Consider a scenarion , Java based appliation having DB Threadpool and use connection from thread pool. and thredpool clas has new connection method as well along with get connection method , in teh calling application we do see only release method called , which is to release connection to pool . in this context to find resource leak what ar ethe potential cause and how do we simulate the connection leak and check in lower environment

**ChatGPT said:**

In this scenario, a Java-based application uses a **DB thread pool** where connections are retrieved via getConnection() and released using release(). However, the thread pool also has a newConnection() method, which may indicate that connections can be created outside the pool mechanism.

**Potential Causes of Resource Leak**

1. **Connections Not Released Properly**
   * If getConnection() is called but release() is not invoked due to exceptions or logic errors, connections will remain open indefinitely.
2. **Use of newConnection() Instead of Pool Connections**
   * If newConnection() creates fresh connections that are never added to the pool, they will not be managed properly and may cause resource exhaustion.
3. **Connection Leakage Due to Long-Running Transactions**
   * If a thread holds a connection without releasing it (e.g., waiting for a long-running query), it reduces the available connections in the pool.
4. **Application Restart Without Closing Connections**
   * If the application crashes or is restarted without properly closing active connections, these may remain open in the database.
5. **Thread Termination Without Cleanup**
   * If a thread using a connection is forcefully terminated, the connection may not be returned to the pool.
6. **Pool Exhaustion Leading to New Connections**
   * If the thread pool runs out of connections and newConnection() is frequently called, it may bypass the pooling mechanism and lead to an excessive number of open database connections.

**How to Simulate Connection Leak in a Lower Environment**

To detect leaks, you can simulate scenarios where connections are not properly released:

1. **Intentionally Skip Calling release()**
   * Modify a test case or introduce a code path where getConnection() is called but release() is not.
   * Example:

java

CopyEdit

Connection conn = threadPool.getConnection();

// Simulate failure: Do not call threadPool.release(conn);

1. **Create Excessive Connections Using newConnection()**
   * Call newConnection() multiple times in a loop without closing them to observe DB connection growth.
   * Example:

java

CopyEdit

for (int i = 0; i < 100; i++) {

Connection conn = threadPool.newConnection();

// Do not close the connection

}

1. **Execute a Long-Running Query Without Releasing Connection**
   * Run a SQL query that takes significant time to complete while holding a connection.
   * Example:

java

CopyEdit

Connection conn = threadPool.getConnection();

Statement stmt = conn.createStatement();

stmt.executeQuery("SELECT pg\_sleep(300)"); // Simulate delay in PostgreSQL

1. **Simulate Application Crash Without Cleanup**
   * Start multiple threads fetching connections but terminate them abruptly before calling release().
   * Example:

java

CopyEdit

Thread t = new Thread(() -> {

Connection conn = threadPool.getConnection();

// Simulate abrupt termination

System.exit(1);

});

t.start();

1. **Overload the Connection Pool**
   * Try to acquire more connections than the pool’s max limit and observe behavior.
   * Example:

java

CopyEdit

List<Connection> connections = new ArrayList<>();

for (int i = 0; i < MAX\_POOL\_SIZE + 10; i++) {

connections.add(threadPool.getConnection());

}

**How to Detect and Debug Connection Leaks**

1. **Monitor Database Open Connections**
   * Run a query to check the active DB connections:

sql

CopyEdit

SELECT \* FROM v$session WHERE status = 'ACTIVE'; -- Oracle

SELECT \* FROM pg\_stat\_activity; -- PostgreSQL

SHOW PROCESSLIST; -- MySQL

1. **Enable Connection Leak Detection in Pool**
   * If using **HikariCP**, set:

properties

CopyEdit

leakDetectionThreshold=5000 # Detects connections held for over 5 seconds

1. **Use a Java Profiler (JVisualVM, YourKit, etc.)**
   * Attach a profiler to the Java process and track java.sql.Connection instances.
2. **Enable Logging for Connection Checkout/Release**
   * Modify the thread pool to log getConnection(), release(), and newConnection() usage.
3. **Database-Level Connection Timeout**
   * Configure database connection timeout to automatically close unused connections:

properties

CopyEdit

wait\_timeout=300 # MySQL auto-closes connections after 300s

**Fixes and Best Practices**

* **Always Use try-with-resources**:

java

CopyEdit

try (Connection conn = threadPool.getConnection()) {

// Use connection

} // Auto-closes connection, ensuring release() is called

* **Validate and Clean Stale Connections Periodically**
* **Avoid Direct Use of newConnection() Unless Necessary**
* **Set a Maximum Lifetime for Connections in Pool**
* **Gracefully Handle Exceptions to Ensure Release**

By simulating these scenarios and using the detection techniques, you can identify and fix connection leaks in your application before they become production issues.

Extend teh Example on how to cleanup the session from DB to release or cleanup

**ChatGPT said:**

To properly **clean up database sessions** and **release leaked connections**, follow these strategies:

**1. Cleaning Up Leaked Connections in Java Code**

Ensure that all connections are properly closed when they are no longer needed. If the application fails to release them, use **try-with-resources** or explicitly close them in a finally block.

**Example: Proper Connection Cleanup Using Try-With-Resources**

This ensures that the connection is **always closed** even if an exception occurs:

java

CopyEdit

public void executeQuery() {

try (Connection conn = threadPool.getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM users")) {

while (rs.next()) {

System.out.println(rs.getString("username"));

}

} catch (SQLException e) {

e.printStackTrace();

}

} // Connection is automatically closed at the end of this block

**Example: Manual Cleanup in Finally Block**

If the connection needs to be explicitly managed:

java

CopyEdit

public void executeQuery() {

Connection conn = null;

try {

conn = threadPool.getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM users");

while (rs.next()) {

System.out.println(rs.getString("username"));

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

if (conn != null) {

threadPool.release(conn); // Ensure the connection is released to the pool

}

}

}

**2. Cleaning Up Stale Sessions from the Database**

If connections are leaked (not properly closed), they may **persist as idle or inactive sessions** in the database. Use the following database-specific queries to detect and terminate them.

**Oracle Database: Kill Inactive Sessions**

**Find Active Sessions**

sql

CopyEdit

SELECT sid, serial#, username, status, machine, program

FROM v$session

WHERE username IS NOT NULL;

**Kill a Stale Session**

sql

CopyEdit

ALTER SYSTEM KILL SESSION 'sid,serial#' IMMEDIATE;

Example:

sql

CopyEdit

ALTER SYSTEM KILL SESSION '123,456' IMMEDIATE;

**Force Disconnect Idle Connections After Timeout**

Modify the sqlnet.ora file:

ini

CopyEdit

SQLNET.EXPIRE\_TIME = 10 # Kills inactive connections after 10 minutes

**PostgreSQL: Terminate Stale Connections**

**Find Open Connections**

sql

CopyEdit

SELECT pid, usename, client\_addr, backend\_start, state

FROM pg\_stat\_activity

WHERE state = 'idle';

**Kill a Specific Connection**

sql

CopyEdit

SELECT pg\_terminate\_backend(12345); -- Replace 12345 with actual PID

**Disconnect All Idle Connections**

sql

CopyEdit

SELECT pg\_terminate\_backend(pid)

FROM pg\_stat\_activity

WHERE state = 'idle' AND backend\_start < now() - interval '30 minutes';

**MySQL: Close Idle Sessions**

**Find Open Connections**

sql

CopyEdit

SHOW PROCESSLIST;

**Kill a Connection**

sql

CopyEdit

KILL 12345; -- Replace 12345 with connection ID

**Set a Timeout for Inactive Sessions**

Modify MySQL settings:

sql

CopyEdit

SET GLOBAL wait\_timeout = 300; -- Auto-close idle connections after 5 minutes

In my.cnf:

ini

CopyEdit

[mysqld]

wait\_timeout = 300

**3. Automatically Clean Up Leaked Connections in Java**

Modify the connection pool configuration to detect and close leaked connections.

**HikariCP Connection Pool Settings**

properties

CopyEdit

leakDetectionThreshold=5000 # Detects leaks after 5 seconds

maximumPoolSize=10 # Limits connections to prevent excessive usage

idleTimeout=300000 # Closes connections after 5 minutes of inactivity

**C3P0 Connection Pool Settings**

properties

CopyEdit

c3p0.unreturnedConnectionTimeout=30 # Closes unreturned connections after 30s

c3p0.maxIdleTimeExcessConnections=60 # Kills excess idle connections after 60s

**4. Scheduled Cleanup Job for Stale Sessions**

Use a **background Java job** to detect and close leaked connections.

**Java Scheduled Cleanup Task**

java

CopyEdit

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.Timer;

import java.util.TimerTask;

public class ConnectionCleanupTask {

public static void main(String[] args) {

Timer timer = new Timer();

timer.schedule(new CleanupTask(), 0, 60000); // Run every minute

}

}

class CleanupTask extends TimerTask {

@Override

public void run() {

try (Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "user", "password");

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT sid, serial# FROM v$session WHERE status = 'INACTIVE'")) {

while (rs.next()) {

int sid = rs.getInt("sid");

int serial = rs.getInt("serial#");

stmt.executeUpdate("ALTER SYSTEM KILL SESSION '" + sid + "," + serial + "' IMMEDIATE");

System.out.println("Killed session: " + sid);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Summary of Best Practices**

✅ **Ensure release() is always called** in Java after using a connection.  
✅ **Use connection pool settings** to automatically clean up stale connections.  
✅ **Monitor open connections** in the database using SQL queries.  
✅ **Set a timeout for idle connections** to prevent long-running unused sessions.  
✅ **Use a scheduled cleanup job** to close leaked or inactive connections.