Testing strategy for Data Access Objects

Testing database access classes is inherently hard, because it is hard to completely isolate the database classes as a single unit, since they rely heavily on the connection to an actual database. One might choose to mock the database and make it return a set of pre-specified values as a response to the queries in the program, but this sort of defeats the purpose, since there is no actual checking of the validity of the queries.

The approach we have chosen to testing database access is to create a sample database, filled with randomly generated entries, which are regenerated each time a test method is run. We have chosen to focus our attention on the PessimisticVoterDAO class. We have chosen this class, since it contains hand-built SQL queries i.e. text strings, which cannot be validated by the compiler or through any other form of testing than sending them to the database. Furthermore, this class is vital to our requirements about voter registration (req. no. 2), especially with regards to concurrency, and access to the database from multiple clients. We have written unit tests that aim to ensure that a high amount of the code in the class is covered but also to ensure that the class works conceptually right, in that it blocks access to the database in the right places. Another reason to focus most of our attention on testing this part of the database classes is that we have written this part of the database access by hand, whereas the other classes use the LINQ to entity framework, a framework which has already been thoroughly tested. Testing scenarios for manual DAO testing are described in the [DVL_event_scenario_DAO.bon] file.