Physical DB Connections vs Logical DB Connections

Before moving on to the details of Physical DB Connections, Logical/Pooled DB Connections, and their implementations, you may like to go through the concept of Connection Pooling. Read the article - Connection Pooling - What's it & Why needed?

Physical Database Connections

These types of DB connections are established either using the DriverManager (JDBC 1.0) class OR DataSource (JDBC 2.0) interface. In this case the connection is established to the DB directly at the time of use of the connection and as soon as the application finishes its interaction with the database, the connection object is destroyed and garbage collected.

Example: establishing a physical DB connection using DriverManager class (JDBC 1.0)

```
...
try {
Class.forName("driverClass");
} catch (ClassNotFoundException cnfe) {
System.err.println("Driver Class Not Found: " + cnfe.getMessage());
}
try {
Connection con = DriverManager.getConnection(dbURL, userID, password);
......
//...business logic implementation
con.close(); //... connection will be garbage collected
} catch (SQLException sqle) {
System.err.println("Exception while establishing DB Con: " + sqle.getMessage());
}
...
```

Logical/Pooled Database Connections

Logical or Pooled database connections are those connection objects which are created and maintained by the Connection Pool manager. The application requiring a connection to the database simply ask the Pool to provide them a connection object. The Pool either finds a free connection object or if all are exhausted then creates (of course, if the maximum limit of connection objects is not reached yet otherwise the request is queued) a new connection objects, registers it to the Pool and then gives that connection, which is finally used to serve the request. Similarly, when the request/application is done with the DB Connection then that is simply returned back to the Pool (and not destroyed or garbase collected as is the case with Physical DB Connections). The creation and destruction (garbae collection would probably be better term here) of all the connection objects of the Pool is maintained by the Pool Manager only and that's the main difference between the two. Otherwise, a logical connection is also internally a physical connection only, though it's an encapsulated one so that it can be managed by the Pool Manager.

Logical/Pooled DB connections are obtained using JDBC 2.0 Pooling Manager interfaces: javax.sql.ConnectionPoolDataSource and javax.sql.PooledConnection. ConnectionPoolDataSource works as a resource manager factory for the objects of type PooledConnection. The implementations of these interfaces are of course implementation dependent. All the J2EE compliant implementations just need to conform to the specifications and how they internally do is of little concern to the application developers.

Example: establishing a logical connection using JDBC 2.0 pooling manager interfaces

```
...
try {
Context context = new InitialContext(params);
ConnectionPoolDataSource cpds = (ConnectionPoolDataSource)
context.lookup(poolSource);
...
//... setting parameters...
PooledConnection pc = cpds.getPooledConnection();Connection con = pc.getConnection();
...
//... business logic implementation...
con.close(); //... connection will returned back to the Pool
} catch (SQLException sqle) {
System.err.println("Exception while establishing DB Con: " + sqle.getMessage());
}
```

Calling close() on a PooledConnection object

Notice that in case of logical connections, the 'con.close()' will only return the connection object back to the Connection Pool whereas the same method when executed on a physical database connection object will cause the connection object to be garbage collected. The reason for this different behavior of the close() method on two Connection objects (acquired through different sources) is that whenever the Pool Manager creates a PooledConnection object, it calls the addConnectionEventListener on that object to register a ConnectionEventListener object with every PooledConnection object. Where does this object come from? Actually the pool manager itself implements the interface ConnectionEventListener and registers itself with every PooledConnection object. This registration causes the pool manager to be notified whenever any event occurs on any

PooledConnection object. The close() call on a Connection object retrieved from a PooledConnection object (the Connection object acquired from a PooledConnection object is nothing but a handle to the actual PooledConnection object) simply notifies the pool manager that the application is done with the PooledConnection object and the pool manager simply deactivates the handle to that PooledConnection object and returns it back to the pool.

How & When a PooledConnection object actually gets destroyed?

A PooledConnection object gets destroyed only when the pool manager calls close() method on any such object because in that case close() method serves the normal process of destroying the object and not of just notifying some listener. Otherwise you need to develop a custom pool manager which doesn't really registers a PooledConnection object with a ConnectionEventListener object (i.e., the pool manager itself provided it implements this interface). In such a case the close() call will actually destroy the physical connection referenced by the PooledConnection object. But, this will obviously defeat the concept of Connection Pooling and therefore such an approach should be discouraged. It's better to go directly with physical connections in such a scenario.