**Introduction to ActiveJDBC**



**What is ActiveJDBC?**

ActiveJDBC is a lightweight ORM and it focuses on simplifying the interaction with databases by removing the extra layer of typical persistence managers and focuses on the usage of SQL rather than creating a new query language.

Additionally, it provides its own way of writing unit tests for the database interaction through the DBSpec class.

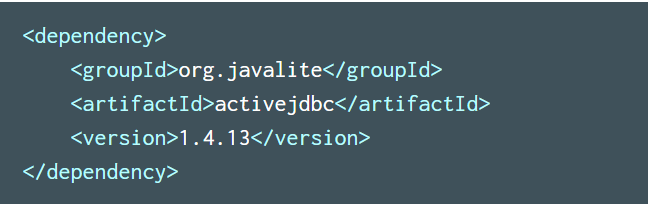
Let’s see how this library differs from other popular Java ORMs and how to use it

1. It infers the DB schema parameters from a database, so that we no need to mapping entities to underlying tables
2. No session No DB component management and no need to learn new query language
3. No need to write model class in server side
4. This implementation provide us test database, once execution done it will auto clean the schema

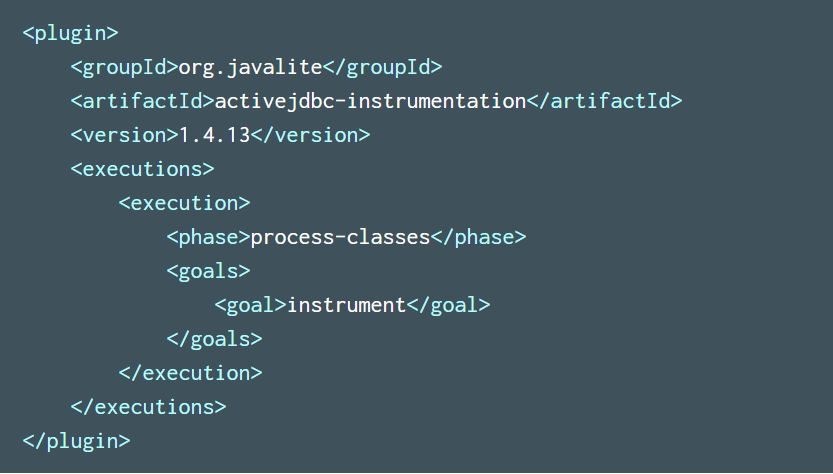
Let’s go with some Pre-requisite and environment setup to work with Active JDBC

**Setting up the Library**

First we need to add ActiveJDBC dependency



Add ActiveJDBC Instrumentation plugin in pom.xml



We should use latest version of ActiveJDBC i.e 2.0 version

What is the use of instrument plugin?

Instrumentation is byte code manipulation that happens after a compile phase. It adds static methods from super class to a subclass

Here super class is org.javalite.activejdbc.Model with which we are going to extends our Entity so that my Entity can achieve all features of javalite provided Model

Assume to fetch list of record based on condition we need to write SQL like below

Select \* from Table where field=?

Same way using Active JDBC instrumentation we can directly invoke static method like below

Assume my Entity class is Person

List<Person> retirees = Person.where ("age >=?", 65);

If you mark above syntax

Class Person Extends org.javalite.activejdbc.Model {}

So that using Person class we can call predefined static method like where () or order () etc. as Person inherited from org.javalite.activejdbc.Model

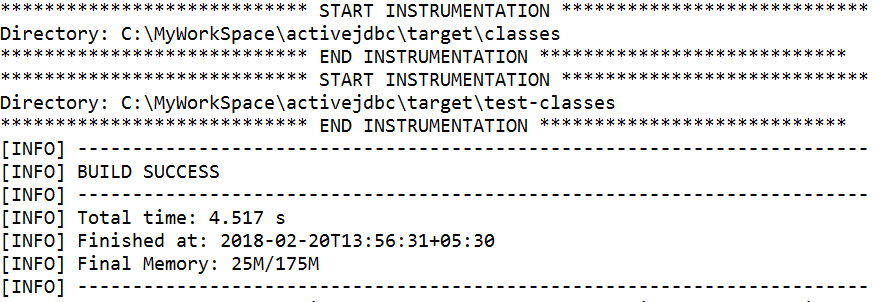
So we need to add this instrumentation plugin, without instrumentation, ActiveJDBC would not be able to know which class is being called, and as a result, what table to query

After add instrument plugin in pom.xml we need to run one build

mvn process-classes

mvn activejdbc-instrumentation: instrument

You can run any of one build, after build you will see in console that instrument start and end like below

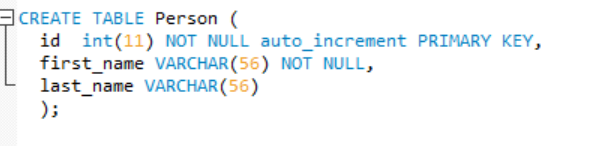


Now Getting started with small POC where we can find how to make DB connection and how we can do CRUD operation

**Development:**

Before develop application first we should create table manually as this ORM just abstraction above the JDBC implementation, where in JDBC we don’t have table auto creation features

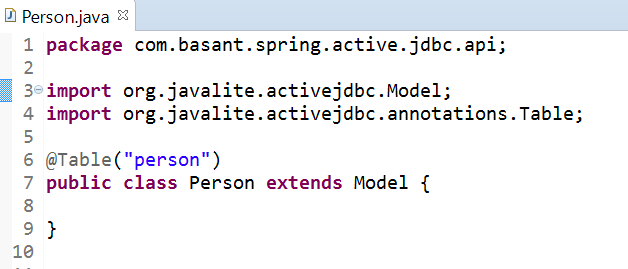
Create Table:



Create Model:

Let’s create Entity/Model class now

We can create a simple model with just one line of code – it involves extending the Model class



In Person Model class we no need to declare any attribute as we already defined attribute as a column while create Table

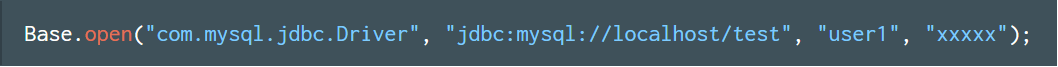
Connecting to a Database:

ActiveJDBC provided 2 class to connect with databases

1. Base
2. DB

**Base:**

If we are using one database then we should go for Base utility to create connection like below

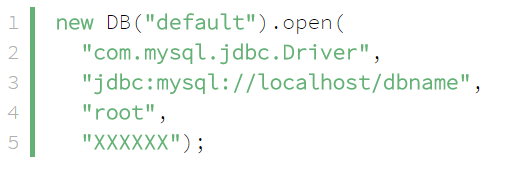


Once it open the connection object is then attached to the current thread, and can be consumed by any ActiveJDBC API.

**DB:**

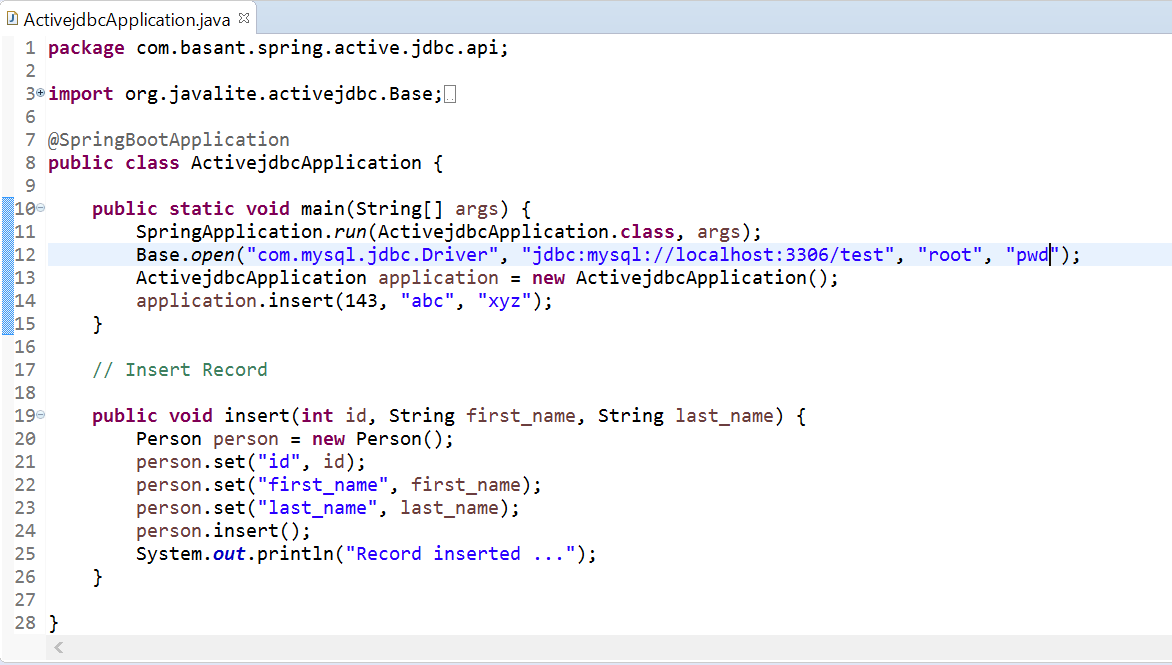
If we are using more than one database we should go for DB

Syntax to create connection



**CRUD Operation:**

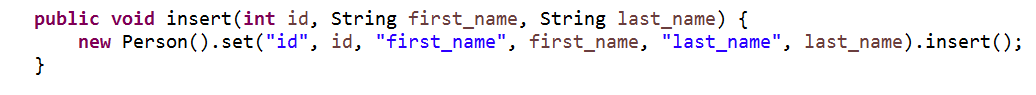
**1. Insert record:**



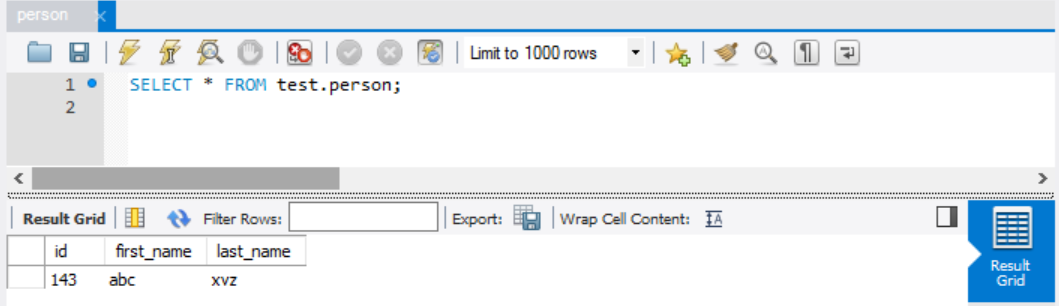
This is self-explanatory. Models are somewhat like maps. There are no setters or getters. You can still write them if you want

**Insert Record approach: 2**

The same logic can be written in one line as below

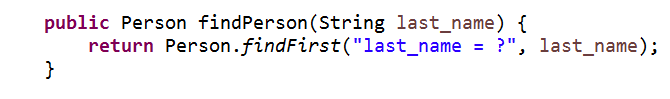
****

**Database:**

****

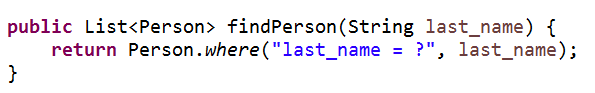
**2. Fetching a Record**

🡪Fetch single record based on last name

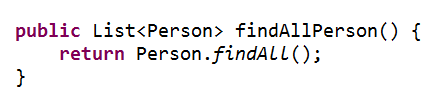


This line will find an instance of Person (conditionally), if one exists, or null if one does not exist

🡪Fetch records based on condition using inbuilt where clause



🡪Fetch all records



**3. Update Record**

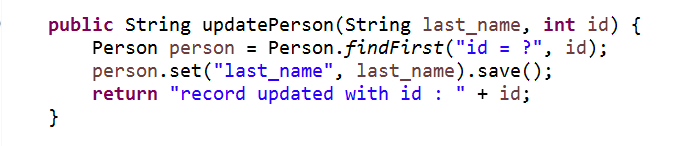
**Assume I want to update person last name based on id**

Update Person set last\_name=? where id=?

Steps:

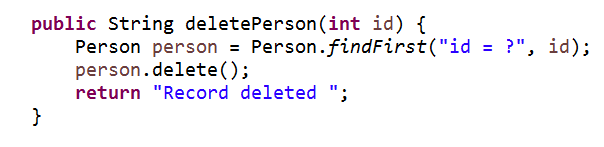
First I need to get that Person object based on ID

Then on same Person object need to set the last name and then update Person Object like below

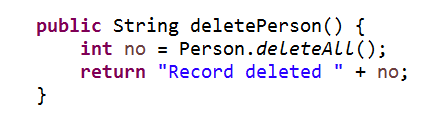


**4. Delete Record**

🡪Delete record based on field

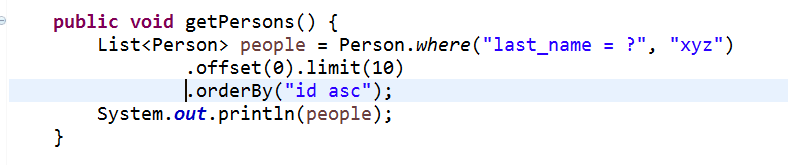


🡪Delete all records



**5. Pagination**

Fetch list of record based on last\_name and paginate it order by id



**Source Code:** [**Download**](https://github.com/Java-Gyan-Mantra/spring-activeJdbc)

**Thanks & Regards**

**Basant Hota**