Connection Failures and Data Manipulation at Non-Thread Safe Shared JDBC Connection

Pitfalls of sharing a connection among threads

Here is a review of the potential pitfalls of sharing a single *Connection* among multiple threads.

- Committing or rolling back a transaction closes all open *ResultSet* objects and currently executing *Statements*, unless you are using held cursors.If one thread commits, it closes the *Statements* and *ResultSets* of all other threads using the same connection.
- Executing a *Statement* automatically closes any existing open *ResultSet* generated by an earlier execution of that *Statement*.If threads share *Statements*, one thread could close another's *ResultSet.*

In many cases, it is easier to assign each thread to a distinct *Connection*. If thread *A* does database work that is not transactionally related to thread *B*, assign them to different *Connections*. For example, if thread *A* is associated with a user input window that allows users to delete hotels and thread *B* is associated with a user window that allows users to view city information, assign those threads to different *Connections*. That way, when thread *A* commits, it does not affect any *ResultSets* or *Statements* of thread B.

Another strategy is to have one thread do queries and another thread do updates. Queries hold shared locks until the transaction commits in SERIALIZABLE isolation mode; use

READ_COMMITTED instead.

Yet another strategy is to have only one thread do database access. Have other threads get information from the database access thread.

Multiple threads are permitted to share a *Connection*, *Statement*, or *ResultSet*. However, the application programmer must ensure that one thread does not affect the behavior of the others.

Recommended Practices (at Oracle)

Here are some tips for avoiding unexpected behavior:

- Avoid sharing *Statements* (and their *ResultSets*) among threads.
- Each time a thread executes a *Statement*, it should process the results before relinquishing the *Connection*.
- Each time a thread accesses the *Connection*, it should consistently commit or not, depending on application protocol.
- Have one thread be the "managing" database *Connection* thread that should handle the higher-level tasks, such as establishing the *Connection*, committing, rolling back, changing *Connection* properties such as auto-commit, closing the *Connection*, shutting down the database (in an embedded environment), and so on.
- Close *ResultSets* and *Statements* that are no longer needed in order to release resources.
    - -> docs.oracle.com/javadb/10.8.3.0/devguide/cdevconcepts89498.html

Ref : oracle.com