Departamento de Computação

Alíria de Santana de Amorim Cruz

Atividade 1 – Teste de Software

São Cristóvão, SE
2024

1. Descrição do problema

O projeto consiste em aplicar uma solução dada para uma das perguntas do site Stack Overflow relacionadas a Teste de software. As strings de busca utilizadas, conforme solicitado pelo professor, foram “[unit-testing] or [junit] or [pytest]”. O objetivo é escolher uma das perguntas e aplicar a resposta dada a ela pelos colaboradores da plataforma.

2. Pergunta selecionada

- Título: What's the best strategy for unit-testing database-driven applications? [Qual a melhor estratégia para testes unitários em aplicativos baseados em bancos de dados?]
- Número de votos: 412
- Quantidade de respostas: 7

I work with a lot of web applications that are driven by databases of varying complexity on the backend. Typically, there's an ORM layer separate from the business and presentation logic. This makes unit-testing the business logic fairly straightforward; things can be implemented in discrete modules and any data needed for the test can be faked through object mocking.

But testing the ORM and database itself has always been fraught with problems and compromises.

Over the years, I have tried a few strategies, none of which completely satisfied me.

- Load a test database with known data. Run tests against the ORM and confirm that the right data comes back. The disadvantage here is that your test DB has to keep up with any schema changes in the application database, and might get out of sync. It also relies on artificial data, and may not expose bugs that occur due to stupid user input. Finally, if the test database is small, it won't reveal inefficiencies like a missing index. (OK, that last one isn't really what unit testing should be used for, but it doesn't hurt.)
- Load a copy of the production database and test against that. The problem here is that you may have no idea what's in the production DB at any given time; your tests may need to be rewritten if data changes over time.

Some people have pointed out that both of these strategies rely on specific data, and a unit test should test only functionality. To that end, I've seen suggested:

- Use a mock database server, and check only that the ORM is sending the correct queries in response to a given method call.

What strategies have you used for testing database-driven applications, if any? What has worked the best for you?

[Eu trabalho com muitos aplicativos da web que são controlados por bancos de dados de complexidade variada no backend. Normalmente, há uma camada ORM separada da lógica de negócios e apresentação. Isso torna o teste unitário da lógica de negócios bastante direto; as coisas podem ser implementadas em módulos discretos e quaisquer dados necessários para o teste podem ser simulados por meio de simulação de objetos.

Mas testar o ORM e o banco de dados em si sempre foi repleto de problemas e comprometimentos. Ao longo dos anos, tentei algumas estratégias, nenhuma das quais me satisfez completamente.

• Carregar um banco de dados de teste com dados conhecidos. Execute testes no ORM e confirme se os dados corretos retornam. A desvantagem aqui é que seu banco de dados de teste precisa acompanhar quaisquer alterações de esquema no banco de dados do aplicativo e pode ficar fora de sincronia. Ele também depende de dados artificiais e pode não expor bugs que ocorrem devido à entrada estúpida do usuário. Finalmente, se o banco de dados de teste for pequeno, ele não revelará ineficiências como um índice ausente. (OK, esse último não é realmente para o que o teste de unidade deve ser usado, mas não faz mal.)
• Carregar uma cópia do banco de dados de produção e teste contra ela. O problema aqui é que você pode não ter ideia do que está no banco de dados de produção em um dado momento; seus testes podem precisar ser reescritos se os dados mudarem ao longo do tempo. Algumas pessoas apontaram que ambas as estratégias dependem de dados específicos, e um teste de unidade deve testar apenas a funcionalidade. Para esse fim, vi sugestões:
• Use um servidor de banco de dados simulado e verifique apenas se o ORM está enviando as consultas corretas em resposta a uma determinada chamada de método.

Quais estratégias vocês usam para testar aplicativos orientados a banco de dados, se houver? O que funcionou melhor para vocês?]

3. Resposta escolhida

I've actually used your first approach with quite some success, but in slightly different ways that I think would solve some of your problems:

1. Keep the entire schema and scripts for creating it in source control so that anyone can create the current database schema after a check out. In addition, keep sample data in data files that get loaded by part of the build process. As you discover data that causes errors, add it to your sample data to check that errors don't re-emerge.

2. Use a continuous integration server to build the database schema, load the sample data, and run tests. This is how we keep our test database in sync (rebuilding it at every test run). Though this requires that the CI server have access and ownership of its own dedicated database instance, I say that having our db schema built 3 times a day has dramatically helped find errors that probably would not have been found until just before delivery (if not later). I can't say that I rebuild the schema before every commit. Does anybody? With this approach you won't have to (well maybe we should, but it's not a big deal if someone forgets).

3. For my group, user input is done at the application level (not db) so this is tested via standard unit tests.

Loading Production Database Copy:
This was the approach that was used at my last job. It was a huge pain because of a couple of issues:

1. The copy would get out of date from the production version
2. Changes would be made to the copy's schema and wouldn't get propagated to the production systems. At this point we'd have diverging schemas. Not fun.

Mocking Database Server:
We also do this at my current job. After every commit we execute unit tests against the application code that have mock db accessors injected. Then three times a day we execute the full db build described above. I definitely recommend both approaches.

[Na verdade, usei sua primeira abordagem com bastante sucesso, mas de maneiras ligeiramente diferentes que acho que resolveriam alguns dos seus problemas:

1. Mantenha todo o esquema e os scripts para criá-lo no controle de origem para que qualquer pessoa possa criar o esquema de banco de dados atual após um check-out. Além disso, mantenha os dados de amostra em arquivos de dados que são carregados por parte do processo de construção. Conforme você descobre dados que causam erros, adicione-os aos seus dados de amostra para verificar se os erros não reaparecem.

2. Use um servidor de integração contínua para construir o esquema do banco de dados, carregar os dados de amostra e executar testes. É assim que mantemos nosso banco de dados de teste sincronizado (reconstruindo-o a cada execução de teste). Embora isso exija que o servidor de CI tenha acesso e propriedade de sua própria instância de banco de dados dedicada, digo que ter nosso esquema de banco de dados construído 3 vezes por dia ajudou drasticamente a encontrar erros que provavelmente não teriam sido encontrados até pouco antes da entrega (se não mais tarde). Não posso dizer que reconstruo o esquema antes de cada confirmação. Alguém o faz? Com essa abordagem, você não precisará (bem, talvez devêssemos, mas não é um grande problema se alguém esquecer).

3. Para meu grupo, a entrada do usuário é feita no nível do aplicativo (não do banco de dados), então isso é testado por meio de testes de unidade padrão.

Carregando cópia do banco de dados de produção: Essa foi a abordagem usada no meu último trabalho. Foi uma grande dor de cabeça por causa de alguns problemas:

1. A cópia ficaria desatualizada da versão de produção

2. As alterações seriam feitas no esquema da cópia e não seriam propagadas para os sistemas de produção. Nesse ponto, teríamos esquemas divergentes. Não é divertido.

Servidor de banco de dados simulado: Também fazemos isso no meu trabalho atual. Após cada confirmação, executamos testes de unidade no código do aplicativo que tem acessadores de banco de dados simulados injetados. Então, três vezes ao dia, executamos a construção completa do banco de dados descrita acima. Definitivamente, recomendo as duas abordagens.]

4. Critério de escolha da respostas

No Stack Overflow a resposta foi escolhida como correta pelo autor da pergunta.