11

Proxy Authentication

Oracle Java Database Connectivity (JDBC) provides proxy authentication, also called N-tier authentication. This feature is supported through both the JDBC Oracle Call Interface (OCI) driver and the JDBC Thin driver. This chapter contains the following sections:

> **Note:**
>
> Oracle Database supports proxy authentication functionality in three tiers *only*. It does not support it across multiple middle tiers.

11.1 About Proxy Authentication

Proxy authentication is the process of using a middle tier for user authentication.

You can design a middle-tier server to proxy clients in a secure fashion by using the following three forms of proxy authentication:

- The middle-tier server authenticates itself with the database server and a client. In this case, an application user or another application, authenticates itself with the middle-tier server. Client identities can be maintained all the way through to the database.
- The client, that is, a database user, is not authenticated by the middle-tier server. The client's identity and the database password are passed through the middle-tier server to the database server for authentication.
- The client, that is, a global user, is authenticated by the middle-tier server, and passes either a Distinguished name (DN) or a Certificate through the middle tier for retrieving the client's user name.

> **Note:**
>
> Operations done on behalf of a client by a middle-tier server can be audited.

In all cases, an administrator must authorize the middle-tier server to proxy a client, that is, to act on behalf of the client. Suppose, the middle-tier server initially connects to the database as

user HR and activates a proxy connection as a user called, jeff. It then authorizes the middle-tier server to proxy a client:

```
CREATE USER jeff;
ALTER USER jeff GRANT CONNECT THROUGH HR;
```

You can also:

- Specify roles that the middle tier is permitted to activate when connecting as the client. For example,

  ```
  CREATE ROLE role1;
  GRANT SELECT ON employees TO role1;
  GRANT ROLE1 TO jeff;
  ALTER USER jeff GRANT CONNECT THROUGH HR WITH ROLE role1;
  ```

  The role clause limits the access only to those database objects that are mentioned in the list of the roles. The list of roles can be empty.
- Find the users who are currently authorized to connect through a middle tier by querying the PROXY_USERS data dictionary view.
- Disallow a proxy connection by using the REVOKE CONNECT THROUGH clause of ALTER USER statement.

> **Note:**
>
> In case of proxy authentication, a JDBC connection to the database creates a database session during authentication, and then other sessions can be created during the life time of the connection.

You need to use the different fields and methods present in the oracle.jdbc.OracleConnection interface to set up the different types of proxy connections.

11.2 Types of Proxy Connections

You can create proxy connections using either a user name or a distinguished name.

- USER NAME

  This is done by supplying the user name or the password or both. The SQL statement for specifying authentication using password is:

  ```
  ALTER USER jeff GRANT CONNECT THROUGH HR AUTHENTICATION REQUIRED;
  ```

  In this case, jeff is the user name and HR is the proxy for jeff.

  The password option exists for additional security. Having no authenticated clause implies default authentication, which is using only the user name without the password. The SQL statement for specifying default authentication is:

  ```
  ALTER USER jeff GRANT CONNECT THROUGH HR
  ```

- DISTINGUISHED NAME

This is a global name in lieu of the password of the user being proxied for. An example of the corresponding SQL statement using a distinguished name is:

```
CREATE USER jeff IDENTIFIED GLOBALLY AS
'CN=jeff,OU=americas,O=oracle,L=redwoodshores,ST=ca,C=us';
```

The string that follows the `identified globally as` clause is the distinguished name. It is then necessary to authenticate using this distinguished name. The corresponding SQL statement to specify authentication using distinguished name is:

```
ALTER USER jeff GRANT CONNECT THROUGH HR;
```

> **Note:**
>
> - All the options can be associated with roles.
> - When opening a new proxied connection, a new session is started on the Database server. If you start a global transaction and then call the `openProxySession` method, then, at this point, you are no longer a part of the global transaction and instead it is like you are in a freshly created JDBC connection. Typically, this never happens because the `openProxySession` method is called prior to creating or resuming a global transaction. In such a case, you are still a part of the global transaction.

11.3 Creating Proxy Connections

A user, say `jeff`, has to connect to the database through another user, say `HR`. The proxy user, `HR`, should have an active authenticated connection. A proxy session is then created on this active connection, with the driver issuing a command to the server to create a session for the user, `jeff`. The server returns the new session ID, and the driver sends a session switch command to switch to this new session.

The JDBC OCI and Thin driver switch sessions in the same manner. The drivers permanently switch to the new session, `jeff`. As a result, the proxy session, `HR`, is not available until the new session, `jeff`, is closed.

> **Note:**
>
> You can use the `isProxySession` method from the `oracle.jdbc.OracleConnection` interface to check if the current session associated with your connection is a proxy session. This method returns `true` if the current session associated with the connection is a proxy session.

A new proxy session is opened by using the following method from the `oracle.jdbc.OracleConnection` interface:

```
void openProxySession(int type, java.util.Properties prop) throws SQLExceptionOpens
```

Where,

`type` is the type of the proxy session and can have the following values:

- `OracleConnection.PROXYTYPE_USER_NAME`

This type is used for specifying the user name.

- `OracleConnection.PROXYTYPE_DISTINGUISHED_NAME`

  This type is used for specifying the distinguished name of the user.

`prop` is the property value of the proxy session and can have the following values:

- `PROXY_USER_NAME`

  This property value should be used with the type `OracleConnection.PROXYTYPE_USER_NAME`. The value should be a `java.lang.String`.

- `PROXY_DISTINGUISHED_NAME`

  This property value should be used with the type `OracleConnection.PROXYTYPE_DISTINGUISHED_NAME`. The value should be a `java.lang.String`.

- `PROXY_ROLES`

  This property value can be used with the following types:

  - `OracleConnection.PROXYTYPE_USER_NAME`
  - `OracleConnection.PROXYTYPE_DISTINGUISHED_NAME`

  The value should be a `java.lang.String`.

- `PROXY_SESSION`

  This property value is used with the `close` method to close the proxy session.

- `PROXY_USER_PASSWORD`

  This property value should be used with the type `OracleConnection.PROXYTYPE_USER_NAME`. The value should be a `java.lang.String`.

The following code snippet shows the use of the `openProxySession` method:

```
java.util.Properties prop = new java.util.Properties();
prop.put(OracleConnection.PROXY_USER_NAME, "jeff");
String[] roles = {"role1", "role2"};
prop.put(OracleConnection.PROXY_ROLES, roles);
conn.openProxySession(OracleConnection.PROXYTYPE_USER_NAME, prop);
```

11.4 Closing a Proxy Session

You can close the proxy session opened with the `OracleConnection.openProxySession` method by passing the `OracleConnection.PROXY_SESSION` parameter to the `OracleConnection.close` method in the following way:

```
OracleConnection.close(OracleConnection.PROXY_SESSION);
```

This is similar to closing a proxy session on a non-cached connection. The standard `close` method must be called explicitly to close the connection itself. If the `close` method is called directly, without closing the proxy session, then both the proxy session and the connection are closed. This can be achieved in the following way:

```
OracleConnection.close(OracleConnection.INVALID_CONNECTION);
```

11.5 Caching Proxy Connections

Proxy connections, like standard connections, can be cached. Caching proxy connections enhances the performance. To cache a proxy connection, you need to create a connection using one of the `getConnection` methods on a cache enabled `OracleDataSource` object.

A proxy connection may be cached in the connection cache using the connection attributes feature of the connection cache. Connection attributes are name/value pairs that are user-defined and help tag a connection before returning it to the connection cache for reuse. When the tagged connection is retrieved, it can be directly used without having to do a round-trip to create or close a proxy session. Universal Connection Pool supports caching of any user/password authenticated connection. Therefore, any user authenticated proxy connection can be cached and retrieved.

It is recommended that proxy connections should not be closed without applying the connection attributes. If a proxy connection is closed without applying the connection attributes, the connection is returned to the connection cache for reuse, but cannot be retrieved. The connection caching mechanism does not remember or reset session state.

A proxy connection can be removed from the connection cache by closing the connection directly.

11.6 Limitations of Proxy Connections

Closing a proxy connection automatically closes every SQL Statement created by the proxy connection, during the proxy session or prior to the proxy session. This may cause unexpected consequences on application pooling or statement caching. The following code samples explain this limitation of proxy connections:

Example 1

```
....
public void displayName(String N)  // Any function using the Proxy feature
{
     Properties props = new Properties();
     props.put("PROXY_USER_NAME", proxyUser);
     c.openProxySession(OracleConnection.PROXYTYPE_USER_NAME, props);
     .......
     c.close(OracleConnection.PROXY_SESSION);
}

public static void main (String args[]) throws SQLException
{
    ............
    PreparedStatement pstmt = conn.prepareStatement("SELECT first_name FROM EMPLOYEES
WHERE employee_id = ?");
    pstmt.setInt(1, 205);
    ResultSet rs = pstmt.executeQuery();
    while (rs.next())
    {
        displayName(rs.getString(1));
         if (rs.isClosed() // The ResultSet is already closed while closing the
connection!
         {
             throw new Exception("Your ResultSet has been prematurely closed!
Your Statement object is also dead now.");
         }
```

```
    }
}
```

In the preceding example, when you close the proxy connection in the `displayName` method, then the `PreparedStatement` object and the `ResultSet` object also get closed. So, if you do not check the status of the `ResultSet` object inside loop, then the loop will fail when the `next` method is called for the second time.

Example 2

```
    ....
    PreparedStatement pstmt = conn.prepareStatement("SELECT first_name FROM EMPLOYEES
WHERE employee_id = ?");
    pstmt.setString(1, "205");
    ResultSet rs = pstmt.executeQuery();
    while (rs.next())
    {
        ....
    }

    Properties props = new Properties();
    props.put("PROXY_USER_NAME", proxyUser);

    conn.openProxySession(OracleConnection.PROXYTYPE_USER_NAME, props);
    .......
    conn.close(OracleConnection.PROXY_SESSION);

    // Try to use the PreparedStatement again
    pstmt.setString(1, "28960");
// This line of code will fail because the Statement is already closed while closing the
connection!
    rs = pstmt.executeQuery();
```

In the preceding example, the `PreparedStatement` object and the `ResultSet` object work fine before opening the proxy connection. But, if you try to execute the same `PreparedStatement` object after closing the proxy connection, then the statement fails.