# OrmLite DAO Factories And Connections Made Easy

By Roger Keays, 7 June 2014

OrmLite comes with some classes to help manage connections, but unfortunately they are **poorly named** and just plain confusing. For example, **OpenHelperManager** manages connections - and see if you can guess how that relates to **OrmLiteSqliteOpenHelper** or the **DaoManager**. This short blog gives you some simple code to make your API more intuitive and readable.

Connection management and singleton objects are never a simple combination. That's why we do things like abstraction and encapsulation. What I set out to do was have an API that looked like this:

```
CategoryDAO dao = CategoryDAO.connect(context);
List categories = dao.getCategories();
dao.release();
```

To get there it helps to know what is going on under the hood. So let's reverse engineer some of those oddly named OrmLite classes:

- **OrmLiteSqliteOpenHelper (Database Definition)**. Defines how to connect to the database and create and update the tables. This is where the real connection object lives.
- **OpenHelperManager (Connection Management)**. Keeps count of the number of times the database (connection) has been requested and released. When the count goes to zero it releases the database resources and when it goes back up it recreates them.
- **DaoManager (Singleton Management)**. Ensures that only one instance of a DAO is created for a given connection. Most apps will only connect to one database, so this effectively implements a singleton pattern. It is necessary because creating DAOs is expensive in Android due to the reflection involved.

Now, we can tie these objects together to implement the API we'd like. Here's an example:

```
public class CategoryDAO extends BaseDaoImpl<Category, Integer> {
  public CategoryDAO(ConnectionSource source) throws SQLException {
    super(source, Category.class);
  }

  /**
  * Static instantiation methods. The database connection is accessed via
  * the OpenHelperManager which keeps a count of the number of objects
  * using the connection. Thus every call to connect() must be matched by
  * a call to release() once the session is done.
  */
  public static CategoryDAO connect(Context context) {
    return with(OpenHelperManager.getHelper(context, Database.class)
        .getConnectionSource());
  }
  public static CategoryDAO with(ConnectionSource connection) {
    try {
      return (CategoryDAO) DaoManager.createDao(connection, Category.class);
    } catch (SQLException e) {
      throw new RuntimeException(e);
    }
  }

  /**
  * Releasing the DAO flags the connection manager that the DAO is no
  * longer using the connection. When the connection count is zero, the
  * connection manager will close the database.
  */
  public void release() {
    OpenHelperManager.release();
  }
}
```

Now our target code above will run, OrmLite will manage the singletons and the OpenHelperThingy will reuse connections across the DAOs. The alternative factory method **CategoryDAO.with (connection)** is useful for unit tests, or for using one DAO inside another like this:

```
  @Override
  public void update(Category category) {
    super.update(category);
```

```
    NoteDAO.with(connectionSource).updateColors(category);
  }
```

Here, we just reuse the same connection as the host DAO. For a unit test, you can set up the connection to the test database manually (or use Robolectric to mock the Android context).

Let me know if you think there is anything wrong with this pattern in the comments below. In my experience readable code goes a long way.

## About Roger Keays

Roger Keays is an artist, an engineer, and a student of life. He has no fixed addressand has leftfootprints on 40-something different countries around the world. Roger is addicted to surfing. His other interests are music, psychology, languages, the proper use of semicolons, and finding good food.