28

Database Resident Connection Pooling

Database Resident Connection Pool (DRCP) is a connection pool in the server, which many clients can share. You should use DRCP when the number of active connections, at a given point of time, is reasonably less than the number of open connections. DRCP is particularly useful for applications with a large number of middle-tier servers. The Database Resident Connection Pooling feature increases Database server scalability and resolves the resource wastage issues of middle-tier connection pooling.

This chapter contains the following sections:

28.1 Overview of Database Resident Connection Pooling

In middle-tier connection pools, every connection pool maintains a minimum number of open connections to the server. Each open connection represents resources that are in use at the server. However, your application does not use all the open connections at a given point of time, which means that there are unused resources that consume server resources unnecessarily. In a multiple middle-tier scenario, every middle tier maintains an individual connection pool, without sharing connections with other middle-tier connection pools, and retains the open connections for long, even if some of those are inactive. For an application with a large number of middle-tier connection pools, the number of inactive connections to the Database server is significantly large in such case, which wastes a lot of Database resources.

For example, if in your application, the minimum pool size is 200 for every middle-tier connection pool, the connection pool has 200 open connections to the server, and the Database server has 200 server processes associated with these connections. If you have 30 similar middle-tier connection pools in your application, then the server has 6000 (200 * 30) corresponding server processes running at any given point of time. Typically, on an average only 5% of the connections, and in turn, server processes are in use simultaneously. So, out of the 6,000 server processes, only 300 server processes are active at any given time. This leads to 5,700 unused processes on the server, resulting in wasted resources on the server.

The Database Resident Connection Pool implementation creates a pool on the Database server side, which is shared across multiple client pools. This significantly lowers memory consumption on the Database server because of reduced number of server processes running simultaneously and increases its scalability.

> **See Also:**
>
> - *Oracle Database Concepts*
> - *Oracle Database Administrator's Guide*

28.2 Enabling Database Resident Connection Pooling

This section describes how to enable Database Resident Connection Pooling (DRCP) on the server side and the client side:

- Enabling DRCP on the Server Side
- Enabling DRCP on the Client Side

28.2.1 Enabling DRCP on the Server Side

You must be a database administrator (DBA) and must log on as `SYSDBA` to start and end a Database Resident Connection Pool (DRCP). This section discusses the following related concepts:

- Starting a Database Resident Connection Pool
- Configuring a Database Resident Connection Pool
- Ending a Database Resident Connection Pool
- Setting the Statement Cache Size

> **Note:**
>
> You can leverage the DRCP features only with a connection pool on the client because JDBC does not have a default pool on its own.

Starting a Database Resident Connection Pool

Run the `start_pool` method in the `DBMS_CONNECTION_POOL` package with the default settings to start the default Oracle Database resident connection pool, namely, `SYS_DEFAULT_CONNECTION_POOL`. For example:

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

Configuring a Database Resident Connection Pool

Use the procedures in the `DBMS_CONNECTION_POOL` package to configure the default Oracle Database resident connection pool. By default, this connection pool uses default parameter values.

> **See Also:**
>
> - DBMS_CONNECTION_POOL
> - Configuring Database Resident Connection Pooling

Ending a Database Resident Connection Pool

Run the `stop_pool` method in the `DBMS_CONNECTION_POOL` package to end the pool. For example:

```
sqlplus /nolog
connect / as sysdba
execute dbms_connection_pool.stop_pool();
```

Setting the Statement Cache Size

If you use DRCP, then the server caches statement information on the server side. So, you must specify the statement cache size on the server side in the following way, where 50 is the preferred size:

```
execute DBMS_CONNECTION_POOL.CONFIGURE_POOL (session_cached_cursors=>50);
```

Related Topics

- About Statement Caching

28.2.2 Enabling DRCP on the Client Side

Perform the following steps to enable DRCP on the client side:

> **Note:**
>
> The example in this section uses Universal Connection Pool (UCP) as the client-side connection pool. For any other connection pool, you must use `oracle.jdbc.pool.OracleConnectionPoolDataSource` as the connection factory.

- Pass a non-null and non-empty `String` value to the `oracle.jdbc.DRCPConnectionClass` connection property
- Append (`SERVER=POOLED`) to the `CONNECT_DATA` configuration parameter in the long connection string

You can also specify `(<service_name>=POOLED)` in the short connection string in the following way:

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

For example:

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

The following code snippet shows how to enable DRCP on the client side:

> **Note:**
>
> In UCP, if you do not provide a connection class, then the connection pool name is used as the connection class name by default.

Example 28-1 Enabling DRCP on Client Side Using Universal Connection Pool

```
String url = "jdbc:oracle:thin:@localhost:5521/orcl:POOLED";
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
// Set DataSource Property
pds.setUser("HR");
pds.setPassword("hr");
System.out.println ("Connecting to " + url);
pds.setURL(url);
pds.setConnectionPoolName("HR-Pool1");
pds.setMinPoolSize(2);
pds.setMaxPoolSize(3);
pds.setInitialPoolSize(2);
Properties prop = new Properties();
prop.put("oracle.jdbc.DRCPConnectionClass", "HR-Pool1");
pds.setConnectionProperties(prop);
```

28.3 Pooled Server Processes Across Multiple Connection Pools

You can set the same Database Resident Connection Pool (DRCP) connection class name for all the pooled server processes on the server and share those across multiple connection pools on the server. For setting the DRCP connection class name, use the `oracle.jdbc.DRCPConnectionClass` connection property.

28.4 Multi-Pool Support in DRCP

You can now develop applications that use multiple, named DRCP pools. This helps in avoiding situations, where connections from a few services can occupy all the pooled servers of DRCP, while the rest of the connections from the other services may need to wait for the availability of the pooled servers.

For making DRCP mark the connection against the appropriate pool, specify `(POOL_NAME=<pool_name>)` in the connection string, along with `(SERVER=POOLED)`, as shown in the following example:

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=<host>)(PORT=<port>))
(CONNECT_DATA=(SERVER=POOLED)(POOL_NAME=<pool_name>)))
```

It is not mandatory to create a DRCP pool for all the services, a few services may have multi-pool DRCP and the rest of the services may share the per-PDB or CDB-wide DRCP. If you are a PDB administrator, then by connecting to a PDB, you can also perform administrative operations like creating and destroying the DRCP.

The following procedures have been added to the `DBMS_CONNECTION_POOL` package to add and remove pooled servers from the multi-pool DRCP:

ADD_POOL Procedure

This procedure adds a new pool to the multi-pool DRCP. For example:

```
exec dbms_connection_pool.add_pool('mypool')
```

REMOVE_POOL Procedure

This procedure removes a new pool from the multi-pool DRCP. For example:

```
exec dbms_connection_pool.remove_pool('mypool')
```

> **See Also:**
>
> - *Oracle Database Development Guide*
> - *Oracle Database PL/SQL Packages and Types Reference*

28.5 Tagging Support in Database Resident Connection Pooling

Database Resident Connection Pooling (DRCP) provides an option to associate a server process with a particular tag name. You can apply a tag to a given connection and retrieve that tagged connection later. Connection tagging enhances session pooling because you can retrieve specific sessions easily.

DRCP also provides support for multiple tagging, which is disabled by default. Set the `oracle.jdbc.UseDRCPMultipletag` connection property to `TRUE` for enabling this feature in your DRCP application.

Once you enable the multiple tagging feature, you can use the same APIs for setting single or multiple DRCP tags. The only difference between both the cases is the separator. You must separate the key and the value of a DRCP tag by an equal (`=`) character, whereas, you must separate multiple tags from one another by a semi-colon (`;`) character.

Remember the following points while working with DRCP tags:

- You cannot specify the key or the value of a tag as null or empty.
- When you specify multiple tags, then the leftmost tag has the highest priority and the rightmost tag has the lowest priority.
- While retrieving a tagged connection, if a complete match is not found (all tags are not matched), then it searches for a partial match.
- You cannot set the `RESET_STATE` service attribute to `LEVEL1` or `LEVEL2`.

> **See Also:**
>
> *Oracle Call Interface Programmer's Guide* for more information about session pooling and connection tagging

28.6 PL/SQL Callback for Session State Fix Up

Starting from Oracle Database 12*c* Release 2 (12.2.0.1), a PL/SQL based fix-up callback for the session state can be provided on the server. This application-provided callback transforms a session checked out from the pool to the desired state requested by the application. This callback works with or without Database Resident Connection Pooling (DRCP).

> **Note:**
>
> The PL/SQL based fix-up callback is only applicable for multiple tagging.

Using this callback can improve the performance of your application because the fix-up logic is run for the session state on the server. So, this feature eliminates application round-trips to the database for the fix-up logic. An appropriate installation user, who must be granted execute permissions on the related package, should register the fix-up callback during application installation.

Example 28-2 Example of PL/SQL Fix-Up Callback

Following is an example implementation of the PL/SQL fix up callback to fix up the session properties `SCHEMA` and `CURRENCY`:

```
CREATE OR REPLACE PACKAGE mycb_pack AS
PROCEDURE mycallback (
desired_props IN VARCHAR2,
actual_props IN VARCHAR2
);
END;
/

CREATE OR REPLACE PACKAGE BODY mycb_pack AS
PROCEDURE mycallback (
desired_props IN VARCHAR2,
actual_props IN VARCHAR2
) IS
property VARCHAR2(64);
key VARCHAR2(64);
value VARCHAR2(64);
pos number;
pos2 number;
pos3 number;
idx1 number;

BEGIN
idx1:=1;

pos:=1;
pos2:=1;
pos3:=1;
property := 'tmp';
-- To check if desired properties are part of actual properties
while (pos > 0 and length(desired_props)>pos)
loop
pos := instr (desired_props, ';', 1, idx1);
if (pos=0)
then
```

```
property := substr (desired_props, pos2);
else
property := substr (desired_props, pos2, pos-pos2);
end if ;
pos2 := pos+1;
pos3 := instr (property, '=', 1, 1);
key := substr (property, 1, pos3-1);
value := substr (property, pos3+1);
if (key = 'CURRENCY') then
EXECUTE IMMEDIATE 'ALTER SESSION SET NLS_CURRENCY=''' || value || '''';
elsif (key = 'SCHEMA') then
EXECUTE IMMEDIATE 'ALTER SESSION SET CURRENT_SCHEMA=' || value;
end if;
idx1 := idx1+1;
end loop;

END; -- mycallback
END mycb_pack;
/
```

> **See Also:**
>
> *Oracle Database JDBC Java API Reference*

28.7 APIs for Using Database Resident Connection Pooling

If you want to take advantage of Database Resident Connection Pooling (DRCP) with higher granular control for your custom connection pool implementations, then you must use the following APIs declared in the `oracle.jdbc.OracleConnection` interfaces:

- `attachServerConnection`
- `detachServerConnection`
- `isDRCPEnabled`
- `isDRCPMultitagEnabled`
- `getDRCPReturnTag`
- `needToPurgeStatementCache`
- `getDRCPState`

> **See Also:**
>
> *Oracle Database JDBC Java API Reference*