its connection to the connection broker, if the client application is still running. The broker holds this connection until a subsequent client connection request comes in or the connection is terminated by the client application or when the client application ends.

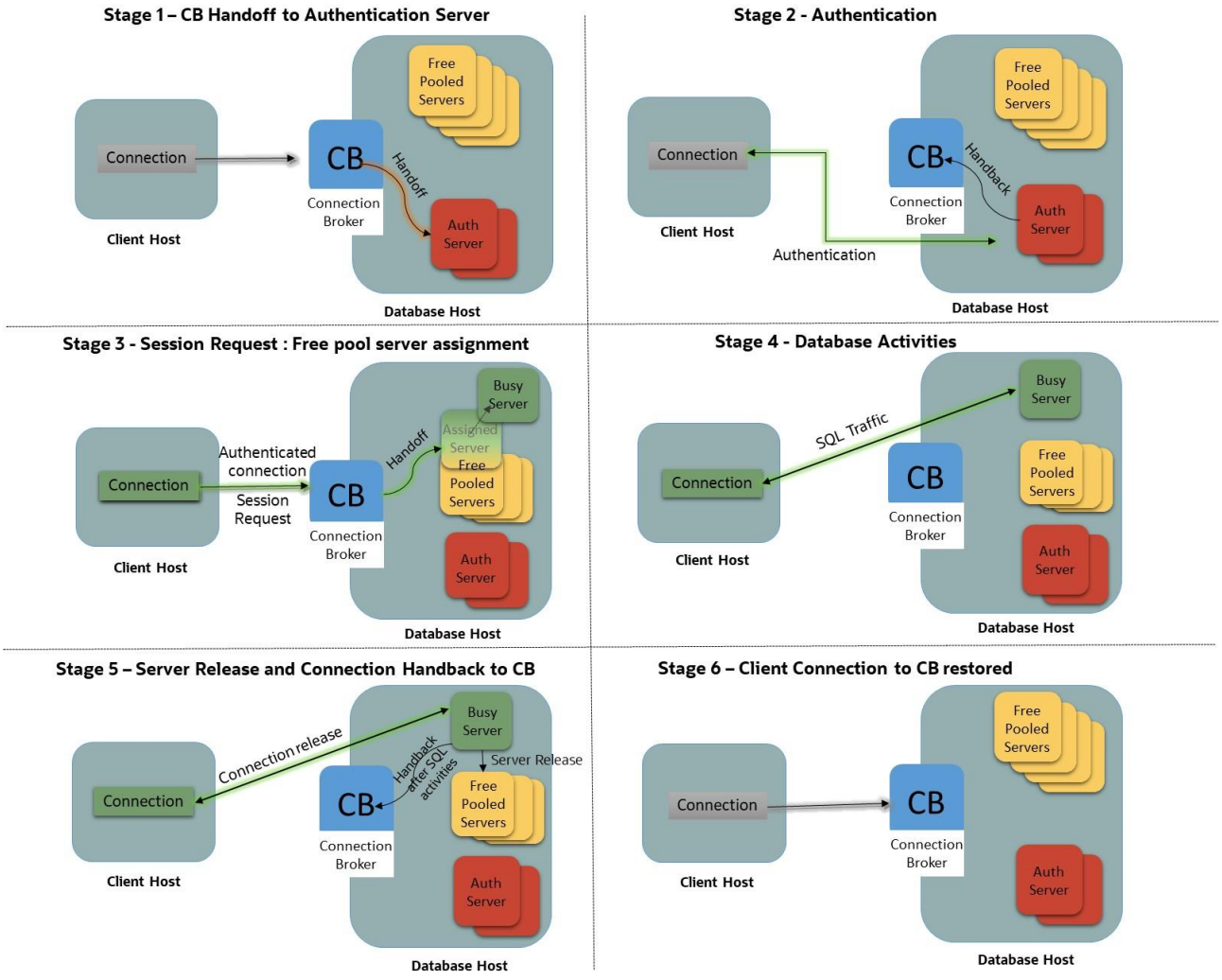

Image 2: Stages of DRCP Operation

The pool size and the number of connection brokers and authentication servers are configurable. There is always at least one connection broker per database instance when DRCP is enabled.

When to use DRCP

DRCP is generally recommended for

- Architectures with application servers (such as PHP) that cannot execute middle-tier connection pools
- Large-scale web deployments with several web servers, micro-services, or middle-tiers that require database access and client-side pools
- Web architectures that need to support high client connection traffic with minimum memory usage on the database host
- Applications where connections are held for a short amount of time
- Applications that mostly use the same database credentials for all connections
- Applications with identical connection or session settings, e.g., date formats and PL/SQL package state

These use cases generally have multi-process applications running on multiple hosts with a large number of connections to the database kept persistent, but not wanting to consume the database server memory when the connections are not active. The database can scale to tens of thousands of simultaneous connections with DRCP.

For example, a middle-tier connection pool with a pool size of 200 will have 200 connections to the database. The database, in turn, will have 200 server processes associated with these connections. Suppose there are 30 similar middle-tier applications, then the database will have 200 * 30 = 6000 corresponding server processes running in dedicated server mode. Let us assume that only 5% of these connections, and in turn, server processes are in use at any given time. In this case, only 300 server processes are active, and 5,700 idle server processes are running as wasted or unused resources on the database side at any given time.

DRCP can solve this resource wastage problem by multiplexing the 6000 client connections over the 300 or more pooled server processes (depending on how long the connections are held), leading to optimal database resource usage and higher scalability.

DRCP is available when connecting over TCP/IP with user ID/password-based database authentication. It is not available using Oracle's bequeath connections.

It is recommended that DRCP is used along with client-side connection pooling for maximum efficiency and optimal database resource consumption. For the best DRCP performance, the application should explicitly specify a connection class, as shown later in this document.

In addition, please take care of these considerations when using DRCP.

Comparing DRCP with other database server process models

Besides the pooled servers of DRCP, Oracle applications can use two other database server process models to access data: dedicated and shared servers.

The following table shows the differences between Dedicated, Shared, and Pooled servers.

DEDICATED SERVERS	SHARED SERVERS	POOLED SERVERS
When the connection is created, a network connection to a dedicated server process and associated session are created	A network connection to the dispatcher process is established when the connection is created. A session is created in the SGA[2].	A network connection to the broker is established and authenticated when the connection is created.

[2] System Global Area – a group of shared memory structures for the whole Oracle Database instance

The dedicated server handles activity on a connection	Each action[3] on a connection goes through the dispatcher, which hands the work to a shared server	When the application requests a session, the broker wakes up and hands the network connection to a pooled server with a session. The pooled server then handles subsequent database requests or activities directly, just like a dedicated server
A program executing but with an idle client connection will maintain a link to a server process and hold session resources	A program executing but with an idle client connection will hold session resources but not maintain a link to a server process	A program executing but with an idle client connection that has already released the database session will maintain a link to the connection broker
Closing a connection causes the session to be freed and the server process to be terminated	Closing a client connection causes the session to be freed, and the client disconnects from the dispatcher	Closing a client connection causes the pooled server with the session to be released to the pool. A network connection to the connection broker is retained
Memory usage is proportional to the number of server processes and sessions. There is one server and one session for each connection.	Memory usage is proportional to the sum of the shared servers and sessions. There is one session for each client connection.	Memory usage is proportional to the number of pooled server processes and their sessions. There is one session per pooled server.

Table 1 – Difference between Dedicated, Shared, and Pooled servers

Note: In the case of DRCP, the client releases the connection to the client connection pool when idle.

Example of host memory usage between Dedicated, Shared, and Pooled servers

Consider an application where the memory required for each session is 400 KB. On a 64-bit operating system, the memory required for each server process could be 8 MB, while DRCP could use 35 KB per connection (mainly in the connection broker). Suppose the number of pooled servers is configured at 100. Also, assume the number of shared servers is set to 100, and the deployed application creates 5000 application connections.

The memory used by each type of server is estimated in the table below.

	DEDICATED SERVERS	SHARED SERVERS	POOLED SERVERS
Database Server Memory	5000 x 8 MB	100 x 8 MB	100 x 8 MB
Session Memory	5000 x 400 KB	5000 x 400 KB Note: For Shared Servers, session	100 x 400 KB

[3] An action is essentially an SQLNet roundtrip, which may involve the execution of one or a bunch of SQL or PL/SQL statements, transaction commit, etc.

		memory is allocated from the SGA	
DRCP Connection Broker Overhead	0 (Not applicable)	0 (Not applicable)	5000 x 35 KB
Total Memory	**42 GB**	**2.8 GB**	**1 GB**

Table 2 – Sample DB memory usage for Dedicated, Shared, and Pooled servers

As can be seen, DRCP pooled servers provide the best database host memory usage among the three options.

Per-PDB DRCP

The multitenant option introduced in Oracle 12c brought in the Container Database (CDB) and Pluggable Database (PDB) model. In this model, only one DRCP pool was available for all PDBs, and management was only done at the CDB level by the ROOT user with SYSDBA privileges (**CDB DRCP**) until recently.

Starting from Oracle Database version 21c, DRCP can be either at the CDB (**CDB DRCP**) or PDB level (**per-PDB DRCP**). In per-PDB DRCP mode, a PDB Admin user can configure, manage and monitor the DRCP pool owned by that PDB. The brokers are still owned by ROOT and shared by all the per-PDB DRCP pools.

Please check the Oracle DRCP Documentation for more details on per-PDB DRCP.

Enabling per-PDB DRCP

By default, the DRCP is at the CDB level. In CDB DRCP mode, a single DRCP pool running in the CDB is shared across all the PDBs. In this mode, the database initialization parameter `ENABLE_PER_PDB_DRCP` will be `'FALSE'`.

`ENABLE_PER_PDB_DRCP` can be set to `'TRUE'` to enable per-PDB DRCP.

For the PDB admin user (say *PDB1ADMIN*) to access the `DBMS_CONNECTION_POOL` package and query the DRCP statistics, the Root user (*SYS*) has to grant the following permissions.

```
GRANT CREATE SESSION, CREATE SYNONYM TO PDB1ADMIN;
GRANT EXECUTE ON DBMS_CONNECTION_POOL TO PDB1ADMIN;
GRANT SELECT ON V_$CPOOL_STATS TO PDB1ADMIN;
GRANT SELECT ON V_$CPOOL_CC_STATS TO PDB1ADMIN;
GRANT SELECT ON V_$CPOOL_CONN_INFO TO PDB1ADMIN;
GRANT SELECT ON V_$CPOOL_CC_INFO TO PDB1ADMIN;
GRANT SELECT ON V_$AUTHPOOL_STATS TO PDB1ADMIN;
```

To make management and monitoring of the DRCP pool easier, the PDB Admin user (*PDB1ADMIN* in this case) can create the following synonyms.

```
CREATE SYNONYM DBMS_CONNECTION_POOL FOR SYS.DBMS_CONNECTION_POOL;
CREATE SYNONYM V$CPOOL_STATS FOR SYS.V_$CPOOL_STATS;
CREATE SYNONYM V$CPOOL_CC_STATS FOR SYS.V_$CPOOL_CC_STATS;
CREATE SYNONYM V$CPOOL_CONN_INFO FOR SYS.V_$CPOOL_CONN_INFO;
CREATE SYNONYM V$CPOOL_CC_INFO FOR SYS.V_$CPOOL_CC_INFO;
CREATE SYNONYM V$AUTHPOOL_STATS FOR SYS.V_$AUTHPOOL_STATS;
```

Once this is done, pool management is allowed only at the PDB level by the respective PDB Admins.

CDB DRCP vs. per-PDB DRCP

CDB DRCP

In CDB DRCP, the ROOT user manages the DRCP Pool in the CDB. All the PDBs share this single DRCP pool.

ROOT MANAGING THE POOL	PDB MANAGING THE POOL	ROOT VIEWING THE STATS	PDB VIEWING THE STATS
start_pool() *stop_pool()* *configure_pool()* *alter_param()* *restore_defaults()* *num_cbrok* and *maxconn_cbrok* not modifiable using *alter_param()* if DB parameter *connection_brokers* has been set through *init.ora* or *ALTER SYSTEM*	none of the procedures on the `dbms_connection_pool` package are allowed at the PDB level	The root can query below gv$tables and the corresponding v$ tables. *gv$cpool_stats* *gv$cpool_cc_stats* *gv$cpool_conn_info* *gv$authpool_stats* *gv$cpool_cc_info*	The *SYS* user connected to a PDB can view stats only from *gv$cpool_conn_info* and *gv$authpool_stats* and corresponding v$ tables.

Table 3 – CDB DRCP behavior

ROOT MANAGING THE POOL	PDB MANAGING THE POOL	ROOT VIEWING THE STATS	PDB VIEWING THE STATS
The procedures on the `dbms_connection_pool` package are not allowed on the root. The root can alter *num_cbrok* and *maxconn_cbrok* values using the DB parameter *connection_brokers*	*start_pool()* *stop_pool()* *configure_pool()* *alter_param()* *restore_defaults()* But *num_cbrok* and *maxconn_cbrok* are not modifiable using *alter_param()*. PDB cannot alter the DB parameter *connection_brokers*	The root can query below gv$tables and the corresponding v$ tables connected to the CDB or PDB in standalone mode. *gv$cpool_stats* *gv$cpool_cc_stats* *gv$cpool_conn_info* *gv$authpool_stats* *gv$cpool_cc_info* The result will contain the DRCP information about all the PDBs	PDB can query below gv$tables and the corresponding v$ tables. *gv$cpool_stats* *gv$cpool_cc_stats* *gv$cpool_conn_info* *gv$authpool_stats* *gv$cpool_cc_info*

Table 4 – per-PDB DRCP behavior

Configuring Database Resident Connection Pooling

This section describes how to configure and enable DRCP on both the server side and the client side:

- Enabling and Configuring DRCP on the Server side
- Application Deployment for DRCP

Note: *DRCP is already started in Oracle Cloud Autonomous Database by default.*

The settings for DRCP can be configured using the DRCP configuration options and DRCP database initialization parameters, which are explained in detail later in this section.

Enabling and Configuring DRCP on the Server side

From Oracle Database 21c onwards, the first choice that the database administrator has to make in DRCP configuration is between per-PDB DRCP or CDB DRCP. So only a database administrator (DBA) logging on with SYSDBA privileges or a PDB Administrator with EXECUTE privileges on the `DBMS_CONNECTION_POOL` package (granted by the `SYS` user) can start and stop a pool. In this section, we will use SQL*Plus to configure DRCP on the database.

For CDB DRCP, the database user with SYSDBA privileges (usually `SYS` user) can use the following commands of the `DBMS_CONNECTION_POOL` package to manage DRCP.

1. Start pool: The `start_pool` procedure starts DRCP. When DRCP starts, Oracle Database names the connection pool created as `SYS_DEFAULT_CONNECTION_POOL` automatically.

```
sqlplus /nolog
SQL> connect / as sysdba
SQL> execute dbms_connection_pool.start_pool()
```

Once started, the pool automatically restarts when the instance restarts (shutdown followed by startup) unless explicitly stopped with the *stop_pool()* procedure.

2. Stop pool: The `stop_pool` procedure stops DRCP. The DRCP pool (`SYS_DEFAULT_CONNECTION_POOL`) will be stopped if it is running.

```
SQL> execute dbms_connection_pool.stop_pool()
```

3. Configure pool: The `configure_pool` procedure configures the pool (`SYS_DEFAULT_CONNECTION_POOL`) with additional options. For example:

```
SQL> execute dbms_connection_pool.configure_pool(
        minsize => 4,
        maxsize => 40,
        incrsize => 2,
        session_cached_cursors => 20,
        inactivity_timeout => 300,
        max_think_time => 600,
        max_use_session => 500000,
        max_lifetime_session => 86400)
```

This procedure is used when all the connection pool parameters must be modified.

4. Alter parameters: Alternatively, the method `dbms_connection_pool.alter_param()` can be used to set a single parameter and does not affect other parameters in the pool (`SYS_DEFAULT_CONNECTION_POOL`):

```
SQL> execute dbms_connection_pool.alter_param(
        param_name => 'MAX_THINK_TIME',
        param_value => '1200')
```

The difference between `alter_param` and `configure_pool` options is that `alter_param` only affects a single parameter, whereas `configure_pool` requires all the parameter values to be specified when it is called.

5. Restore defaults: The `restore_defaults()` procedure resets the default configuration values of the DRCP pool (`SYS_DEFAULT_CONNECTION_POOL`).

```
SQL> exec dbms_connection_pool.restore_defaults()
```

If DRCP is at the PDB level (per-PDB DRCP), then the PDB administrator (with privileges enabled) will have to execute the above commands for the corresponding PDBs.

DRCP Configuration Settings

The table below shows the list of *DRCP configuration options* that the *configure_pool* and *alter_param* procedures can use:

DRCP OPTION	DESCRIPTION
pool_name	The name of the pool to be configured. Currently, the only name supported is the default value SYS_DEFAULT_CONNECTION_POOL.
minsize	Sets the minimum number of pooled servers in the pool. The default value is 4 when DRCP is configured at the CDB level and 0 when per-PDB DRCP is enabled.
maxsize	Sets the maximum number of pooled servers allowed in the pool. If this limit is reached and all the pooled servers are busy, then connection requests wait until a server becomes free. The default is 40.
incrsize	Sets the increment number by which pooled servers are increased when servers are unavailable for connections, and the pool is not yet at its maximum size. The default is 2.
session_cached_cursors	Turns on the database parameter SESSION_CACHED_CURSORS for all the pool connections. Typically, this number is set to the size of the working set of frequently used statements. The cache uses cursor resources on the server. The default is 20. An *init.ora* parameter is also available for setting the value for the entire database instance. The pool option allows a DRCP-based application to override the instance setting.
inactivity_timeout	Time to live, in seconds, for a free server in the pool. After this time, the free server process is terminated. This parameter helps to shrink the pool when it is not used to its maximum capacity. This parameter will not apply if the pool size is already at *minsize*. The default is 300 seconds.

max_think_time	The maximum time of inactivity, in seconds, allowed after the client is connected to a pooled server. If the application code or script does not issue a database call for this amount of time, the pooled server may be returned to the pool for reuse, and the client connection is terminated. The application will get an `ORA-3113` or `ORA-3115` error if it tries to use the connection later. The default is 120 seconds.
max_txn_think_time	The maximum time of inactivity, in seconds, for a client after it obtains a pooled server from the pool with an open transaction. If the client application does not make a database call within the time frame provided by *max_txn_think_time* after getting the pooled server from the pool, the pooled server is released, and the client connection is terminated. The default value for this parameter is the *max_think_time* parameter value. Applications can set this parameter value higher than the *max_think_time* value to provide more time for the connections with open transactions.
max_use_session	The maximum number of times a server can be taken and released to the pool before it is flagged for restarting. The default is 500000.
max_lifetime_session	Time to live, in seconds, for a pooled server before it is restarted. The default is 86400 seconds.
num_cbrok	The number of connection brokers created to handle connection requests. This parameter can be set with *alter_param()*. The default is 1. Multiple connection brokers can help distribute the load of incoming client connection requests from many client applications. When per-PDB DRCP is in effect, the PDB admin cannot modify the value of this parameter. Only the root DBA can modify the `CONNECTION_BROKERS` database initialization parameter, as illustrated here.
maxconn_cbrok	Sets the maximum number of connections that each connection broker can handle. The operating system's per-process file descriptor limit must be set sufficiently high to support the number of connections specified. This parameter can only be set with *alter_param()*. The default is 40000. When per-PDB DRCP is in effect, the PDB admin cannot modify the value of this parameter. Only the root DBA can modify the `CONNECTION_BROKERS` database initialization parameter, as illustrated here.

Table 5 – DRCP Configuration Options

You can also set database initialization parameters for additional configuration and optimization in DRCP:

DRCP PARAMETER	DESCRIPTION
`ENABLE_PER_PDB_DRCP`	Available from Oracle Database 21c onwards. This parameter specifies if DRCP is configured at the CDB level or per PDB. The default value is FALSE, which will configure DRCP at the CDB level. When it is set to TRUE, one

	isolated connection pool is created for each PDB, and no connection pool is created at the CDB level.
DRCP_DEDICATED_OPT	Available from Oracle Database 19.11 onwards. This parameter configures the use of dedicated optimization with DRCP. The default is YES in Oracle Database 19c and NO from Oracle Database 21c onwards. Dedicated optimization is enabled by setting this parameter to YES. Dedicated optimization makes DRCP operate like a dedicated server when the number of connections to the DRCP broker is less than the maximum size of the DRCP pool. Dedicated optimization allows the number of open pooled servers to grow to the maximum size, even when the connections are inactive. Depending on the value of the ENABLE_PER_PDB_DRCP parameter, this parameter can be modified by the CDB root user or the PDB admin user.
DRCP_CONNECTION_LIMIT	Available from Oracle Database 21c onwards. This parameter provides limits on the number of DRCP connections for a PDB. If a PDB has a session limit, the default is 10 * sessions. Otherwise, it is unlimited.
MAX_AUTH_SERVERS	Available from Oracle Database 19.10 onwards. This parameter specifies the maximum number of server processes in the DRCP authentication pool. This value must be greater than or equal to the MIN_AUTH_SERVERS parameter value. If the MIN_AUTH_SERVERS value is 0, this value must be at least 1. The default value is 25. Depending on the value of the ENABLE_PER_PDB_DRCP parameter, this parameter can be modified by the CDB root user or the PDB admin user.
MIN_AUTH_SERVERS	Available from Oracle Database 19.10 onwards. This parameter specifies the minimum number of server processes in the DRCP authentication pool. This value must be less than or equal to the values of both MAX_AUTH_SERVERS and PROCESSES parameters. Depending on the value of the ENABLE_PER_PDB_DRCP parameter, this parameter can be modified by the CDB root user or the PDB admin user.
CONNECTION_BROKERS	This parameter specifies the connection broker types, the number of connection brokers of each type, and the maximum number of connections per broker. When per-PDB DRCP is enabled, a PDB admin user cannot set this parameter in the PDB.

Table 6 – DRCP Database Initialization Parameters

The ALTER SYSTEM SQL command can be used to modify all of the above parameters except ENABLE_PER_PDB_DRCP. ENABLE_PER_PDB_DRCP can be set only through the database configuration file.

Configuring brokers for per-PDB DRCP

Since broker processes are shared among all the PDBs, the pool parameters *num_cbrok* and *maxconn_cbrok* values from DBA_CPOOL_INFO are ignored and cannot be modified by the PDB admin using dbms_connection_pool.alter_param(). These parameters can be set using the database parameter

CONNECTION_BROKERS and can only be dynamically altered in the ROOT container. By default, a single broker process will be started with a maximum limit of 40000 connections per broker.

Example:
```
CONNECTION_BROKERS='((TYPE=POOLED)(BROKERS=2)(CONNECTIONS=40000))'
```

The BROKERS option will set the number of connection brokers, and the CONNECTIONS option will set the maximum number of connections per broker.

Using DRCP with Oracle Real Application Clusters (RAC)

When DRCP is used with Oracle RAC[4], each database instance has its own connection broker and pool of servers. DRCP configuration in an Oracle RAC environment is applied to every database instance. Hence, each pool has an identical configuration. For example, all pools will start with *minsize* server processes. A single *dbms_connection_pool* command will alter the pool of each instance at the same time. However, the database initialization parameters in Table-6 (except the ENABLE_PER_PDB_DRCP and CONNECTION_BROKER parameters) can be set to different values in different instances.

Using DRCP with Oracle Cloud Autonomous Database (ADB)

DRCP is enabled in Oracle Cloud Autonomous Databases (ADBs) by default. Note that the client applications will have the option of using DRCP as specified in the following sub-section. Oracle ADBs do not allow users to start or stop DRCP.

Working with DRCP in client applications

Once DRCP is enabled on the database, applications can use DRCP by specifying '*: POOLED*' in the Easy Connect string (as in the example below) or (SERVER=POOLED) in the TNS connect string when they connect to the database.

DRCP with '*:POOLED*' in the Easy Connect String

```
oraclehost.company.com:1521/booksdb.company.com:POOLED
```

Enabling DRCP with SERVER=POOLED in the TNS Connect String

```
BOOKSDB = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=oraclehost.company.com)
(PORT=1521))(CONNECT_DATA = (SERVICE_NAME=booksdb.company.com)(SERVER=POOLED)))
```

Applications connecting to Oracle Database through DRCP should use connections for short database activities and then close them promptly after the database activities are completed.

Managing DRCP Connections with multiple applications or microservices

Note: In this section, the statements about the usage of applications with DRCP hold good for the usage of micro-services with DRCP as well.

[4] Real Application Clusters - a database option in which a single database is hosted by multiple instances on multiple nodes

DRCP guarantees that sessions in pooled servers used initially by one database user can only be reused by connections with that same user identifier. DRCP also further partitions the pool into logical groups or "connection classes". The pooled servers are also partitioned based on the service names. It is recommended that applications provide a connection class for the best performance when connecting to the database through DRCP.

DRCP also allows applications to set the session purity attribute to control the reusability of the pooled sessions.

The connection class and session purity settings help applications to unleash the full power of DRCP and provide the best performance and experience to the end users.

What is Connection Class?

The connection class defines a logical name for the type of connection an application wants to use and share across multiple application processes or other applications. The right set of connection classes partitions the connections effectively and prevents session information leakage.

The connections are not shared across users and service names. The connection class adds an extra level of sharing boundary that applications can maintain when they use the same user and service name. Applications that require different states in the sessions should use different user names and/or connection classes.

Let us suppose that the server initially belonged to a different connection class. In that case, the current session will be destroyed, and the server will migrate to the new class and get a newly created session. If no pooled servers are available, the connection request waits for one to become available. This behavior allows the database to continue without becoming overloaded.

If no free pooled servers match a request for a user ID in the specified connection class, and if the pool is already at its maximum size, then an idle server in the pool with a different class will be used, and a new session will be created for it.

For example, for the same username Blake, applications in a group called Sales may be willing to share pooled servers between themselves but not with an application group called CRM, as illustrated in the following diagram.

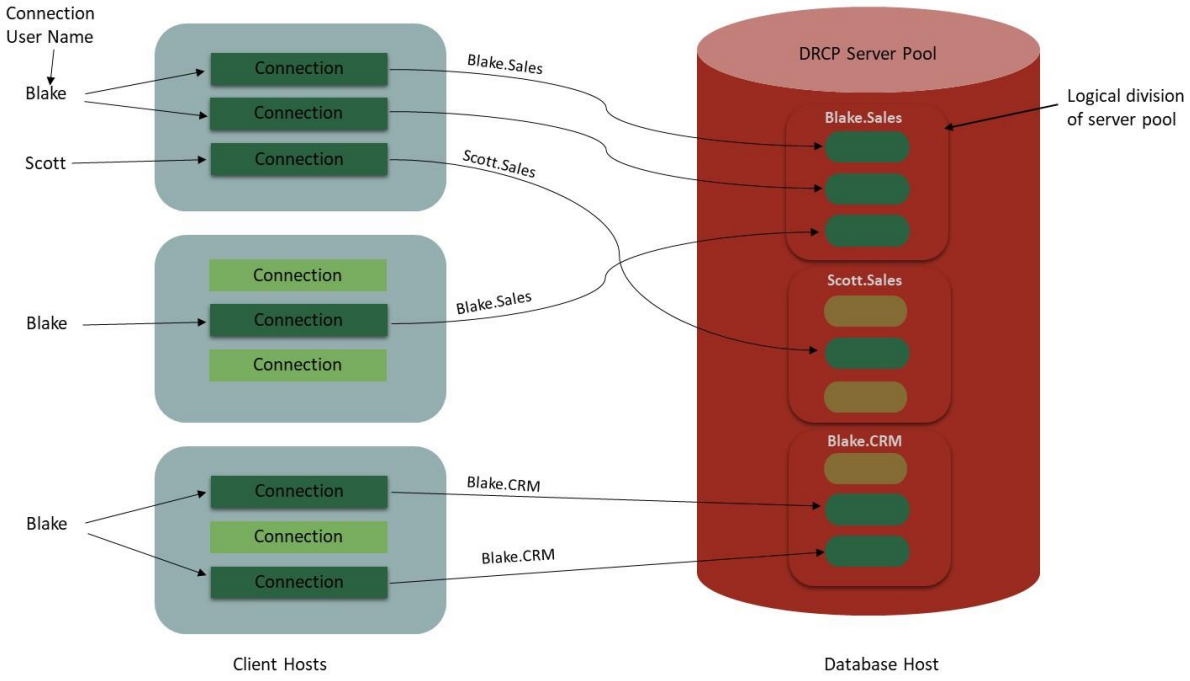

Image 3: DRCP Pool Sharing across applications

What is Session Purity?

The Session Purity attribute specifies if the application wants a "brand new" session (NEW) or if the application logic is set up to reuse a "pooled" session (SELF).

DRCP Session Purity and Connection Class defaults

DRCP ATTRIBUTE OR SETTING	DEFAULT VALUE FOR A CONNECTION FROM AN APPLICATION CONNECTION POOL	DEFAULT VALUE FOR A CONNECTION NOT FROM AN APPLICATION CONNECTION POOL
PURITY	SELF	NEW
CONNECTION CLASS	For applications using Oracle Call Interface (OCI) libraries, a randomly generated unique name for each client-side session pool is used as the default connection class for all connections in the session pool. Python-oracledb Thin mode uses SHARED as the default connection class name for its application pools. For JDBC Thin, the default is the name of the connection pool specified if UCP is set. If UCP is not set, the class gets a random name. In Managed and Core ODP.NET, the default is null.	SHARED

Table 7 – Session Purity and Connection Class Defaults

Session Purity and Connection Class in the connection string

When it is not possible to change the application setting in the application code, you can override the value set in the application by setting the parameters POOL_CONNECTION_CLASS and POOL_PURITY in the connect string.

The `POOL_CONNECTION_CLASS` and the `POOL_PURITY` attributes specified in a connect string will have the highest priority and thus, will override the default or application specified values (set through OCI *OCIAttrSet* or *OCISessionGet* calls or by the Python, JDBC, and ODP.NET thin drivers).

The valid values for POOL_PURITY are SELF and NEW. An appropriate error message will be thrown if any other value is specified. These values are not case-sensitive. (Note: When using SELF in connect string, session requests with NEW purity will not be dropped from the application pool even if the application passes the *OCI_SESSRLS_DROPSESS* mode in *OCISessionRelease()*.)

Value for `POOL_CONNECTION_CLASS` can be any string conforming to connection class semantics and is case sensitive.

These two parameters cannot have an empty value. Otherwise, an appropriate error message will be thrown. These two parameters are ignored if the SERVER is not POOLED.

With TNS Connect string:

```
ServerPool =
  (DESCRIPTION =
        (ADDRESS=(PROTOCOL=tcp)(HOST=oraclehost)(PORT=1521))
        (CONNECT_DATA=(SERVICE_NAME=<service name>)(SERVER=POOLED))
        (POOL_CONNECTION_CLASS=CCNAME)(POOL_PURITY=SELF))
```

With Easy Connect string:

```
conn[ect] [USER NAME]@'[HOST]:[PORT NO]/[SERVICE
NAME]:POOLED?POOL_PURITY=SELF&POOL_CONNECTION_CLASS=CCNAME'
```

In Easy Connect syntax, the *pool_connection_class* and *pool_purity* attributes can be used with Oracle Database 21c onwards. If the applications use Oracle Client libraries, then these attributes are supported in the Easy Connect syntax from Oracle Client version 12 or later.

For more information on the usage of these DRCP parameters, please check out the latest technical brief on Easy Connect syntax.

Monitoring DRCP

In-built data dictionary views and dynamic performance views are available in Oracle Database to monitor the performance of DRCP. Database administrators can check statistics such as the number of busy and free servers and the number of hits and misses in the pool against the total number of client requests.

The in-built views available in Oracle Database for looking at DRCP statistics are:

```
DBA_CPOOL_INFO
V$CPOOL_STATS
V$CPOOL_CC_STATS
V$CPOOL_CONN_INFO
V$CPOOL_CC_INFO
V$AUTHPOOL_STATS
```

In the following subsections, we will be using SQL*Plus to query the data dictionary views for DRCP.

DBA_CPOOL_INFO

The DBA_CPOOL_INFO view displays configuration information about the connection pool, such as the pool status, the maximum and the minimum number of connections, etc.

The following example checks if the pool has been started (*ACTIVE* status) and finds the maximum number of pooled servers allowed:

```
SQL> SELECT connection_pool, status, maxsize FROM dba_cpool_info;

CONNECTION_POOL                STATUS          MAXSIZE
```

```
---------------------------   ----------      ----------
SYS_DEFAULT_CONNECTION_POOL    ACTIVE            40
```

V$CPOOL_STATS View

The V$CPOOL_STATS view displays information about the DRCP statistics for a database instance. The V$CPOOL_STATS view can assess the efficiency of the connection pool settings.

The query in the following example shows an application using the pool effectively. The low number of misses indicates that servers and sessions were reused by the sharing applications and the purity to SELF. The wait count shows just over 10% of requests had to wait for a pooled server to become available:

```
SQL> SELECT num_requests, num_hits, num_misses, num_waits FROM v$cpool_stats;

NUM_REQUESTS    NUM_HITS        NUM_MISSES      NUM_WAITS
------------    ----------      ----------      ----------
10031           99990           40              1055
```

If the connection class is set (allowing pooled servers and sessions to be reused), then NUM_MISSES will be low. If the pool *maxsize* value is too small for the connection load, then NUM_WAITS will be high.

When CDB-level DRCP is enabled, this view returns data only when queried from a CDB root (SYS user) and returns 0 rows when queried from a PDB. In the case of per-PDB DRCP, this view will return data when queried from a CDB root (SYS user) and PDB (PDBADMIN user).

Note: SQL*Plus does not set PURITY by default and hence does not reuse DRCP sessions.

V$CPOOL_CC_STATS View

The view V$CPOOL_CC_STATS contains the connection class level statistics for the pool per instance:

```
SQL> SELECT cclass_name, num_requests, num_hits, num_misses
     FROM v$cpool_cc_stats;
CCLASS_NAME                      NUM_REQUESTS         NUM_HITS NUM_MISSES
---------------------------     -------------        -------- -----------
HR.MYCLASS                       100031               99993       38
```

When CDB-level DRCP is enabled, this view returns data only when queried from a CDB root (SYS user) and returns 0 rows when queried from a PDB. In the case of per-PDB DRCP, this view will return data when queried from both the CDB (Root user) and the PDB (PDBADMIN user).

V$CPOOL_CONN_INFO View

You can monitor the view V$CPOOL_CONN_INFO to identify misconfigured machines, for example, that do not have the connection class set correctly. This view displays the connection information of each connection to the connection broker. This sample query below maps the machine name to the class name:

```
SQL> SELECT cclass_name, machine FROM v$cpool_conn_info;
CCLASS_NAME                                              MACHINE
```

```
---------------------------------------          ------------

GK.OCI:SP:wshbIFDtb7rgQwMyuYvodA                 gklinux
```

In this example, you would examine applications on the Linux machine (*gklinux)* and make sure that *cclass* is set. More examples of usage for the `V$CPOOL_CONN_INFO` view can be found here.

V$CPOOL_CC_INFO View

V$CPOOL_CC_INFO holds information on the pool-to-connection class mapping for the DRCP pool of each database instance. The following sample query identifies all the connection classes in the database instance:

```
SQL> SELECT pool_name, cclass_name FROM v$cpool_cc_info;

POOL_NAME                               CCLASS_NAME                     CON_ID
--------------------------------------- ------------------------------- ------
SYS_DEFAULT_CONNECTION_POOL              HR.MYCLASS                        3
```

In this example, the user is *HR,* and the Connection Class is *MYCLASS*.

V$AUTHPOOL_STATS View

V$CPOOL_AUTHPOOL_STATS shows the statistics for the authentication servers of DRCP. This view is available from Oracle Database 21c. The following sample query looks at the authentication server statistics:

```
SQL> select num_srvs, num_busy, num_free, num_waiters from v$authpool_stats;

  NUM_SRVS   NUM_BUSY   NUM_FREE NUM_WAITERS
---------- ---------- ---------- -----------
         3          0          3           0
```

This example shows three authentication server processes that are free and ready to receive any connection authentication requests.

DRCP examples with different languages

To enable and use DRCP with applications, we have to:

1. Configure and enable DRCP in the database
2. Configure the application to use a DRCP connection
3. Deploy the application

If the below code snippets are executed without configuring the database for DRCP, the connections will not succeed, and an error will be returned to the application.

DRCP with Python

The latest Python interface for Oracle Database, *python-oracledb (*package name: *oracledb)*, supports DRCP.

Application Deployment for DRCP

To request the database to use a DRCP pooled server, you can use a specific connection string in *oracledb.create_pool()* or *oracledb.connect()* similar to one of the following syntaxes.

Using Oracle's Easy Connect syntax, the connection parameters would look like this:

```
import oracledb
connection = oracledb.connect(user="hr", password=userpwd,
                              dsn="dbhost.example.com/orcl:pooled”,
                              cclass="MYAPP")
```

Or if you connect using a Net Service Name named *customerdb*:

```
connection = oracledb.connect(user="hr", password=userpwd,
                              dsn="customerdb")
```

In this case, only the Oracle Network configuration file tnsnames.ora needs to be modified:

```
customerdb = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=dbhost.example.com)
             (PORT=1521))(CONNECT_DATA=(SERVICE_NAME=CUSTOMER)(SERVER=POOLED)))
```

You can also specify to use a DRCP pooled server by setting the *server_type* parameter when creating a standalone connection or a python-oracledb connection pool.

For example:

```
pool = oracledb.create_pool(user="hr", password=userpwd,
                            dsn="dbhost.example.com/orclpdb", min=2, max=5,
                            increment=1, server_type="pooled")
```

Setting Connection Class and Purity attributes

This user-chosen name provides some partitioning of DRCP session memory. So, reuse is limited to similar applications. It provides maximum pool sharing if multiple application processes are started.

To create an application connection pool requesting DRCP servers using a connection class name (*cclass* attribute) and get a connection:

```
pool = oracledb.create_pool(user="hr", password=userpwd,
                            dsn="dbhost.example.com/orclpdb:pooled",
                            min=2, max=5, increment=1,
                            cclass="MYAPP")
connection = pool.acquire()
```

The purity of all the connections in the pool will be set to SELF (*PURITY_SELF* value in python-oracledb) by default, which is also the recommended best practice.

The python-oracledb connection pool size does not need to match the DRCP pool size. The DRCP pool size determines the limit on overall execution parallelism.

Connection class names can also be passed to the *acquire()* function:

```
connection = pool.acquire(cclass="OTHERAPP")
```

To change the purity of the connection in the pool to *NEW*, set the purity attribute to `PURITY_NEW` when using the *acquire()* function.

```
connection = pool.acquire(cclass="MYAPP", purity=oracledb.PURITY_NEW)
```

You can use this connection object to run any database transactions.

```
with connection.cursor() as cursor:
    print("Performing query using DRCP...")
    for row in cursor.execute("select sysdate from dual"):
        print(row)
```

This code snippet will print the current system date of the database host.

If the *cclass* parameter and SELF purity are not set, then the pooled server sessions will not be reused optimally, and the DRCP statistic views may record large values for NUM_MISSES.

DRCP allows the session memory of the connection to be reused or cleaned every time a connection is acquired from the pool. In pool or connection creation, the purity parameter value can be PURITY_NEW, PURITY_SELF, or PURITY_DEFAULT. By default, python-oracledb pooled connections use PURITY_SELF, and standalone connections use PURITY_NEW.

Setting the Connection Class and Purity in the Connection String

For the python-oracledb Thin mode, you can specify the connection class and purity in the Easy Connect string itself for Oracle Databases from version 21c onwards. This removes the need to modify an existing application when you want to use DRCP:

```
dsn = "localhost/orclpdb:pooled?pool_connection_class=MYAPP&pool_purity=self"
```

DRCP with JDBC

Oracle JDBC drivers support DRCP.

The DRCP implementation creates a pool on the server side, which is shared across multiple client pools. JDBC applications use Universal Connection Pool (UCP) for client-side pooling. UCP significantly lowers memory consumption (because of the reduced number of server processes) and increases the scalability of the database layer.

Java applications must use a client-side pool such as UCP for JDBC or a third-party Java connection pool to track check-in and checkout operations of the server-side connections. The benefit of using UCP over a third-party client pool is that UCP transparently takes care of attaching and detaching server connections.

If UCP is not used for any reason, the connections must use *attachServerConnection()* and *detachServerConnection()* functions to attach and detach connections to the Connection Broker respectively.

To enable DRCP on the client side, you must do the following:

- Pass a non-NULL, non-empty string value to the DRCP connection class property

```
oracle.jdbc.DRCPConnectionClass
```

- Pass (SERVER=POOLED) in the long connection string.

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=<hostname>)(PORT=<port>))(CONNECT_DATA=
(SERVICE_NAME=<service name>)(SERVER=POOLED)))
```

You can also specify (SERVER=POOLED) in the short URL form as follows:

```
jdbc:oracle:thin:@//<host>:<port>/<service_name>[:POOLED]
```

For example:

```
jdbc:oracle:thin:@//localhost:5221/orclpdb:POOLED
```

By setting the same DRCP Connection class name for all the pooled server processes using the connection property *oracle.jdbc.DRCPConnectionClass*, you can share pooled server processes on the server across multiple connection pools.

In DRCP, you can also apply a tag to a given connection and easily retrieve that tagged connection later.

Enabling DRCP on the client side using Universal Connection Pool (UCP)

The *PoolDataSource* object from the *oracle.ucp.jdbc* package is used to create the UCP.

```
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;
import java.util.Scanner;
import oracle.ucp.jdbc.PoolDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;

public class DRCPSamplewithUCP{
final static String url = "jdbc:oracle:thin:@//localhost:1522/orclpdb:POOLED";

static public void main(String args[]) throws SQLException {
 PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
 pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
 Scanner sc = new Scanner(System.in);

 // Set DataSource Properties - Get DB credentials as input
 System.out.print("Enter the DB User name: ");
 String dbUser = sc.nextLine();
 System.out.print("Enter the DB password: ");
 String dbPassword = sc.nextLine();
```

```
System.out.println ("Connecting to " + url);
pds.setUser(dbUser);
pds.setPassword(dbPassword);
pds.setURL(url);

//Set UCP Properties
pds.setInitialPoolSize(1);
pds.setMinPoolSize(4);
pds.setMaxPoolSize(20);

// Get the Database Connection from Universal Connection Pool.
try (Connection conn = pds.getConnection()) {
  System.out.println("\nConnection obtained from UniversalConnectionPool");
    // Perform a database operation
     doSQLWork(conn);
     System.out.println("Connection returned to the UniversalConnectionPool");
   }
}

 // Displays system date (sysdate).
 public static void doSQLWork(Connection connection) throws SQLException {
   // Statement and ResultSet are auto-closable by this syntax
   try (Statement statement = connection.createStatement()) {
     try (ResultSet resultSet = statement
         .executeQuery("select SYSDATE from DUAL")) {
       while (resultSet.next())
         System.out.print("Today's date is " + resultSet.getString(1) + " ");
     }
   }
   System.out.println("\n");
  }
}
```

This code prints the system date using a UCP JDBC connection that uses DRCP. In this example, we used the Easy Connect Syntax.

JDBC connect string also supports the TNS URL Format

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=<protocol>)
(HOST=<dbhost>)(PORT=<dbport>)) (CONNECT_DATA=(SERVICE_NAME=<service-name>))
```

Setting Connection Class and Purity attributes

The connection class is set through the *oracle.jdbc.DRCPConnectionClass* property. If UCP is used, then the connection class defaults to the UCP pool's name if one is set and otherwise to the randomly generated one.

You can set the connection class name through a Java *Properties* object and add it to the UCP's Connection properties.

```
Properties prop = new Properties();
 prop.put("oracle.jdbc.DRCPConnectionClass", "MyConClass");
 pds.setConnectionProperties(prop);
```

The purity of the DRCP connection is set via the *oracle.jdbc.DRCPPurity* property (default: SELF) if required.

```
prop.put("oracle.jdbc.DRCPPurity", "NEW");
```

DRCP with Oracle Call Interface (OCI)

Oracle Call Interface (OCI) libraries provide APIs in the C language to access and work with Oracle Database. The OCI session pool APIs *OCISessionPoolCreate()*, *OCISessionGet()*, and *OCISessionRelease()* should be used with DRCP for the best performance.

An OCI application initializes the environment for the OCI session pool for DRCP by invoking *OCISessionPoolCreate()*, which is described in the *Oracle Call Interface Programmer's Guide*. Use *OCISessionPoolCreate()* with *sessMin*, *sessMax,* and *sessIncr* parameters set appropriately for the application and the DRCP pool settings. In single-threaded applications using DRCP, set *sessMin* to 0 or 1 and *sessMax* to 1. In multi-threaded applications using DRCP and application-side connection pooling, set the *sessMin* and *sessMax* (1 or higher) parameters based on the threading requirements.

To get a session from the OCI session pool for DRCP, an OCI application invokes *OCISessionGet()*, specifying *OCI_SESSGET_SPOOL* for the mode parameter.

To release a session to the OCI session pool for DRCP, an OCI application invokes *OCISessionRelease()*.

The OCI session pool can transparently cache connections to the connection broker to improve performance. An OCI application can reuse the sessions of a similar state by setting the connection class in the *OCIAttrSet() function with the OCI_ATTR_CONNECTION_CLASS attribute* using the *OCIAuthInfo* handle before invoking *OCISessionGet()*.

Session purity specifies whether an OCI application can reuse a pooled session (*OCI_SESSGET_PURITY_SELF*) or must use a new session (*OCI_SESSGET_PURITY_NEW*).

The *OCISessionGet()* function can take in a session purity setting value of *OCI_SESSGET_PURITY_NEW* or *OCI_SESSGET_PURITY_SELF*. Alternatively, the application can set *OCI_ATTR_PURITY_NEW* or *OCI_ATTR_PURITY_SELF* on the *OCIAuthInfo* handle before calling *OCISessionGet()*. Both these methods are equivalent. The default purity value for the OCI session pool is SELF, and a standalone connection is NEW.

Session Purity and Connection Class behavior of OCI functions with DRCP

The following table indicates how this feature works with applications using Oracle Call Interface (OCI) libraries:

APPLICATION PURITY	CONNECT STRING PURITY	APPLICATION SESSION RELEASE MODE	SESSION RELEASE BEHAVIOR	SESSION GET BEHAVIOR
Unset or NEW or SELF	NEW	Default/ OCI_SESSRLS_DROPSESS (OCI attribute to drop the session)	The session will be dropped	New session
NEW	SELF	Default	The session will be retained	If a matching session is found, you will get an existing session; else, a new session
NEW	SELF	OCI_SESSRLS_DROPSESS	The session will be retained	If a matching session is found, you will get an existing session; else, a new session
SELF	SELF	Default	The session will be retained	If a matching session is found, you will get an existing session; else, a new session
SELF	SELF	OCI_SESSRLS_DROPSESS	The session will be dropped	If a matching session is found, you will get an existing session; else, a new session

Table 8 – Session Purity and Connection Class Behavior with DRCP

The default purity is NEW for non-session pool applications and SELF for session pool applications.

When *POOL_PURITY=SELF* in the connect string, session reuse is desired. Session having NEW purity and specifying *OCI_SESSRLS_DROPSESS* in *OCISessionRelease()* mandates to drop the session, preventing session reuse.

For applications that specify *POOL_PURITY*=SELF through the connect string and session having NEW purity and *OCI_SESSRLS_DROPSESS* in *OCISessionRelease()*, it will be perceived that the session reuse capability is of higher importance. So *OCI_SESSRLS_DROPSESS* will be ignored by the server, and sessions will not be dropped.

An application that wants to give precedence to *OCI_SESSRLS_DROPSESS* in *OCISessionRelease()* rather than session reuse should not use POOL_PURITY=SELF in connect string.

Setting the Connection Class and Session Purity attributes for a new session

The following code snippet shows how a connection pooling OCI application sets up a new DRCP session.

```
#include <oci.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>

OraText userName[129];
OraText userPassword[129];


/* Request a "pooled" connection */
const OraText connectString[] =
"localhost:1522/orclpdb.oradev.oraclecorp.com:pooled";


/* DRCP connection class name */
const OraText connectionClassName[] = "OCIConnectionPool";

int main(int argc, char** argv)

{

 OCIEnv *envhp = NULL;
 OCIError *errhp = NULL;
 OCIAuthInfo *authInfop = NULL;
 OCISvcCtx *svchp = NULL;
 OraText *poolName = NULL;
 ub4 poolNameLen = 0;
 OCISPool *spoolhp = NULL;
 int rc;


 /* Add error handling and other variables as required */

 //Initialize the DB Context
 rc = OCIEnvNlsCreate(&envhp, OCI_DEFAULT, 0, NULL, NULL, NULL, 0, NULL, 0, 0);
 /* Add error handling code for consistency below ... */


 //Initialize all the handles (error, authentication & session pool)
 rc = OCIHandleAlloc(envhp, (void **)&errhp, OCI_HTYPE_ERROR, 0, NULL);
 /* Add any error handling code below ... */


 rc = OCIHandleAlloc(envhp, (void **)&authInfop, OCI_HTYPE_AUTHINFO, 0, NULL);
 /* Add any error handling code below ... */


 rc = OCIHandleAlloc(envhp, (void **)&spoolhp, OCI_HTYPE_SPOOL, 0, NULL);
 /* Add any error handling code below ... */


 // Get the DB credentials through user input (recommended)
 printf("Enter the DB username: ");
 scanf("%s", userName);
 printf("Enter the DB password: ");
 scanf("%s", userPassword);
```

```
 //Create the Session Pool
 rc = OCISessionPoolCreate(envhp, errhp, spoolhp, &poolName, &poolNameLen,
 connectString, strlen((char *)connectString), 0, UB4MAXVAL, 1,
 (OraText *)userName, strlen((char *)userName), (OraText *)userPassword,
 strlen((char *)userPassword), OCI_SPC_NO_RLB | OCI_SPC_HOMOGENEOUS);


// OCIAttrSet method for setting the Connection Class name
 OCIAttrSet(authInfop, OCI_HTYPE_AUTHINFO, (dvoid *)connectionClassName, (ub4)
 strlen((char *)connectionClassName), OCI_ATTR_CONNECTION_CLASS, errhp);


 // OCISessionGet mode method
 OCISessionGet (envhp, errhp, &svchp, authInfop, poolName, poolNameLen, NULL, 0,
NULL, NULL, NULL, OCI_SESSGET_SPOOL);
/* Add the DB query code below... */


 // Destroy the Session Pool and Free the handles
 OCISessionPoolDestroy(spoolhp, errhp, OCI_DEFAULT);
 OCIHandleFree((dvoid *)spoolhp, OCI_HTYPE_SPOOL);
 OCIHandleFree((dvoid *)authInfop, OCI_HTYPE_AUTHINFO);
 OCIHandleFree((dvoid *)errhp, OCI_HTYPE_ERROR);
 OCIHandleFree((dvoid *)envhp, OCI_HTYPE_ENV);
}
```

To set the purity value for the connections, use *OCIAttrSet()* function:

```
 ub4 purity = OCI_ATTR_PURITY_NEW;
 OCIAttrSet(authInfop, OCI_HTYPE_AUTHINFO, &purity, (ub4)sizeof(purity),
 OCI_ATTR_PURITY, errhp);
```

DRCP with ODP.NET

ODP.NET[5] has three driver types: Core, Managed, and Unmanaged. Unmanaged ODP.NET uses OCI (Oracle Call Interface) libraries and runs on .NET framework. Managed and Core ODP.NET have 100% managed code that directly works with Oracle Database and runs on .NET framework and .NET Core, respectively.

Here is a sample code that uses Managed/Core ODP.NET driver code:

```
// This application uses the following connect descriptor:
// oracle =
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=<hostname>)(PORT=<port>))(CONNECT_DATA=
(SERVICE_NAME=<service name>)(SERVER=POOLED)))
```

[5] Oracle Data Provider for .NET – Oracle's Implementation of ADO .NET data provider for Oracle Database

```
using System;
using Oracle.ManagedDataAccess.Client;

class DRCP
{
    static void Main()
      {
            string constr = "user id=hr;password=hr;data source=oracle";
            OracleConnection con = new OracleConnection(constr);
            con.DRCPConnectionClass = "GroupA";
            con.Open();
            con.Dispose();
      }
}
```

We can also use the Easy Connect Syntax for specifying the *data source* attribute of the connect string (with DRCP enabled) as

```
data source=//<hostname>:<port>/<service_name>:pooled
```

A valid connect string for a DRCP connection with Easy Connect Syntax would be

```
"user id=hr;password=hr;data source=//localhost:1522/orclpdb.us.acme.com:pooled"
```

Enabling DRCP with Unmanaged ODP.NET requires some additional configuration as follows:

- Set the ODP.NET configuration file setting, *CPVersion* to 2.0 or,
- If the *CPVersion* configuration option is not set, have *(SERVER=POOLED)* in the TNS alias used by the application or '*:pooled*' in the Easy Connect String.

Setting ODP.NET Connection Class and Session Purity properties

For DRCP connections to be shared across multiple client-side ODP.NET connection pools, set the *OracleConnection.DRCPConnectionClass* property to a string value before opening the ODP.NET connection. This property will set the connection class for that connection. ODP.NET will initially try to obtain an idle connection with the same DRCP connection class property value. If it does not find one, it will establish a new connection.

For example, The following line in the earlier ODP.NET code sample sets the connection class.

```
con.DRCPConnectionClass = "GroupA";
```

This property can be used to set and get the connection class names. Its value is unique to each client-side connection pool. The default value of this property is *null*. The character limit is 1024 minus the number of characters in the user id. This property must be set before opening the connection if used.

The default purity value in ODP.NET is SELF (*Pooled* in ODP.NET parlance) and is the recommended value.

To set the DRCP Purity attribute to NEW, use the *OracleConnection.DRCPPurity* property in the earlier ODP.NET code sample as follows:

```
con.DRCPPurity = = OracleConnection.OracleDRCPPurity.New;
```

DRCP with Node.js

To use DRCP in node-oracledb, the Node.js driver for Oracle Database:

The *oracledb.createPool()* or *oracledb.getConnection()*'s property *connectString* (or its alias *connectionString*) must specify to use a pooled server, either by the Easy Connect syntax like *myhost/sales:POOLED*, or by using a tnsnames.ora alias for a connection that contains *(SERVER=POOLED)*.

For efficiency, it is recommended that DRCP connections should be used with node-oracledb's local connection pool.

Example:

```
tnsnames.ora:

customerpool = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=dbhost.example.com)
(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=CUSTOMER)(SERVER=POOLED)))
```

or

```
Easy Connect: dbhost.us.oracle.com:2222/dbsvc.company.com:POOLED
```

With TNS alias, use:

```
const oracledb = require('oracledb');

pool = await oracledb.createPool({

user: "scott", password: "tiger", connectString: "customerpool", poolMax: 1,
poolMin: 1,  poolPingInterval: 0 });

connection = await pool.getConnection();
```

With Easy Connect syntax, use:

```
const oracledb = require("oracledb");

pool = await oracledb.createPool({

  user: "scott",

  password: "tiger",

  connectString: " dbhost.us.oracle.com:2222/dbsvc.company.com:POOLED",

  poolMax: 1,

  poolMin: 1,

  poolPingInterval: 0,

});
```

```
connection = await pool.getConnection();
```

when you are using node-oracledb's local connection pool.

For standalone connections, the following code snippets will work:

TNS Alias pointing to DRCP

```
const connection = await oracledb.getConnection({
user: "scott",
password: "tiger",
connectString: "customerpool"
    });
```

or

Easy Connect String pointing to DRCP

```
const connection = await oracledb.getConnection({
user: "scott",
password: "tiger",
connectString: "dbhost.us.oracle.com:2222/dbsvc.company.com:POOLED"
    });
```

Note that all the above code snippets should be written inside an *Async* function. It is recommended to embed all of these code snippets in a *try-catch-finally* block.

You can use this connection object to run any database transactions.

```
console.log("System Date:");
result = await connection.execute(
  `SELECT sysdate
   FROM dual`
);
let ts = result.rows[0][0];
console.log(ts);
if (connection) await connection.close();
```

This will print the current system date of the database host and then close the connection.

Setting Connection Class and Purity

Node-oracledb provides the *connectionClass* attribute to set a connection class name.

```
oracledb.connectionClass = "NodePool";
```

The 'Purity' value is always SELF for DRCP connections with node-oracledb. This allows the reuse of both pooled server process and session memory, giving maximum benefit from DRCP. There is no separate parameter or function for setting the purity for connections in node-oracledb.

The connection class and purity values can also be set using the Easy Connect syntax shown in the *DRCP with Python* sub-section.

DRCP with PHP

The OCI8 1.4 extension for PHP can be used with Oracle client libraries version 9.2 and higher. However, DRCP functionality is only available when PHP is linked with Oracle 11g client libraries and connects to Oracle Database 11g.

Once installed, use PHP's *phpinfo()* function to verify that OCI8 has been loaded.

Before using DRCP, the new *php.ini* parameter *oci8.connection_class* should be set to specify the connection class used by all the requests for pooled servers by the PHP application.

```
oci8.connection_class = MYPHPAPP
```

The parameter can be set in *php.ini*, *.htaccess,* or *httpd.conf* files. It can also be set and retrieved programmatically using the PHP functions *ini_set()* and *ini_get()*.

PHP Application Deployment for DRCP

PHP applications must specify the server type POOLED in the connect string to use DRCP. Using Oracle's Easy Connect syntax, the PHP call to connect to the sales database on *myhost* would look like this:

```
$c = oci_pconnect('myuser', 'mypassword', 'myhost/sales:POOLED');
```

If PHP uses an Oracle Network alias that looks like:

```
$c = oci_pconnect('myuser', 'mypassword', 'salespool');
```

Then only the Oracle Network configuration file *tnsnames.ora* needs to be modified:

```
salespool=(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)

(HOST=myhost.domain.com)(PORT=1521))

(CONNECT_DATA= (SERVICE_NAME=sales)(SERVER=POOLED)))
```

It is recommended that *oci_close($c)* is called immediately after completing the database work, without leaving it to the implicit connection closure that happens at the end of the script.

Note: *oci_pconnect()* uses SELF purity, while the other connect functions of PHP, *oci_connect()* and *oci_new_connect()* use NEW purity.

Conclusion

DRCP allows applications to use a connection pool in the database shared across multiple application servers and mid-tier deployments. These applications must actively wrap database activity with calls to get or release sessions to use DRCP effectively. Such applications can establish connections quickly and use minimal database resources for large numbers of connections.

Oracle's database proxy solution, Connection Manager in Traffic Director Mode (CMAN-TDM) has its own pooling feature – Proxy Resident Connection Pooling (PRCP), which works similarly to DRCP. If an application works well with DRCP, it will work just as well with PRCP. The only change necessary (on the application side) for PRCP is that the TNS alias or Easy Connect string should point to the PRCP server instead of the database/DRCP server.

To sum it all up, the benefits of DRCP are as follows:

- DRCP allows resources to be shared among multiple client applications and middle-tier application servers
- DRCP improves the scalability of databases and applications by reducing resource usage on the database host

More Information

For more information, please refer to the following links and documents:

- Understanding DRCP, Oracle Database Administrator's Guide
- Using Database Resident Connection Pool, Oracle Database Developer Guide
- Database Resident Connection Pooling, Oracle Database Concepts
- Database Resident Connection Pooling, Oracle Call Interface Programmer's Guide
- Database Resident Connection Pooling, Oracle Data Provider for .NET Developer's Guide
- Database Resident Connection Pooling, Oracle JDBC Developer's Guide
- New DRCP Parameters, Oracle New Features Guide
- CMAN-TDM – An Oracle Database connection proxy for scalable and highly available applications, CMAN-TDM Technical Brief
- Oracle Database 21c Easy Connect Plus, A Technical Brief on Easy Connect and Easy Connect Plus
- Application Programming using Pooling and Caching, A Technical Brief on pooling and caching of Oracle Database resources
- GOL tracks ticket purchases in under 60 seconds using Oracle Cloud Infrastructure, Oracle Customer References

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle