What best practices do you use for testing database queries?

Asked 16 years, 1 month ago Modified 6 years, 4 months ago

Viewed 14k times

13

I'm currently in the process of testing our solution that has the whole "gamut" of layers: UI, Middle, and the omnipresent Database.

Before my arrival on my current team, query testing was done by the testers manually crafting queries that would theoretically return a result set that the stored procedure should return based on various relevancy rules, sorting, what have you.

This had the side effect of bugs being filed against the tester's query more often than against the actual query in question.

I proposed actually working with a known result set that you could just infer how it should return since you control the data present -- previously, data was pulled from production, sanitized, and then populated in our test databases.

People were still insistent on creating their own queries to test what the developers have created. I suspect that many still are. I have it in my mind that this isn't ideal at all, and just increases our testing footprint needlessly.

So, I'm curious, which practices do you use to test scenarios like this, and what would be considered ideal for the best end-to-end coverage you can get, without introducing chaotic data?

The issue I have is where's the best place to do what testing. Do I just poke the service directly, and compare that dataset to that which I can pull from the stored procedure? I have a rough idea, and have been successful enough so far, but I feel like we're still missing something important here, so I'm looking to the community to see if they have any valuable insights that might help formulate my testing approach better.

database tdd integration-testing end-to-end

Share

Improve this question

Follow

asked Nov 3, 2008 at 23:36

Steven Raybell

523 ● 2 ● 6 ● 14

I'm going to go ahead and add the comment that the target environment is actually all MS technology based (SQL, IIS, .NET), for better or for worse. But, I still appreciate the mention of tools that use things like Python, despite my lack of skill there. – Steven Raybell Nov 4, 2008 at 4:12

9 Answers

Sorted by: Highest score (default)

3

Testing stored procs will require that each person who tests has a separate instance of the db. This is a requirement. If you share environments you won't be able to rely upon the results of your test. They'll be worthless.

You will also need to ensure that you roll back the db to it's previous state after every test so as to make the results predictable and stable. Because of this need to roll back the state after every test these tests will take a lot longer to complete than standard unit tests so they'll probably be something you want to run over night.

There are a few tools out there to help you with this. DbUnit is one of them and I also believe Microsoft had a tool Visual Studio for Database Professionals that contained some support for DB testing.

Share Improve this answer Follow

answered Nov 3, 2008 at 23:42

community wiki
Justin Bozonier

3

Here are some guidelines:

1. Use an isolated database for unit testing (e.g. No other test runs or activity)
2. Always insert all the test data you intend to query within the same test

3. Write the tests to randomly create different volumes of data e.g. random number of inserts say between 1 and 10 rows
4. Randomize the data e.g. for a boolean field random insert and true or false
5. Keep a count in the test of the variables (e.g. number of rows, number of trues)
6. For the Asserts execute query and compare against local test variables
7. Use Enterprises Services transactions to rollback database to previous state

See the link below for the Enterprises Services Transaction technique:

http://weblogs.asp.net/rosherove/articles/DbUnitTesting.aspx

Share Improve this answer Follow

answered Nov 3, 2008 at 23:55

user32378

293 ● 1 ● 3

1 1-10 rows isn't going to test much. And especially not performance on 10M-1B rows. – Jonathan Leffler Nov 4, 2008 at 2:32

1

As part of our continuous integration, we run our nightly 'build' of the database queries. This involves a suite of DB calls which are updated regularly from the real calls in the code as well as any expected ad-hoc queries.

These calls are timed to ensure that:

1/ They don't take too long.

2/ They don't differ wildly (in a bad way) from the previous night.

In this way, we catch errant queries or DB changes quickly.

Share Improve this answer Follow

answered Nov 3, 2008 at 23:48

paxdiablo
880k ● 241 ● 1.6k ● 2k

These "suit of DB calls", are they clones of the real calls? And if so, do you maintain these along with any queries in code? – Steven Raybell Nov 3, 2008 at 23:51

We extract each real call from the code and do some regex magic to make a static query from it, then add this static call to the suite. We also store the source line so we can detect changes and redo it. – paxdiablo Nov 4, 2008 at 0:10

1

The query planner is your friend, especially in this case. It is always good practice to check to see that indexes are used when you expect them to be and that the query doesn't require extra work to be done. Even if you have stress tests included in your suite, it is still a good idea to catch expensive queries before your app starts grinding to a halt.

Share Improve this answer Follow

answered Nov 4, 2008 at 0:02

Dana the Sane
15.2k ● 8 ● 60 ● 81

1

We have a blank database set aside for each developer and tester.

When the tests are run - each test clears the database and loads the data it is expecting to use. This gives us a known state at all times.

We can then test several different scenarios on the same DB (one after the other) and we never stamp on another testers toes.

That covers testing the data access itself. For service testing we do much the same thing but we test the inside of the service only - we don't actually hit the service we create an instance of the service processing class and pass in everything we need. That way we are testing the code and not the infrastructure (message etc..)

Share Improve this answer Follow

answered Nov 4, 2008 at 0:38

Brody
2,074 ● 1 ● 18 ● 23

1

Django offers a database unit test capability. You can borrow their design ideas and reproduce it in other environments.

The Django folks offer a subclass of Python's standard unittest `TestCase` class that populates a database with a known fixture -- a known set of data rows.

In the case of Django (and Python) it's easiest to populate the database from a JSON data extract. Other file formats for the fixture can be used for other frameworks. For example, if you're working in Oracle, you might find CSV files easier to work with.

This `TestCase` subclass allows to write a typical-looking TestCase that exercises the database with the known data fixture.

Additionally, the Django test runner creates a temporary schema for test purposes. This is easy for Django because they have a complete object-relational management component that includes DDL creation. If you don't have this available, you'll still need the DDL script so you can create and dispose of a test schema for unittest purposes.

Share Improve this answer Follow

answered Nov 4, 2008 at 2:18

S.Lott
391k ● 82 ● 517 ● 788

1

SQLServerCentral has an article here (you may have to register but it is free and without strings) about a TSQL Unit Testing Framework called tsqlUnit. It is open source and follows along in the tradition of the xUnit framework.

It follows the SEAT TDD pattern:

Setup - prepare the test conditions by manipulating the objects, tables, and/or data

Exercise - invoke the production code

Assert - check that the actual result equals the expected result

Teardown - return everything back to the way it was before the test started. This is actually done by Rolling back a transaction, which keeps everything nice and tidy.

Although I have not used it, it looks promising and is certainly something I will be looking at in more detail.

The framework can be downloaded here.

Share Improve this answer Follow

answered Nov 4, 2008 at 22:43

jheppinstall
2,358 ● 4 ● 23 ● 27

0

I find it useful to test the SQL being sent down to the database rather than the result of querying the database.

Not that I don't do the later, but I find it much faster to test for that than having the database too much lifting.

Share Improve this answer

Follow

answered Nov 4, 2008 at 0:31

Allain Lalonde

93.2k ● 71 ● 189 ● 238

0

This is a heavy setup, but I recommend TDDing containers.

Upon running your test script, build a new container that has your database running inside, seed it with mock data, then run the queries and test against what was returned and whether or not the queries were successful.

This way you have control over how close to production your testing environment.

[ThoughtWorks]()

Share Improve this answer

Follow

answered Jul 31, 2018 at 22:28

steviesh

1,774 ● 6 ● 20 ● 33