Clean up your campsite!

bit.ly/CleanCamp

Refactoring Monolith Database
Stored Procedures

@RileyMajor

Riley Major
That Conference 2017
Day: Tue, Aug 8   **Time:** 10:30 AM   **Location:** Tamarind *(map)*
Level: 200   **Primary Category:** DataStorage   **Secondary Category:**
Tags: testing, SQL Server, SQL, Stored Procedures
Your campsite is a mess. There are "totally safe to burn" plastic wrappers melted in last night's ashes. The roasting sticks are coated with dirt-encrusted marshmallow goo. And some sort of animal went through the trash bag you left out. (Oops.) We get it. Move fast and break things. Just ship it! But what's left a giant stored procedure with cursors, temp tables, and mystery calculations. It's a big black box that nobody wants to touch. Let's fix that. We'll open the lid on an example monolith and do major surgery. What's left will perform better, be easier to understand, encourage code reuse, and be easier to test. You might even begin to like writing SQL. (This talk is geared to developers using Microsoft SQL Server, but many of its principles apply to any RDBMS.)
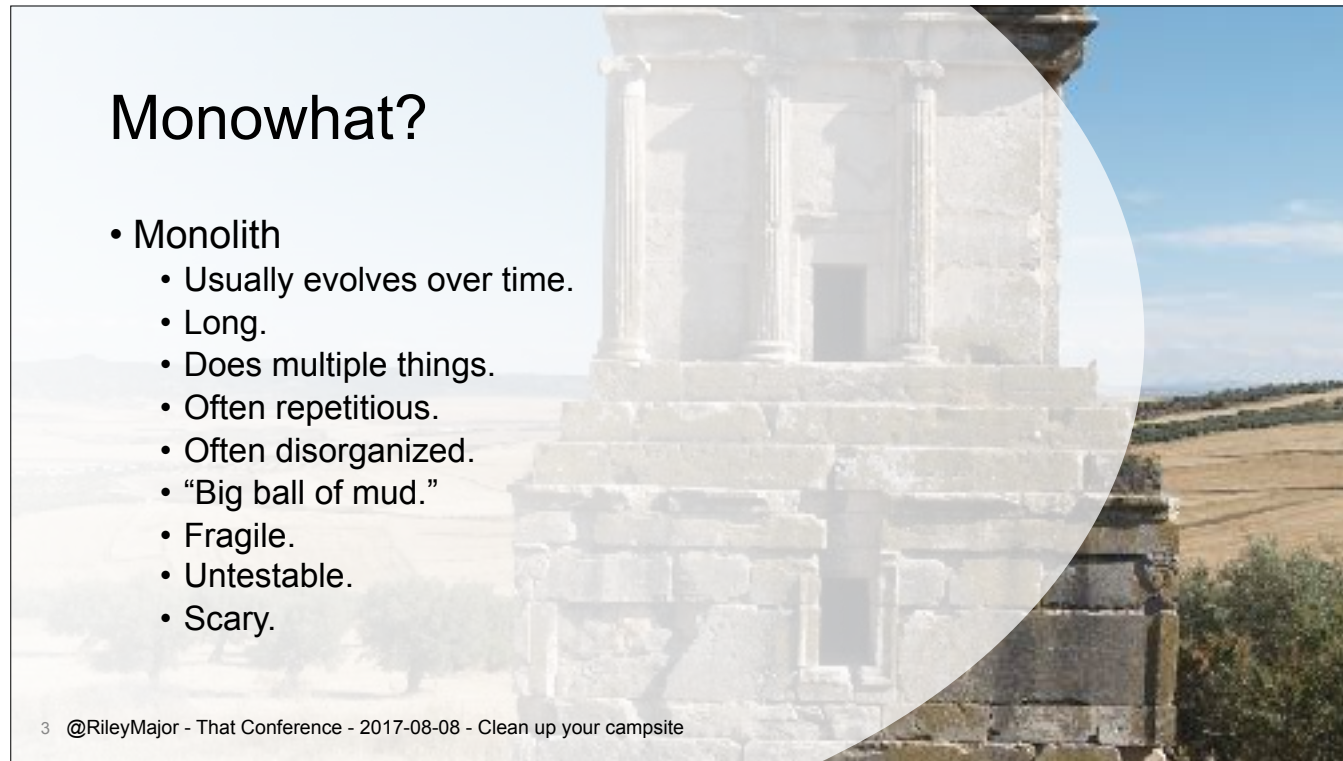
THANK YOU, THAT CONFERENCE SPONSORS!

Monowhat?

- Monolith
    - Usually evolves over time.
    - Long.
    - Does multiple things.
    - Often repetitious.
    - Often disorganized.
    - "Big ball of mud."
    - Fragile.
    - Untestable.
    - Scary.

Layers. Like an onion.

- Programming Layers
  - Presentation
  - Business Logic
  - Data Storage
- Tiers
  - Client Application (JS / HTML or Binary)
  - Server (PHP / NodeJS / ASP.NET)
  - Database (SQL Server, MySQL, NoSQL)
- "Best Practice"
  - Presentation in Client Application
  - Business Logic in Server

In "normal" programming, the widely accepted goal is to separate display logic, business logic, and data logic. You don't want your display logic mixed with your business logic because you will have multiple client applications (e.g. web, desktop app, mobile app). Business logic in the client means it's duplicated and every update requires redistribution. Display logic in the business logic means a complex mess which has to figure out how to behave based on which client app is accessing.

Conventional wisdom says not to put the business logic (and certainly not the display logic) in the database. There are good reasons.

Oh noes!

- Monolith
  - Presentation in Database!
  - Business Logic in Database!
- Bad
  - Database scaling is $$$hard$$$.
  - Causes vendor lock-in.
  - Database languages are primitive.
- But
  - Close to data. Less overhead.
  - Who really changes databases?
  - SQL more powerful than you think.

5 @RileyMajor - That Conference - 2017-08-08 - Clean up your campsite

http://www.vertabelo.com/blog/notes-from-the-lab/business-logic-in-the-database-yes-or-no-it-depends

Yes, database server licensing is expensive and you can't scale horizontally as easily, but maybe your ORM is can cause performance headaches too. Database servers are really good at some computational tasks (e.g. aggregating and unifying large amounts of data).

Either way, this talk presumes you already have a monolith, so it doesn't matter whether it's a good idea. What's done is done, at least for now.
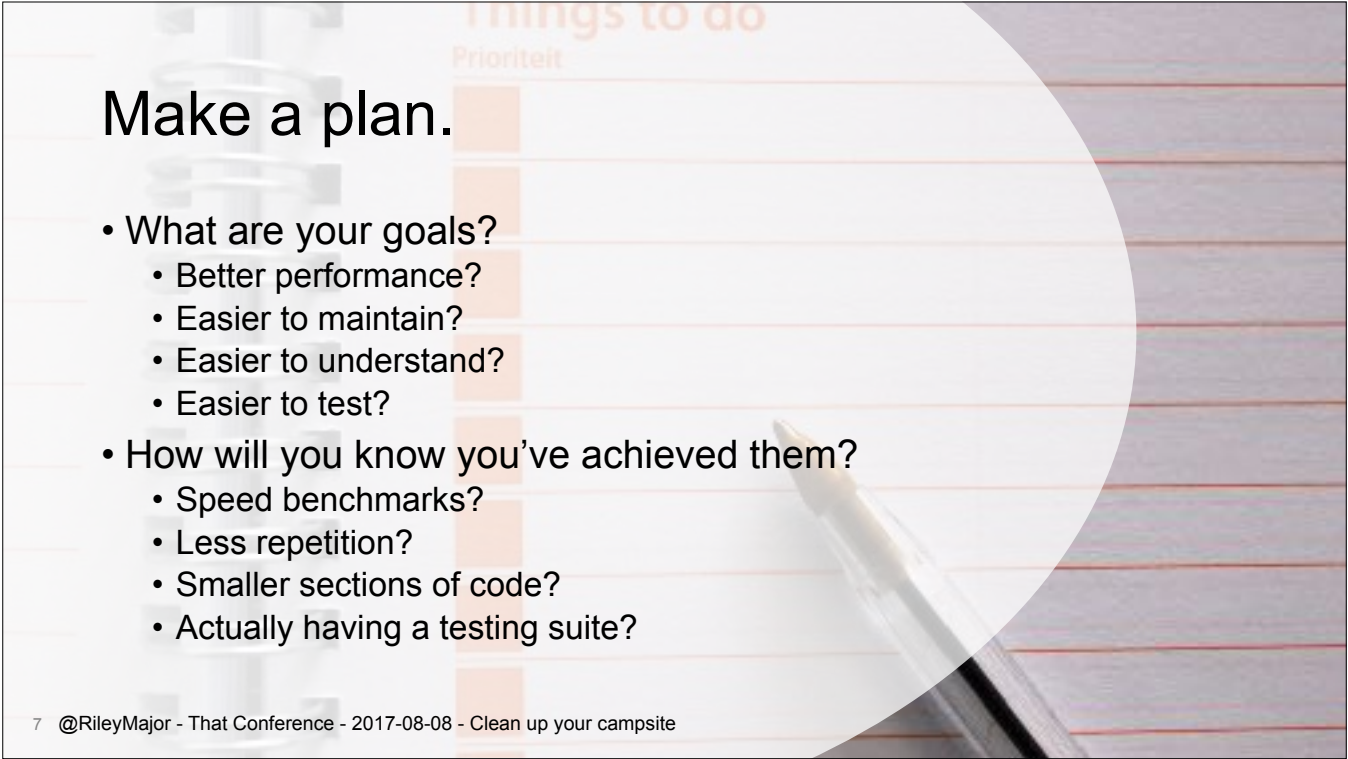
It's turtles all the way down.

- Separate layers even in the same tier.
    - Browser: MVVM.
    - Server: MVC.
    - Database: TBD.
- Database Layer
    - Presentation / Business Logic
        - IF / CASE / SET / SUM / + / DATEADD
    - Data Access
        - SELECT / UPDATE / INSERT / DELETE
- Testability, Isolation, and Portability.

Even within the same tier, there are good reasons to separate your layers. You should be able to manipulate the look of your application with HTML and CSS without wading through calls to the database server. And the code processing data input shouldn't be working about how to format a table.

There are benefits of separation at the database layer as well. The biggest is testability. You can comprehensively test individual bits of logic in isolation rather than having to test the whole process with just a few scenarios. Different people can work on different sections at the same time, as long as the interfaces between the separate pieces aren't changing. And if you dislike this stuff in the database so much, isolating the logic allows it to be more easily lifted to another tier.

Make a plan.

- What are your goals?
  - Better performance?
  - Easier to maintain?
  - Easier to understand?
  - Easier to test?
- How will you know you've achieved them?
  - Speed benchmarks?
  - Less repetition?
  - Smaller sections of code?
  - Actually having a testing suite?

Not only will answering these questions help guide your activities, they will help justify the project. Speed and reliability translate to less user disruption which generally translates to profit dollars.

Survey the damage.

- You can't avoid a thorough code review.
- Look for data modification.
  - INSERT, UPDATE, and DELETE.
  - Note columns affected.
- Look for external effects.
  - CLR
  - Email generation.
- Look for transaction handling.
  - BEGIN TRAN, COMMIT TRAN, ROLLBACK TRAN

You have to go down your rabbit holes. If your proc calls another proc, guess what, you're going to read through that other proc. And then the one it calls. You need to understand the breadth of the effects of the parent proc through all its children. And most importantly, you need to see any way the impacts can escape your SQL Server. You can contain what happens inside SQL, but you can't claw back anything which escapes.

Because you're primarily interested in what lasting effects the procedure has, you can largely skim any views or user-defined functions as they are naturally read-only.

CLR functions are a black box as they generally have access to do anything they want, including calling back and making changes to the database which will fall outside of the transaction.

If you have existing transaction handling, you have additional challenges. As you will see, we're going to use transactions to create a repeatable testing platform. Ideally, you could wrap most of your BEGIN TRAN statements in an "IF @@TRANCOUNT = 0" syntax to prevent them from occurring when you create your transaction. But that can still break things if you rely on nested transactions in your logic. Then again, if you do, are you sure you need to? Maybe now's the time to change that. They greatly increase complication.

Don't break anything

- Build a development environment.
  - You need to be able to play around.
  - You need realistic data (volume and content).
  - But maybe not real data.
- Work in isolation.
  - Were the changes from your process or another?
  - Is it slow because resources are used elsewhere?
- How can you tell if you broke something?
  - You need to capture a before and after state.
  - Aim for a deterministic process.

https://twitter.com/stahnma/status/634849376343429120

Deterministic

- Function always returns the same value for the same inputs.
- Easy to test: send the same values in before and after changes.

Yes	No	No	No
Return x + y	Return x + y + rand()	Return x + y + hour(now())	Return x + y + (SELECT TOP 1 z FROM Table)

@RileyMajor - That Conference - 2017-08-08 - Clean up your campsite

Play it again, Sam.

- Good
  - X -> Y
  - :: changes ::
  - X -> Y
- Bad
  - X -> Y
  - :: changes ::
  - X -> Z

Good luck with that.

- Real world not deterministic.
- Monoliths change state.
- Running second time not the same.
- ¯_(ツ)_/¯
  - X -> Y
  - :: changes ::
  - Y -> Q
- Need to go back in time.
- Transactions!

Your testing process won't be officially deterministic, as the tables could be modified in other ways in between runs; they aren't static. But practically speaking, if you're working in isolation, then a SELECT from a table in the first run will result in the same value as the next run.

Become a wrapper.

- To test impact of code changes, wrap your calls:
  - Begin transaction.
  - Run original code.
  - Capture changed data.
  - Rollback transaction (to revert data).
  - Run new code.
  - Capture changed data.
- Now compare the 2 captured data sets…

Oops

- Where did the changes go?
- Captured data is also rolled back!
- How can you save data which has been killed?

Build a Ghost House

- How capture doomed data?
  - Outside SQL Server? Hard.
  - Another thread with NOLOCK? Hard.
- What's immune from transactions?
- Variables!
- You can't have a variable for every row.
- One big XML? Ouch.
- Table variables survive transactions.
- They're ghost houses!

Rolling back a transaction doesn't clear out the variables set within it.

If the large volume of data modifications prevent a full capture, consider using aggregates, checksums, or hashes to detect changes.

Spooky Playground - Create House

```
DECLARE @Orders TABLE
    (
        CompareOrderID int,
        ColA_Orig int,
        ColA_AfterMonolith int,
        ColA_AfterChanges int
    );
```

@RileyMajor - That Conference - 2017-08-08 - Clean up your campsite

Spooky Playground - Fill House

```
INSERT INTO @Orders
(
    CompareOrderID, ColA_Orig
)
SELECT
    OrderID, ColA
FROM
    Orders;
```

Spooky Playground - Catch Ghosts

BEGIN TRAN; EXEC Monolith;

UPDATE @Orders
SET ColA_AfterMonolith = ColA
FROM Orders
WHERE OrderID = CompareOrderID;

ROLLBACK TRAN;

Spooky Playground - Again

```
BEGIN TRAN; EXEC Monolith_New;

UPDATE @Orders
SET ColA_AfterChanges = ColA
FROM Orders
WHERE OrderID = CompareOrderID;

ROLLBACK TRAN;
```

Spooky Playground - Compare

```
SELECT
    *

FROM
    @Orders
WHERE
    ColA_AfterMonolith <> ColA_AfterChanges;
```

@RileyMajor - That Conference - 2017-08-08 - Clean up your campsite

Mock your Black Boxes

- Transactions only work on the database.
- External effects aren't rolled back.
- Replace external calls with "mocks".
- They look and act like external calls.
- But you control the guts.
- Return hard-coded sample data.
- Have the mock log its inputs.

Make your time!

- Date/Time functions kill determinism.
- You have to control "now".
- Otherwise no two runs could be the same.
- So make your own time.
- And send it in as a parameter.

If your code behaves differently based on the current time, you won't know if differences in behavior are due to your changes or the ticking clock.
One of the first alterations to your monolith should be adding a @Now or @CurrentDateTime parameter. Then find every call to getdate(), sysdatetime(), etc. and replace it with that parameter.
You'll have to pass it all the way down your stack as well (procedures which call procedures which call functions, etc.).
Views are tricky. Could hard-code a date or convert to inline user-defined functions fed the time as a parameter.

Your petard should hoist.

- Move variable DECLAREs to the top.
- Reveals duplication.
- Reveals common data sources.
- Displays breadth of data required.
- Caution: DECLARE assignment.

"The scope of a variable lasts from the point it is declared until the end of the batch or stored procedure in which it is declared."
https://technet.microsoft.com/en-us/library/ms187953(v=sql.105).aspx

There's no such thing as a variable scoped to a loop in SQL Server.
And the declare statement doesn't reset the variable (though its default assignment does).
Therefore, there's little benefit in sprinkling declarations throughout the code.
Bring them all to the top as it exposes opportunities for consolidation and potential bugs (e.g. assuming reset in loop).

One SELECT to rule them all.

- Gather scattered SELECT statements at top.
- Reveals duplication.
- Prepares for separation.
- Prepares for shorter transactions.
- Use a single SELECT with fancy SQL if practical.

Sometimes, this results in such a complex SELECT statement that performance suffers due to optimizer problems (not finding a good plan before timeout due to the complex space) or execution engine limitations (e.g. re-evaluating string or XML parsing operations). Also, sometimes it's not able to short-circuit entire unnecessary sections as effectively as an IF statement could.

Measure twice, cut once.

- Find INSERT/UPDATE/DELETE.
- Replace with variables SETs.
- Move data modification to end of proc.
- Results in 3 main sections:
    - Data gathering.
    - Computation
    - Data modification.

25 @RileyMajor - That Conference - 2017-08-08 - Clean up your campsite

Find the various INSERT, UPDATE, and DELETE statements sprinkled about the code.

Replace them with assignments to placeholder variables.

Recreate them at the bottom of the procedure using the placeholder variables.

Again, this reveals opportunities for consolidation, which reduces code duplication and often performs better.

Also, depending on the isolation level, this can also result in less blocking, as updates are concentrated at the end after computations are complete.

Cases of CASES

- What's left? Logic.
- Lots of IFs, SETs, and calculations.
- Pull it all together in one giant statement.
- Usually performs better.
- Can be clearer.
- Can reduce code.
- Prepares for separation.
- CASE, Derived Tables, and CTEs are your friends.

What remains in the middle of the procedure should now be mostly calculations and control of flow logic.
Most IF statements and logic can be combined into a single– perhaps complicated—SELECT statement using fancy SQL.
Derived tables and common table expressions reduce code duplication and encourage a functional style which ports more easily to actual functions later.
Consolidate static values and variables at the top so you can see all required inputs.

Building Blocks

- Still one procedure = Still a monolith.
- Separate
    - Data Gathering -> Inline UDFs
    - Calculation -> Inline UDF.
- Allows data gathering re-use.
- Allows testing suite for business rules.
- Allows read-only monolith actions.

It's still a monolith so long as all of the code is in one procedure.
Separate the initial big SELECT statement to its own inline user-defined function (which is fed the current time).
That allows it to be used for other processes and be tested by itself (e.g. finding what rows will be effected by the process).
Separate the business logic into its own deterministic inline user-defined function (which is fed all the data it needs).
That allows a permanent testing framework for the business logic.
Separate hard-coded values to their own function or view. Allows reuse and eliminates typos using compiler enforcement.

It's all better now…

Monolith
- Usually evolves over time.
- Long.
- Does multiple things.
- Often repetitious.
- Often disorganized.
- "Big ball of mud."
- Fragile.
- Untestable.
- Scary.

Reformed monolith.
- Recently written.
- Short.
- Orchestrates multiple things.
- Repeated code eliminated.
- Organized into functions.
- Vials of reagents to mix.
- Problems isolated.
- Testable.
- Benign.

Riley Wheeler Major

ThatConf@RileyMajor.com
twitter.com/rileymajor
github.com/rileymajor
linkedin.com/in/rileymajor/

Enterprise Architect
Manna Freight Systems, Inc.

Image Credits

Title

Sam Howzit Follow
Campfire
https://www.flickr.com/photos/aloha75/8608867490
8608867490_303df542c0_k.jpg

Monolith

Institute for the Study of the Ancient World Follow
The Mausoleum of Ateban (III)
https://www.flickr.com/photos/isawnyu/5834281724/
5834281724_99511f0d3a_o.jpg

Layers

John Fowler
Layers
https://www.flickr.com/photos/snowpeak/12547422744/
12547422744_77bf3b1337_k.jpg

Oh Noes

Henry Burrows
Surprised?
https://www.flickr.com/photos/foilman/4998878794
4998878794_f554510315_o_crop.jpg

Turtles

andre/atroe
Stack of turtles
https://www.flickr.com/photos/30312936@N04
4645520534_cbd2dcbae7_o_crop.jpg

To-Do

Nikki Buitendijk
To do
https://www.flickr.com/photos/nikkibuitendijk/14670721802
14670721802_4a6e2b9333_o.jpg

Survey the Damage

Metropolitan Transportation Authority of the State of New York Follow
Fix&Fortify - Greenpoint Tubes
https://www.flickr.com/photos/mtaphotos/9504656210
9504656210_a7972b1039_o.jpg

Don't Break Anything

Stig Nygaard
Trekroner
https://www.flickr.com/photos/stignygaard/34673211491
34673211491_ecfb9db477_o.jpg

Play it again, Sam!

naql
Record
https://www.flickr.com/photos/naql/34691183380
34691183380_429a0c7120_k_crop.jpg

Good luck with that.

JD Hancock
Mad Man With A Box
https://www.flickr.com/photos/jdhancock/7995637778
7995637778_75df99c01e_o.jpg

Become a wrapper

Matt Reinbold
Hold All My Calls
https://www.flickr.com/photos/furryscalyman/1034889957
1034889957_de68410cfa_o_crop.jpg

Build a Ghost House

Jamiecat *
Amityville Haunted House
https://www.flickr.com/photos/jamiecat/4910864986
4910864986_b1f81032a3_o.jpg

Spooky Playground

Taber Andrew Bain
Spooky Playground Equipment Detail
https://www.flickr.com/photos/andrewbain/1542241604
1542241604_24f9cd70c7_o_crop.jpg

Mock your black boxes

r2hox
BLACK BOX - 6
https://www.flickr.com/photos/rh2ox/13890877727
13890877727_7d75107f1e_o_crop.jpg

Your Petard Should Hoist

Larry
Hoists
https://www.flickr.com/photos/kayarewhy/14646253804
14646253804_035cbbc5aa_o.jpg

One SELECT to rule them all

idreamlikecrazy
One Ring to rule them all
https://www.flickr.com/photos/purple-lover/13583362554
13583362554_33a81b110e_o_flipped.jpg

Measure Twice Cut Once

carrotmadman6
Scissors
https://www.flickr.com/photos/carrotmadman6/4732395647
4732395647_e62cbe959c_o.jpg

Cases of CASEs

Lisa Parker
crates
https://www.flickr.com/photos/lisa-parker/4787352735
4787352735_32b926d54c_o_cropped.jpg

Building Blocks

Eric Kilby
Augusta Block
https://www.flickr.com/photos/ekilby/18099902934
18099902934_a12d573729_k.jpg

All Better Now

Waldemar Merger
Green Hill
https://www.flickr.com/photos/paxx/9546367406
9546367406_bcbe43bfe7_o.jpg