Session Pool Backend in Django For Oracle

|  |  |
| --- | --- |
| Author | Suraj Kumar Shaw |
| Manager | Krishna Mohan |
| Version | 16 Jan 2024 |

**Connection management**

Django creates standalone connection for connecting to database and processing request and it closes the connection after end of the request. So it creates a overhead of connection creation for every request.

Django provides persistent connection functionality as well. Persistent connections avoid the overhead of re-establishing a connection to the database in each HTTP request. They’re controlled by the [**CONN\_MAX\_AGE**](https://docs.djangoproject.com/en/4.2/ref/settings/#std-setting-CONN_MAX_AGE) parameter which defines the maximum lifetime of a connection.

The default value is **0**, preserving the historical behaviour of closing the database connection at the end of each request. To enable persistent connections, set [**CONN\_MAX\_AGE**](https://docs.djangoproject.com/en/4.2/ref/settings/#std-setting-CONN_MAX_AGE) to a positive integer of seconds.

**New Connection Management Proposal**

Oracle provides the feature of **connection pooling**. **Connection pooling** is the use of a group of reusable physical connections by several sessions to balance loads. It can handle concurrent requests efficiently. It can leverage **DRCP** as well with connection pooling. Database Resident Connection Pool (DRCP) is a connection pool in the server that is shared across many clients. All the benefits of **connection pooling** can be leveraged, for both **DRCP** and **non-DRCP.** DRCP is just one of the many advantages users can have when using connection pooling. Other advantages includes high availability and run-time load balancing.  
  
The new proposal is targeted for Django 5.0 and it will be supported with oracledb(thin and thick) and cx\_oracle as well.

**NOTE** – Use default value of CONN\_MAX\_AGE while using connection pooling.

**Configuration**

A new attribute **USE\_CONNECTION\_POOL** has been introduced in this proposal along with **‘POOL\_MIN’,’POOL\_MAX’, ‘POOL\_INCREMENT’,‘POOL\_CCLASS’, ‘POOL\_TIMEOUT’**. The value can be set through the settings.py file present in the Django project.

The database configuration in the settings.py file looks something like below.

DATABASES = {

'default': {

'ENGINE':'django.db.backends.oracle',

'NAME':'localhost:51521/orclpdb1',

'USER':'sks1',

'PASSWORD':'tiger',

'USE\_CONNECTION\_POOL':'True',

'POOL\_MIN':1,

'POOL\_MAX':10,

'POOL\_INCREMENT':1,

'POOL\_CCLASS':'ab',

}

}

Using the attribute **USE\_CONNECTION\_POOL** in the settings.py file and setting it as **TRUE** users will be able to use session pooling in Django. Default value for the parameter is being set as **FALSE** and based on the requirement user can turn it on.

At present it allows users to provide inputs for **‘POOL\_MIN’,’POOL\_MAX’, ‘POOL\_INCREMENT’,‘POOL\_CCLASS’, ‘POOL\_TIMEOUT’**. The default value for min is 0, max is 100, increment is 1 and timeout is 100. Users can alter the values through settings.py file. For other parameters it will use the default value provided by oracledb.

All parameters are optional. A brief description of each parameter follows:

**‘POOL\_MIN’**: the minimum number of connections the pool should contain.

**’POOL\_MAX’**: the maximum number of connections the pool should contain.

**‘POOL\_INCREMENT’**: the number of connections that should be added to the pool whenever a new connection needs to be created

**‘POOL\_CCLASS’**: connection class to use for Database Resident Connection Pooling (DRCP)

**‘POOL\_TIMEOUT’**: length of time (in seconds) that a connection may remain idle in the pool before it is terminated. If it is 0 then connections are never terminated

**Design Specification**

The feature is specific to oracle backend and the code changes are also in the oracle specific layer i.e. oracle dialect of Django.

When the parameter **USE\_CONNECTION\_POOL** is set it will create a pool if already not created and store the pool object in the **settings\_dict** itself with the **POOL** as key.

Once the request is served the call to conn.close() which is already present will release the connection back to pool for reuse. The pool itself will get closed automatically when it goes out of scope.

**if** self.settings\_dict["USE\_CONNECTION\_POOL"]:

pool **=** self.settings\_dict.get("POOL")

**if** pool **is** None:

pool **=** self.settings\_dict["POOL"] **=** Database.create\_pool(

user**=**self.settings\_dict["USER"],

password**=**self.settings\_dict["PASSWORD"],

dsn**=**dsn(self.settings\_dict),

min**=**self.settings\_dict["POOL\_MIN"],

max**=**self.settings\_dict["POOL\_MAX"],

increment**=**self.settings\_dict["POOL\_INCREMENT"],

cclass**=**self.settings\_dict["POOL\_CCLASS"],

timeout**=**self.settings\_dict["POOL\_TIMEOUT"],

**\*\***conn\_params,

)

**return** pool.acquire()

The setting of default values is done in utils.py file which is in generic layer

The default values for the input parameters are as follows.

conn.setdefault("USE\_CONNECTION\_POOL", False)

conn.setdefault("POOL\_MIN", 0)

conn.setdefault("POOL\_MAX", 100)

conn.setdefault("POOL\_INCREMENT", 1)

conn.setdefault("POOL\_CCLASS", None)

conn.setdefault("POOL\_TIMEOUT", 100)

**Resources**

Blog post: [Always Use Connection Pools — and How](https://medium.com/@cjones-oracle/always-use-connection-pools-and-how-909bc2c65444)

Blog post: [Understanding DRCP](https://medium.com/oracledevs/oracle-database-drcp-and-node-js-618d84b63e77)

DRCP Technical Brief: [Extreme Oracle Database Connection Scalability with Database Resident Connection Pooling (DRCP)](https://www.oracle.com/docs/tech/drcp-technical-brief.pdf)