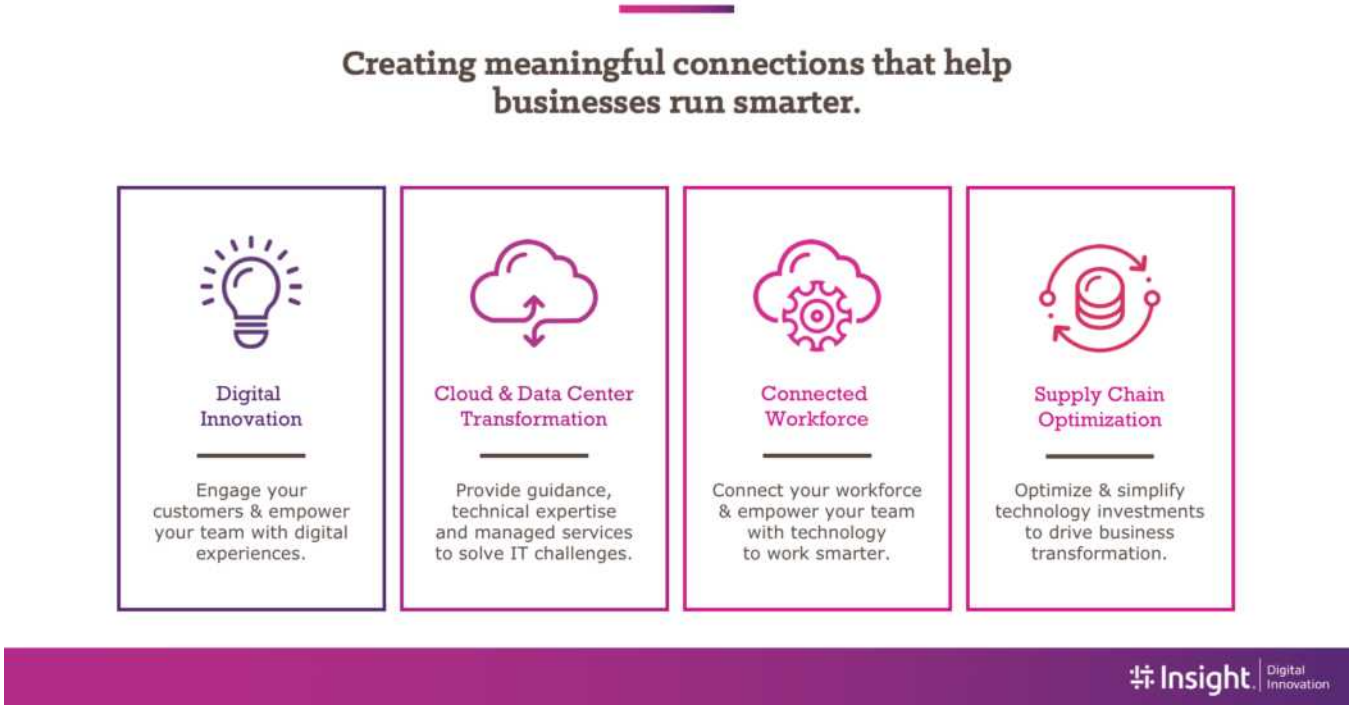

Good afternoon everyone –
Thanks for being here to learn
About SQL Server testing using TDD

My name is Jeff McKenzie,
And I am a Practice Manager for Applications and Infrastructure
At Insight Digital Innovation in Columbus Ohio

We used to be Cardinal Solutions
But acquired in August 2018 by Insight

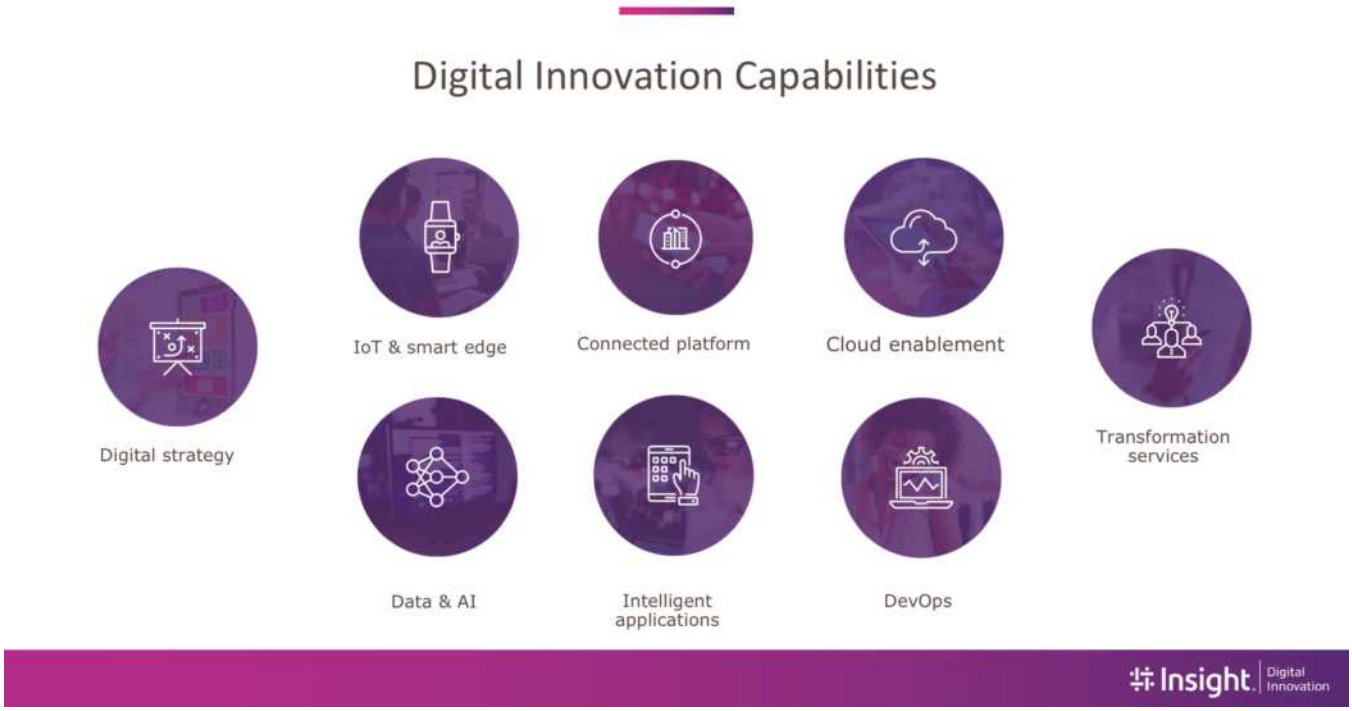



Insight is a global, fortune 500 company

-- does a lot of things in tech

But, acquired Cardinal to help expand their
Digital Innovation division

Digital innovation solves business problems using:
- Custom development
- With established as well as emergent technologies




- Do a lot of cloud work, app modernization
- Big data, predictive analytics
- Devops on both the Microsoft and open source side
- As well as a fair amount of IoT solutions

## Award winning technology

Our combined IT industry knowledge and technology expertise have earned us numerous Microsoft honors through the years.

Gold Microsoft Partner | Azure Expert MSP

WINNER Artificial Intelligence Global Partner of the Year

WINNER IoT Global Partner of the Year

WINNER Mobile App Development Global Partner of the Year

**2019**

*2019 Microsoft U.S. Intelligent Cloud Partner of the Year – App Innovation*

*2019 Microsoft U.S. Partner Choice Award Winner – Data and AI*

**2018**

*2018 Microsoft Worldwide Partner of the Year for Open Source on Azure*

*2018 Microsoft Worldwide Partner of the Year for Artificial Intelligence*

*2018 Microsoft Worldwide Financial Services Partner of the Year Finalist*

*2018 Microsoft U.S. Partner of the Year for Dev Ops*

*2018 Microsoft U.S. Partner of the Year for Internet of Things*

**2017**

*2017 Microsoft Worldwide Partner of the Year Winner for Mobile App Dev*

**2016**

*2016 Microsoft Worldwide Partner of the Year for Internet of Things*

Insight | Digital Innovation

As a worldwide Microsoft parter, Insight has received
a number of awards over the years,

Including: Iot Partner of the year in 2016
Mobile App Dev and Open Source Azure in 2017
Ai and Modern Workpace awards in 2018

Azure Expert MSP –
Completed a rigorous application process with Microsoft to verify successful completion
of projects across almost all Azure service offerings
We passed a 300-hour on-site audit by an independent third party
Also have more than 1,000 Azure-focused engineers and service professionals

https://azure.microsoft.com/en-us/partners/


Today we are going to learn about TDD –

How many of you have heard of TDD,
know what it means, or tried it? [hands]

How many use TDD
on a fairly regular basis? [hands]


So My goal for today is to show you

How to use TDD practices

Specifically in SQL Server,

Whether that's standalone SQL,

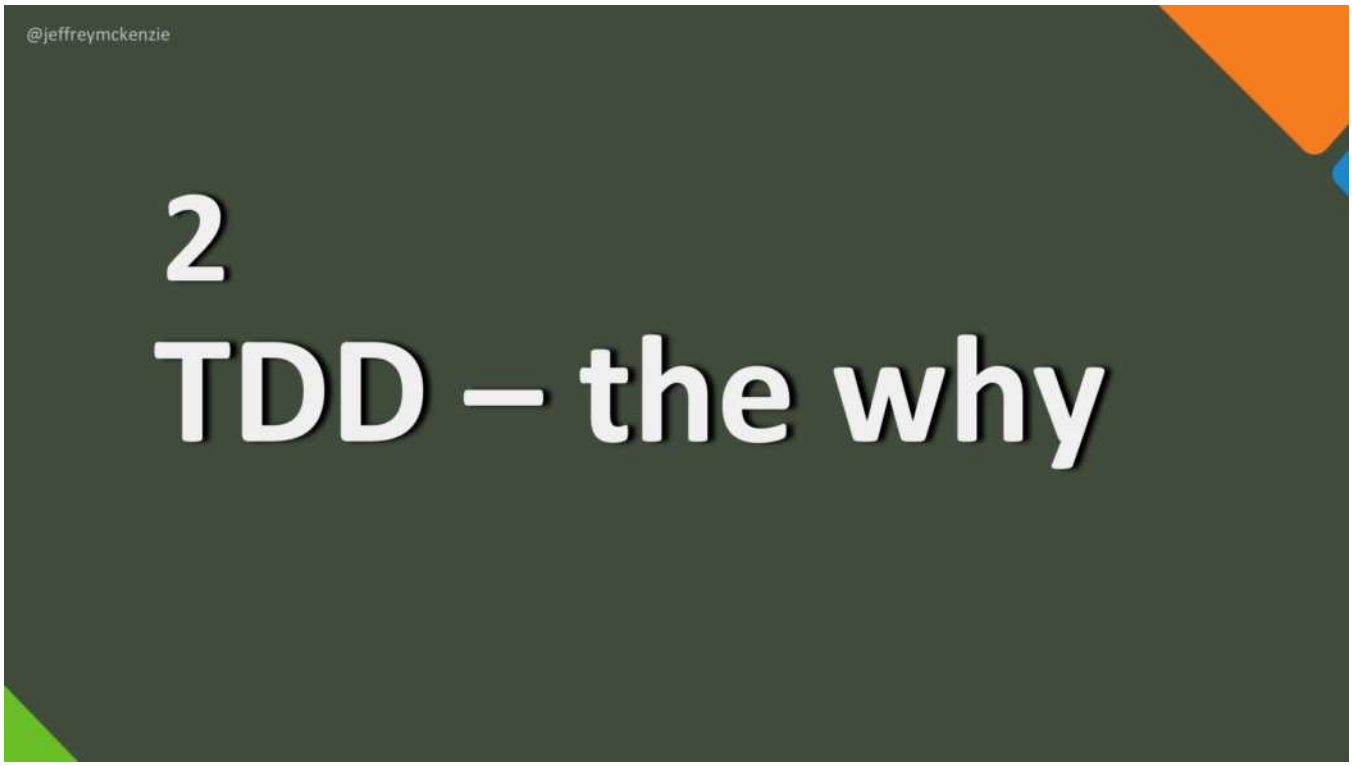
Or the data layer in a larger application. ***


To realize that goal, we are first going to

Define TDD, explain it, qualify it,

To make sure we are all on the same page

When we talk about what TDD is and is not. ***




A lot of us have heard about the practice of TDD

But maybe we don't hear as much about why.

So we will examine the benefits of TDD. ***


Finally we will take a look at a specific example

Of a real-life production scenario

And demonstrate how to use TDD

To write SQL Server code. ***


First, I'd like to get an idea of your technical background:
How many would consider yourselves
…application developers?
…DBAs or primarily SQL developers?
…QA or testing?
Any roles I missed? ***

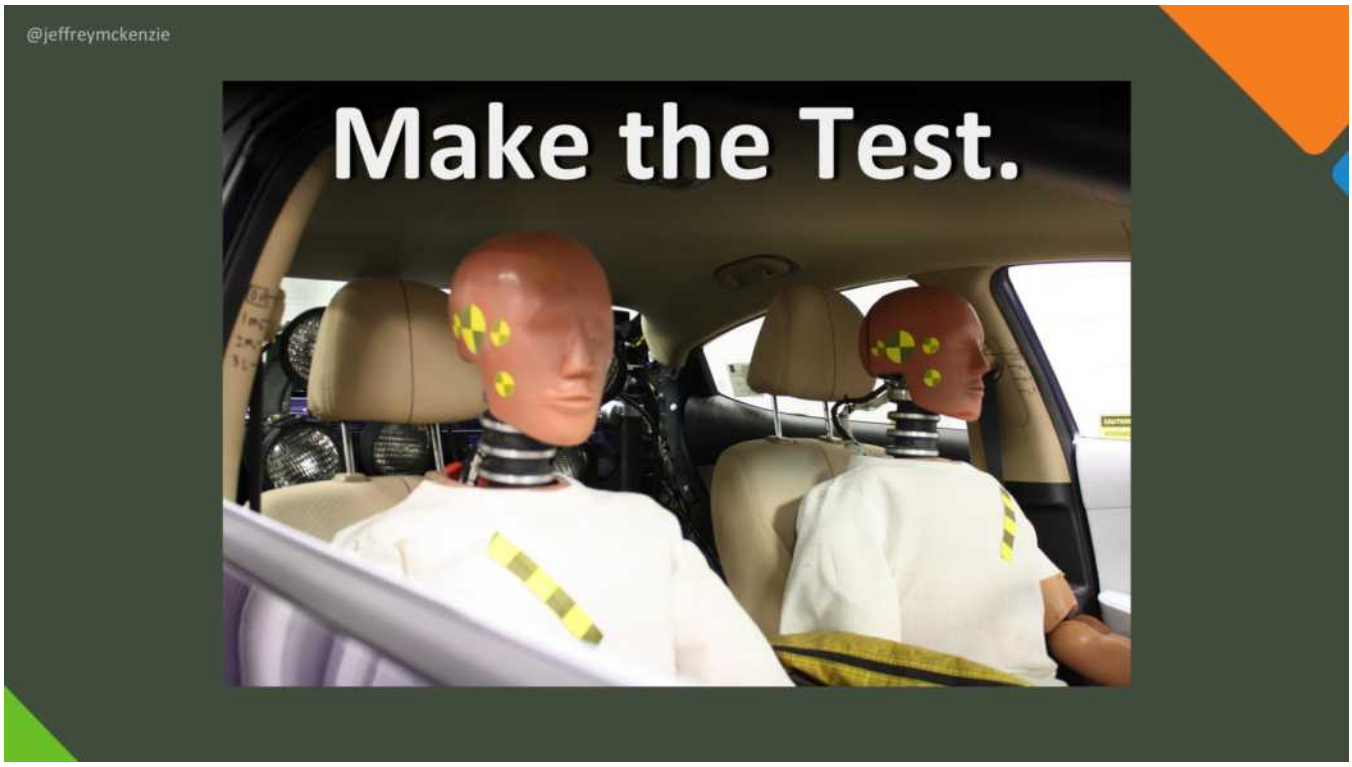
=========
https://upload.wikimedia.org/wikipedia/commons/4/42/Townshend_smashing_guitar.jpg
By Heinrich Klaffs [CC BY-SA 2.0 (https://creativecommons.org/licenses/by-sa/2.0)], via Wikimedia Commons


**Test Driven** because tests are written first, before any code.

With a traditional approach, tests are created after,

verify what's already been coded.

Three basic steps for TDD:
Make the test, make it fail, and make it work. ***




First step – hardest barrier when learning

Write the test first– before code

Get requirements, want to make something happen

Writing test doesn't feel like it

Important point about a test… ***

=========
https://upload.wikimedia.org/wikipedia/commons/b/b2/V08383P339.jpg
By Calspan Corporation, National Highway Traffic Safety Administration [Public domain], via Wikimedia Commons


Make it simple --  not do too much.

Easy to create and execute

The tests we are talking about here are unit tests.

[ASK]
How would you describe a unit test? [CLICK]***

=========
Picture of easy button, by me.

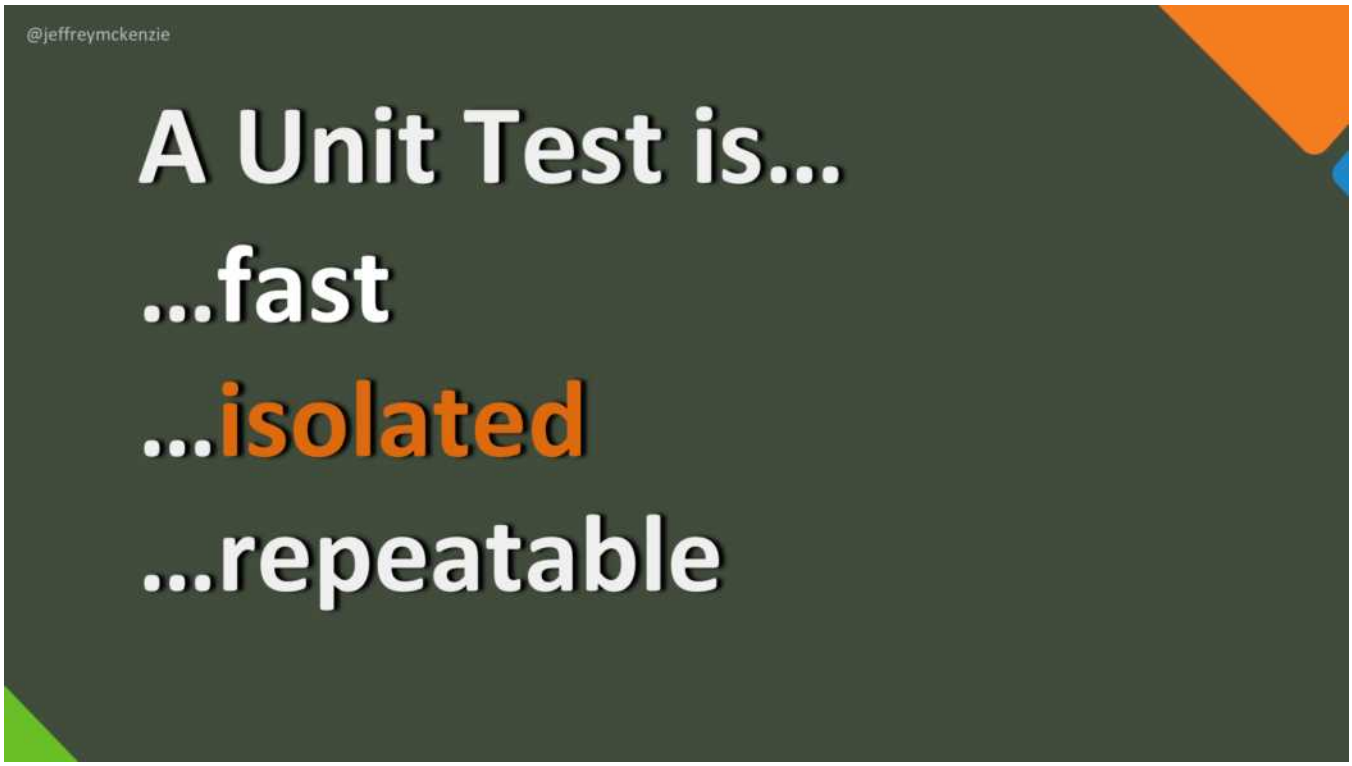

Next → -- Attributes of Unit test --


Fast.

Run tests while working,

see if broken, know when done

Simple = fast ***



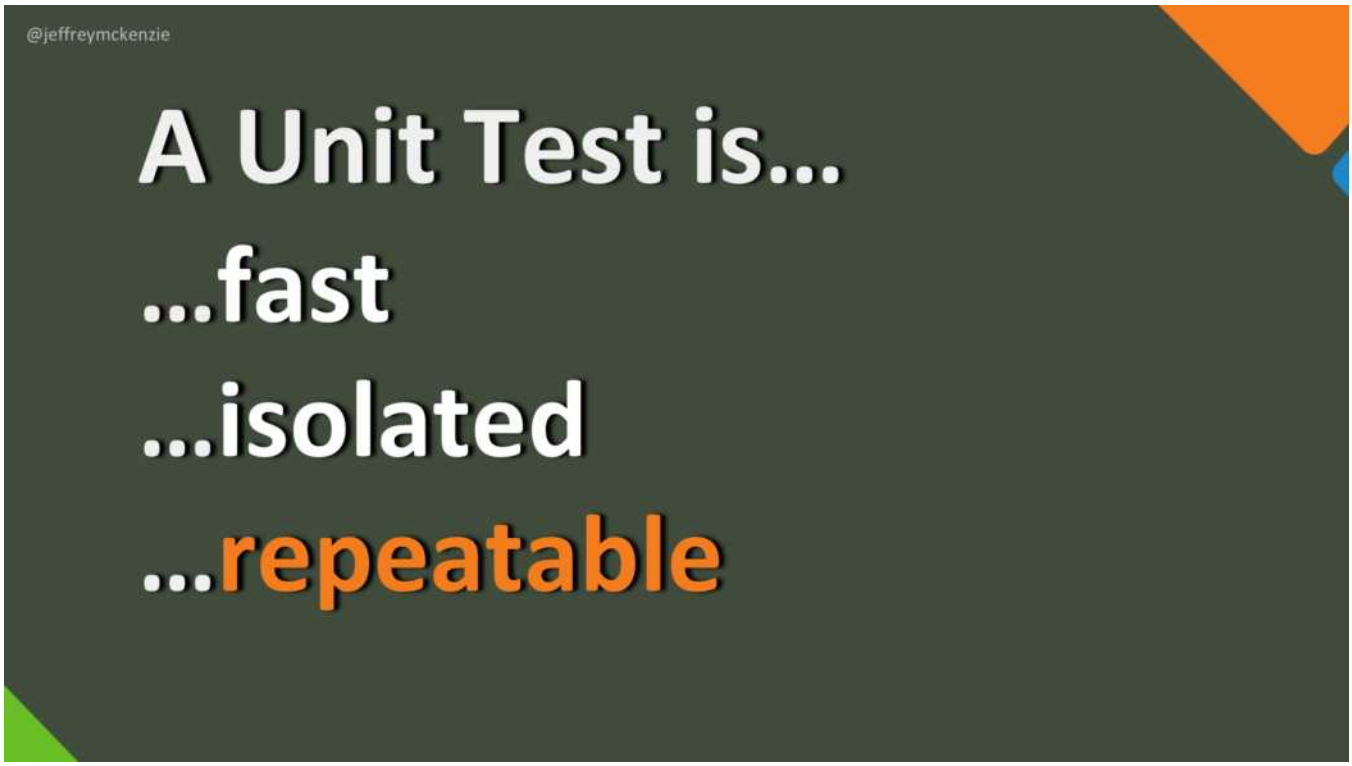

Also faster if isolated --

isolated = reduce/remove dependencies,

Other classes, methods

Isolation = testing right place,

If fails, know where to look. ***




Finally, repeatable

If code tested not changed, same result every time

Setup/teardown should be in test,

No manual work req btwn tests

Again, helps the test run quickly. ***


Try quick concrete example…

Easy button, Computer model/simulation

First requirement →

Create an easy button…

=========
Picture of easy button, by me.


With a click method,

Returning "That was easy." ***

@jeffreymckenzie

```
public void Test_EasyButton_Click() {
    //Arrange


    //Act

    //Assert

}
```

Start – call it Test Easy Button Click

triple A pattern → Arrange, Act, Assert.

Set up arrange first,

Everthing needed to execute code under test ***

@jeffreymckenzie

```
public void Test_EasyButton_Click() {
    //Arrange
    var button = new EasyButton.Button();
    String expected = "That was easy.";
    //Act

    //Assert

}
```

C#, same pattern for any language

Easy button instance

Variable called expect –

holds value we want to see***

@jeffreymckenzie

```
public void Test_EasyButton_Click() {
    //Arrange
    var button = new EasyButton.Button();
    String expected = "That was easy.";
    //Act
    String actual = button.Click();
    //Assert

}
```

Act section –

do actual code execution,

assign output of click method

To a variable called actual. ***

@jeffreymckenzie

```
public void Test_EasyButton_Click() {
    //Arrange
    var button = new EasyButton.Button();
    String expected = "That was easy.";
    //Act
    String actual = button.Click();
    //Assert
    Assert.AreEqual(actual, expected);
}
```

Assert –

a verification that action executed way we want

Here, assertion is output = that was easy

Simple test – just one thing ***


After simple test, make sure fails

If write passing by mistake,

Won’t know when you’re done,

Or what code supposed to do

=========
https://upload.wikimedia.org/wikipedia/commons/5/54/Hydrogen_balloon_explosion.jpg
By Maxim Bilovitskiy (Own work) [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons

@jeffreymckenzie

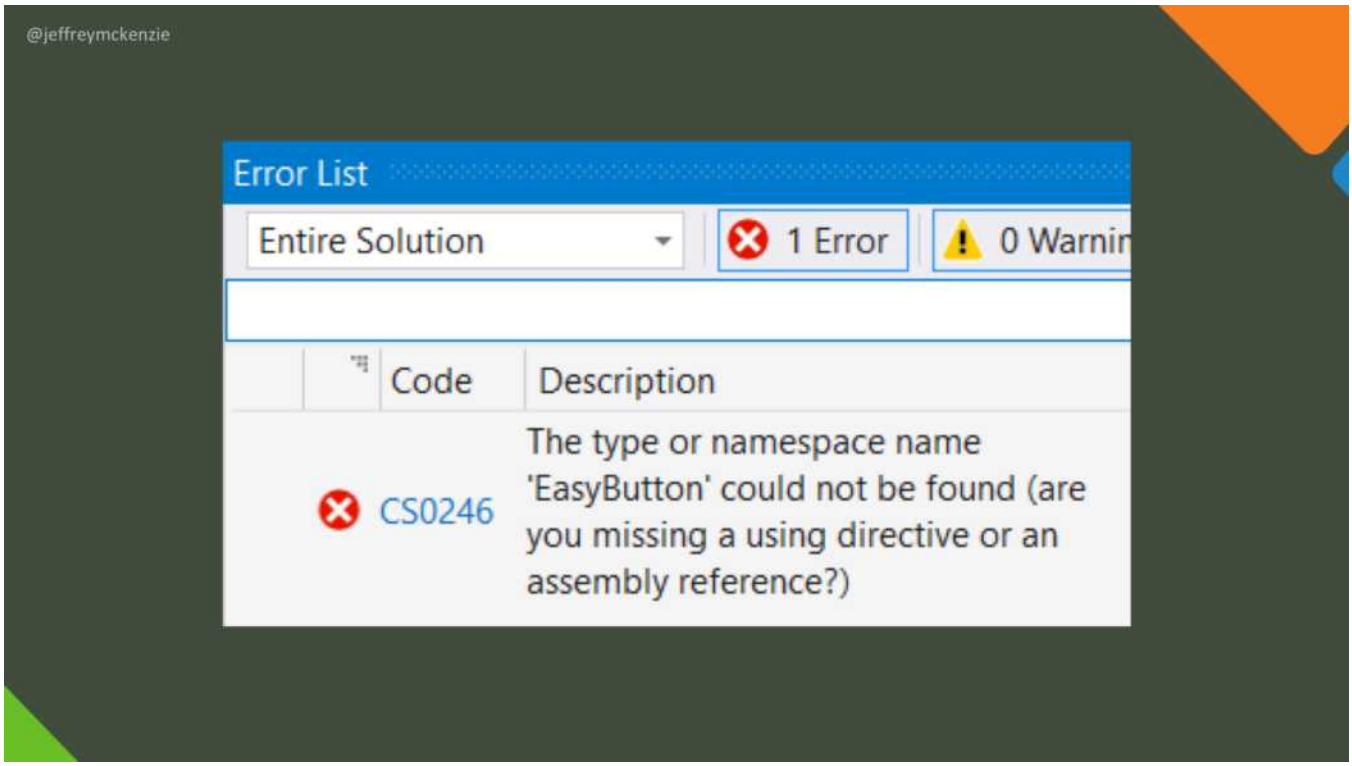
```
public void Test_EasyButton_Click() {
    //Arrange
    var button = new EasyButton.Button();
    String expected = "That was easy.";
    //Act
    String actual = button.Click();
    //Assert
    Assert.AreEqual(actual, expected);
}
```

Back to our test -- is it going to pass?

No – why not?

No code yet, no EasyButton

Attempt to run = ***




doesn't even compile.

Tells me that EasyButton doesn't exist,

Which is good because

We haven't made it yet.

Completed “Make it fail” step. ***


Next step → make it work.

trick = little code as possible.

Idea of TDD = satisfy all requirements
Least amount of work.

less code = less go wrong
Next → write the implementation ***

=========
https://upload.wikimedia.org/wikipedia/commons/c/c4/Hillary_Clinton_%2824007578223%29.jpgBy Gage Skidmore from Peoria, AZ, United States of America (Hillary Clinton) [CC BY-SA 2.0 (https://creativecommons.org/licenses/by-sa/2.0)], via Wikimedia Commons

@jeffreymckenzie

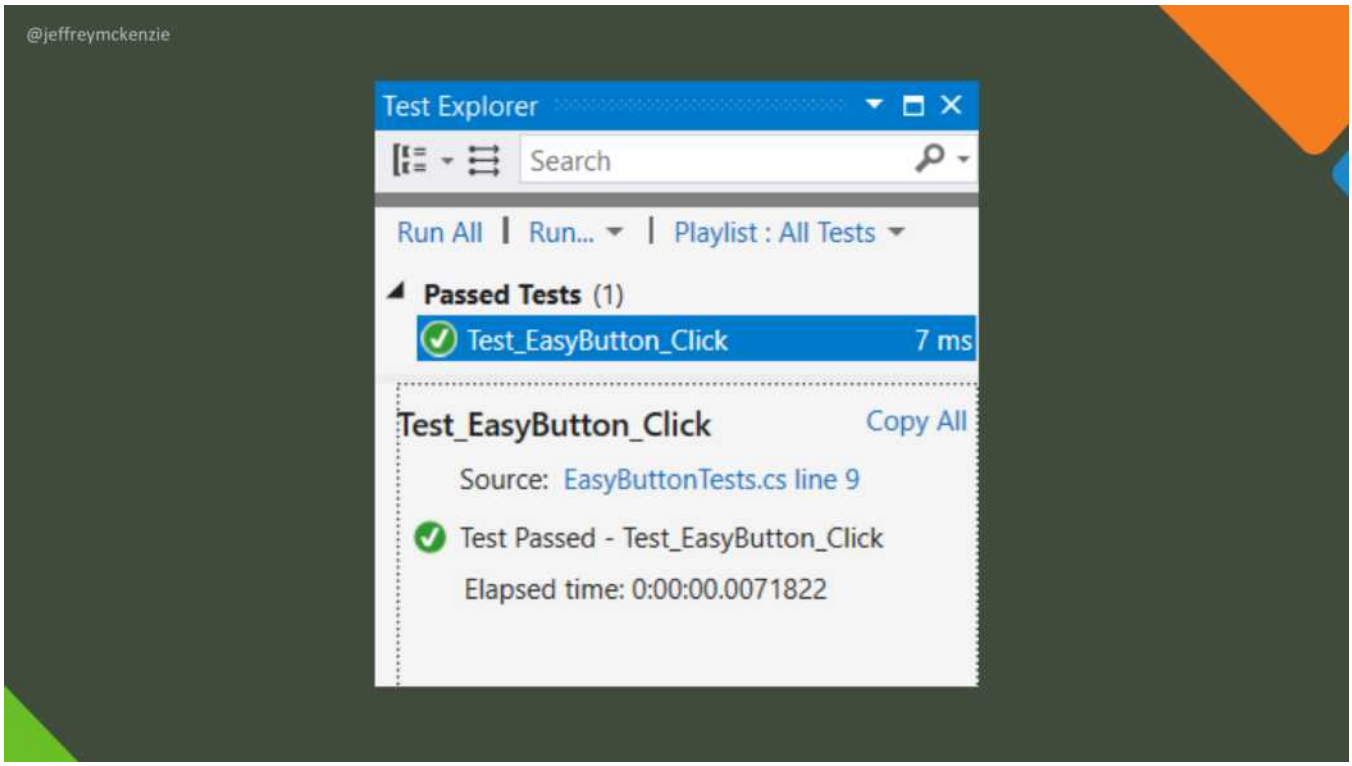
```
public class Button {
    public String Click() {
        return "That was easy.";
    }
}
```

Pretty simple –

create the class…

Create the method…

return the output. ***

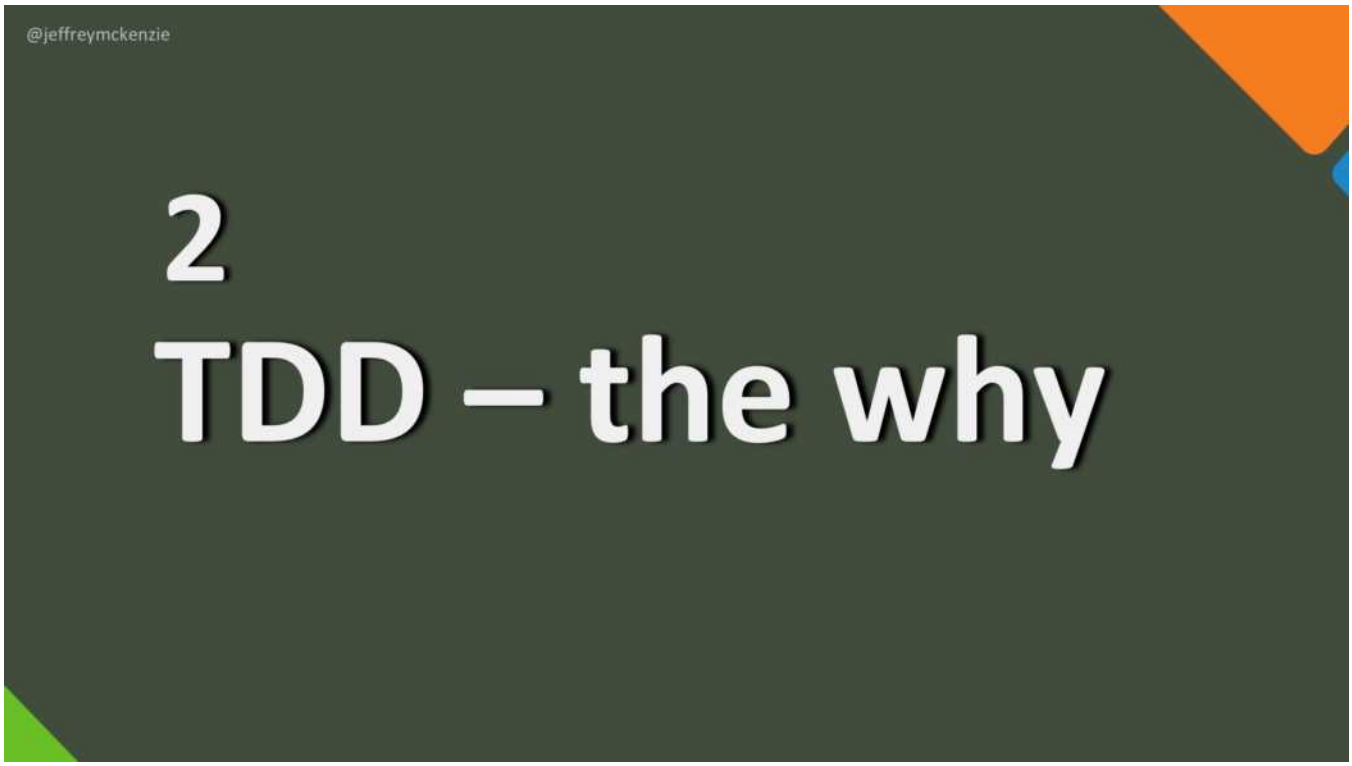



Re-run test = success

Coded only what needed to pass test

Coding ahead, anticipating =

risk of adding too much, unused code. ***




Now have general idea of TDD,

Examine why want to use, benefits

Important b/c extra work involved –

more time to write test for each requirement

& constantly run/update – a second code base***


first benefit TDD, improves code quality.

If diligent in test effort,

w/unit test for all reqs,
EQUALS
baked in verification app functionality.

Constant check correct = better code, less defect ***


Another area – design/org of code

b/c writing least possible code & test first…

Forces you to think in advance

as in EasyButton example: had to think about

result we wanted when writing test. ***


If disciplined in writing tests,

suites become form of doc for app

written = out of date
Passing test = code is used and working

My #1 reason for using TDD =
confidence to make changes ***


If continually rerunning old tests, then

Know immediately if broken, made mistake.


Brings us to main topic of today's session=

How bring benefits of TDD

To your SQL Server database.

Two questions:

**How** and **Where** do we test our data? ***

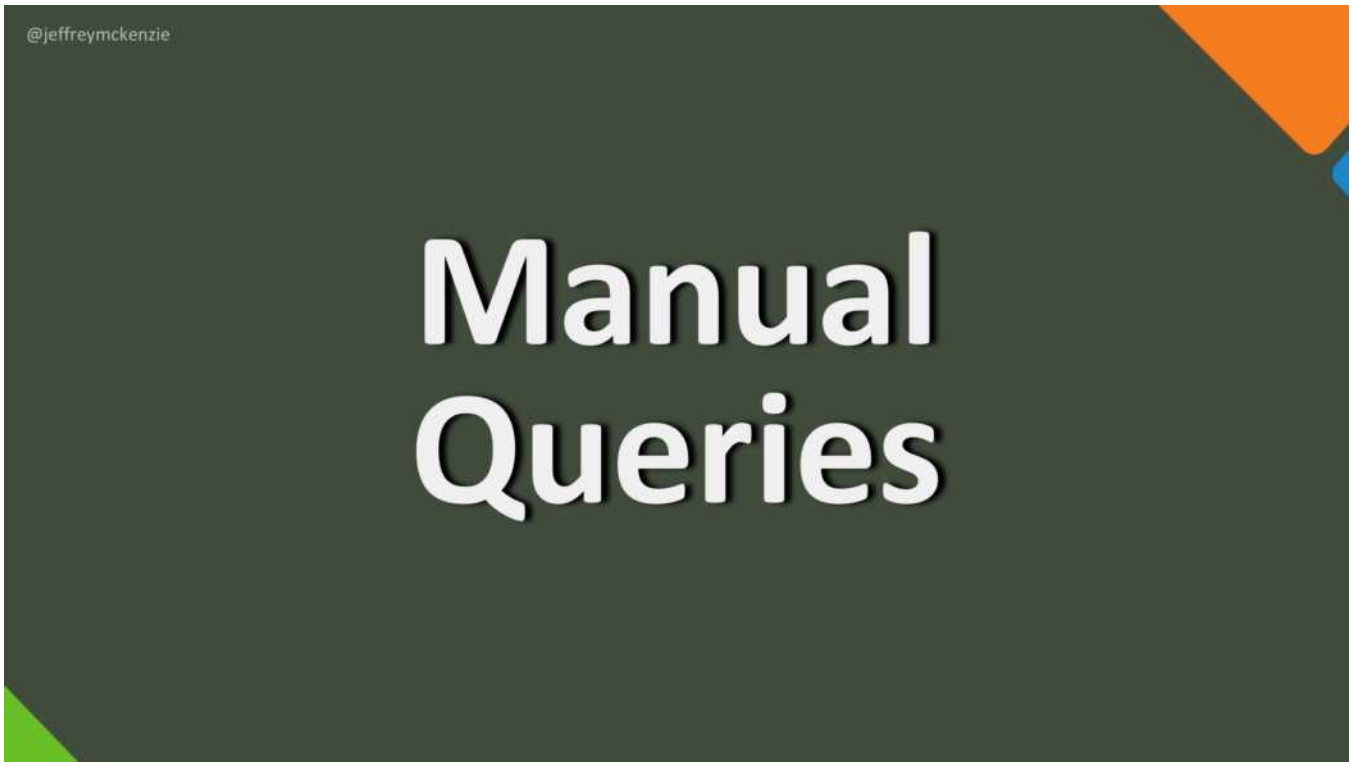

[ASK] What are some ways you approach data testing?

When writing app that depends on database,

We test without database because…

DB is external dependency =

slowness, unrelated issues ***




One way = manually query the database.
Write/run query/proc, verify

Works, but as data model and code changes,
How do you know still works? Run again --
Can automate, but have to roll own framework

Another method →
automation thru integration tests***


These tests are step up from manual queries,
problem = designed to test app itself,
data only indirectly.

Could be integration test succeeds coincid.
underlying SQL Has unidentified problems,
Or not testing all data paths in DB . ***


In addition to How, need ask Where

Don't need test everything – example, in app
don’t need test object prop has value assigned

Make sure test functionality,
not the underlying framework or language.
No test insert statement.
No test our ORM. ***


What is useful to test == complex behavior/logic.
Although stored procs not used to extent used to,

still great choice – intensive ops/multi DBOs

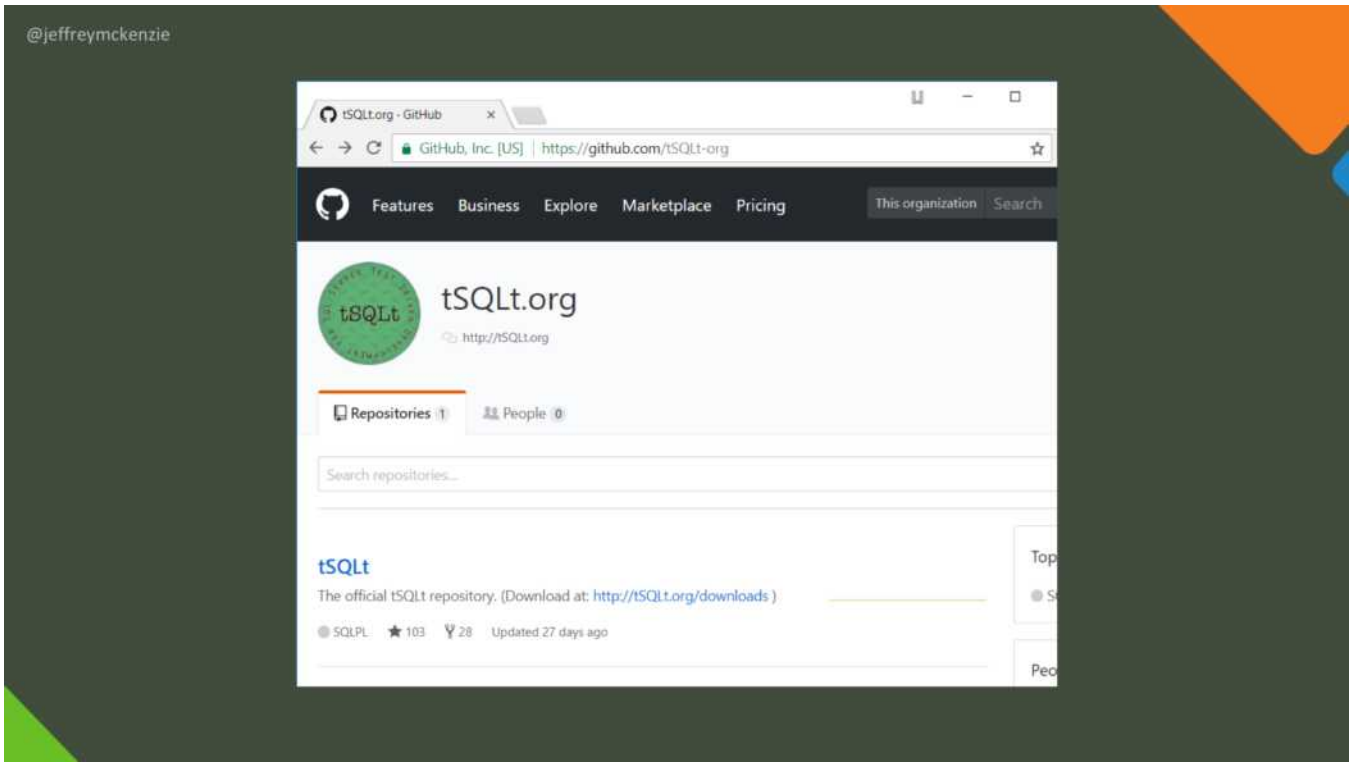
Wouldn't it be nice if way…
unit test SQL like app?
Well, it just so happens that we can! ***


T SQL T, unit test framework for SQL Server

Allows TDD approach to writing SQL

Installed entirely within SQL server

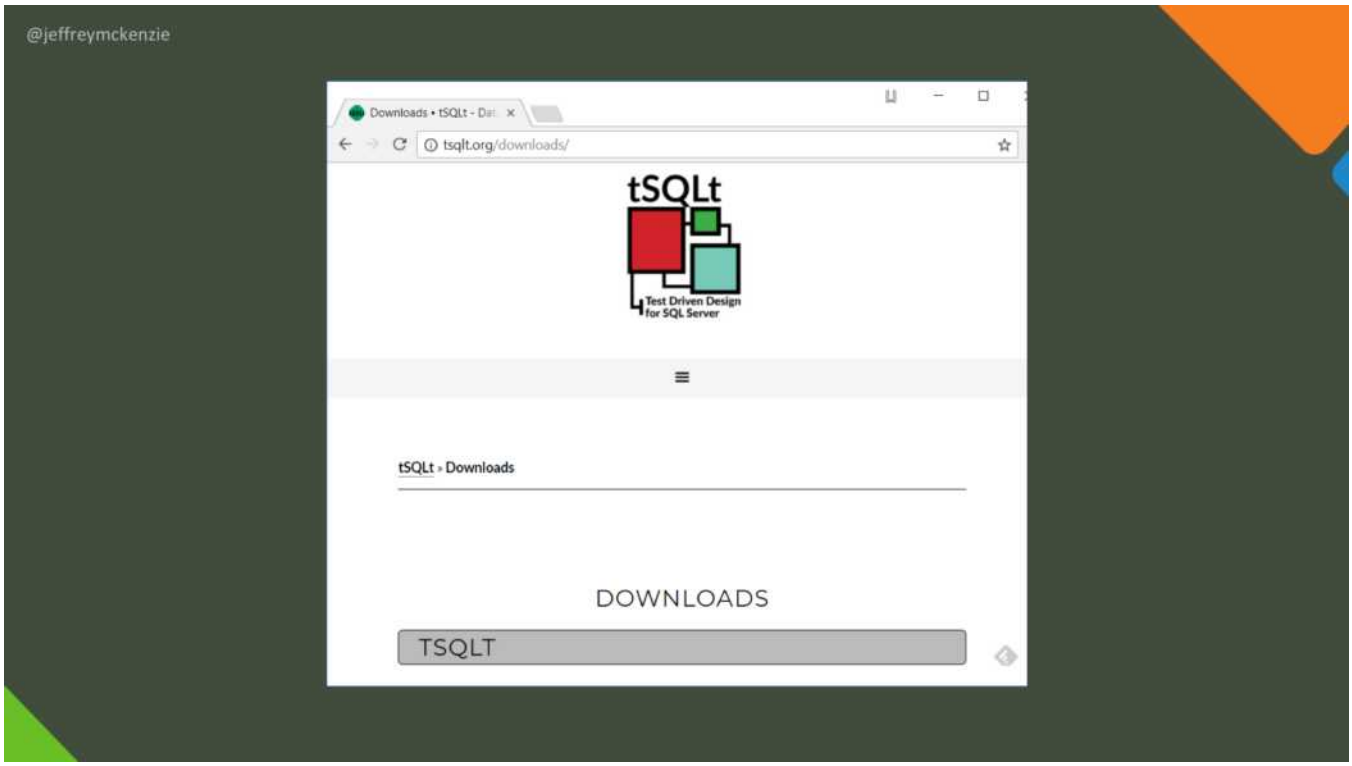
-- open source, GitHub. ***




TSQLT repository = tSQLt-org GitHub account,

Get zipped distributable from web site…

On download page at tsqlt.org ***




The whole framework clocks in zipped at 84K,

So that should pull down pretty fast… ***


We are going to walk through
real world production feature –
Implemented in SQL Server using tSQLt framework.

interest of protect client confidentiality:
changed almost everything about project:
the industry, client, product –
Only business problem remains..
Next → introduce you to our client… ***


Does anyone know this guy? [Ron Swanson]
best known for = director Parks/Rec Dept
in Pawnee Indiana, 6 years.

Less well known, after retiring from a long and
illustrious career as a civil servant,

Ron Swanson decided to purchase and run
favorite store… called… anyone? ***

=========
https://vignette.wikia.nocookie.net/parksandrecreation/images/0/06/Food_and_Stuff_2.png/revision/latest?cb=20120730155117
http://parksandrecreation.wikia.com/wiki/File:Food_and_Stuff_2.png


Food and Stuff.  anyone familiar with this place?
729 Glenmore Blvd, Pawnee, Indiana.

Ron buys all of his groceries there – describes as
"a discount food outlet equidistant
from my home and my work".

According to Mr. S, have broad/diverse
catalog of items, Including household paints…***

=========
http://parksandrecreation.wikia.com/wiki/Food_and_Stuff
https://vignette.wikia.nocookie.net/parksandrecreation/images/1/15/Food_and_Stuff.png/revision/latest?cb=20120730155051


garden supplies…

=========
https://upload.wikimedia.org/wikipedia/commons/b/bd/EWM_paint_2007.jpg
By Tom Murphy VII (Taken by uploader (user:brighterorange)) [Public domain], via Wikimedia Commons


Industrial tubing…

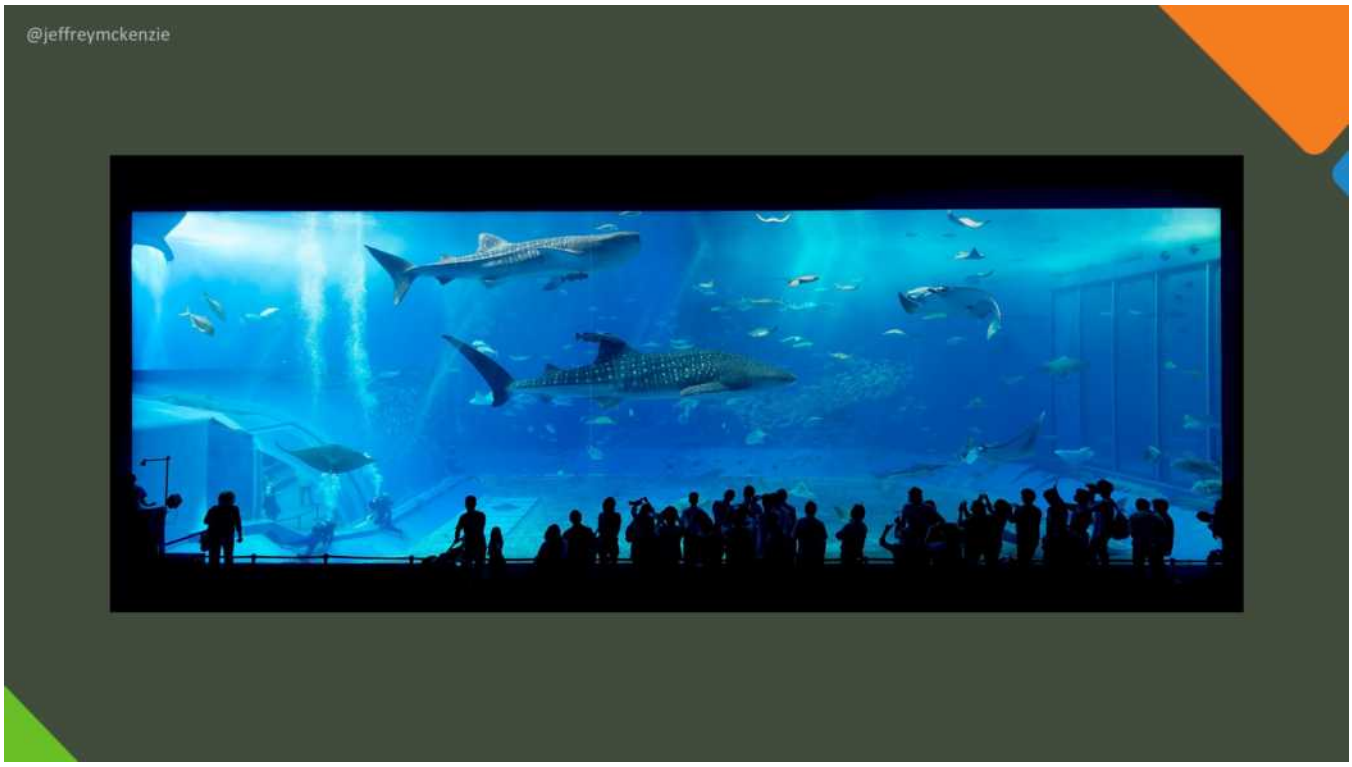
=========
https://upload.wikimedia.org/wikipedia/commons/d/d8/Shovel_leaning_against_a_wall.jpg
Santeri Viinamäki [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons


buckets of different sizes…..

=========
https://upload.wikimedia.org/wikipedia/commons/6/65/Pvc_cevi.jpg
Mm.zaletel from sl [GFDL (http://www.gnu.org/copyleft/fdl.html)], via Wikimedia Commons


Fishtanks….

=========
https://upload.wikimedia.org/wikipedia/commons/a/a2/Bassines_de_toutes_les_couleurs_march%C3%A9_%C3%A0_Hanoi.JPG
By Dinkum (Own work) [CC0], via Wikimedia Commons




shelving units ….

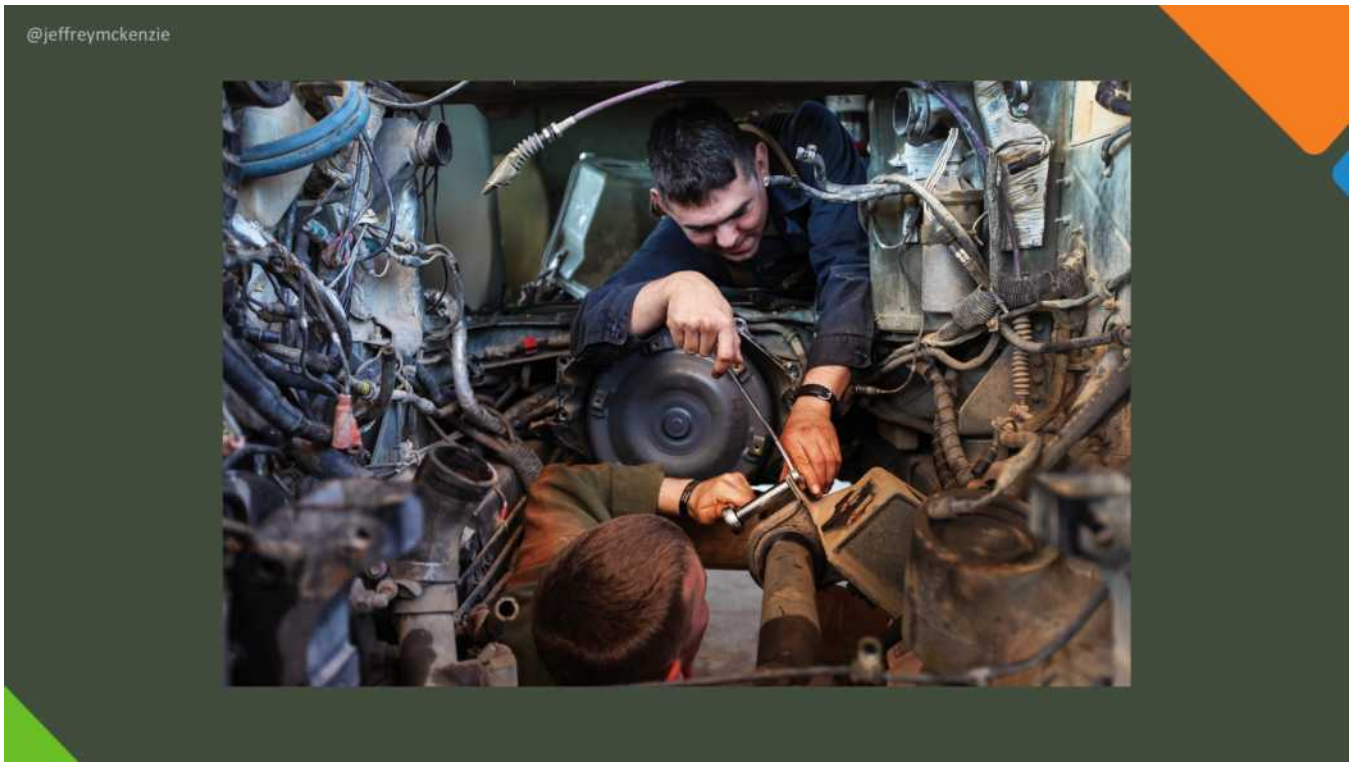
=========
https://upload.wikimedia.org/wikipedia/commons/4/47/Okinawa_Aquarium.jpg
By Jordy Meow (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons


And lead-based paints ….

=========
https://upload.wikimedia.org/wikipedia/commons/0/00/KAST_kast_designed_by_Marcel_Douwe_Dekker_in_1992.jpg
By Marcel Douwe Dekker (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0) or GFDL (http://www.gnu.org/copyleft/fdl.html)], via Wikimedia Commons


And it's not only products –
They perform various services,
Such as engine repair…

=========
https://upload.wikimedia.org/wikipedia/commons/6/69/LeadPaint1.JPG
By Thester11 (Own work) [CC BY 3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons




Passport photos…

=========
https://upload.wikimedia.org/wikipedia/commons/d/d7/Under_the_Hood_%289664838146%29.jpg
By Marines from Arlington, VA, United States (Under the Hood) [Public domain], via Wikimedia Commons


And catering

=========
https://upload.wikimedia.org/wikipedia/commons/4/4c/Miami_Passport_Agency_Director_Dooley_and_Deputy_Director_Ward_Show_Secretary_Kerry_a_New_Passport_in_Their_Manufacturing_Room_During_the_Secretary%27s_Day_Trip_to_the_City_%2826406970916%29.jpg
By U.S. Department of State from United States [Public domain], via Wikimedia Commons


So Ron's been doing pretty well…

=========
https://upload.wikimedia.org/wikipedia/commons/3/34/1956_-_Americus_Hotel_Buffet.jpg
Unknown author [Public domain], via Wikimedia Commons


business is strong…

As part of his shop,

He's got a point of sale system

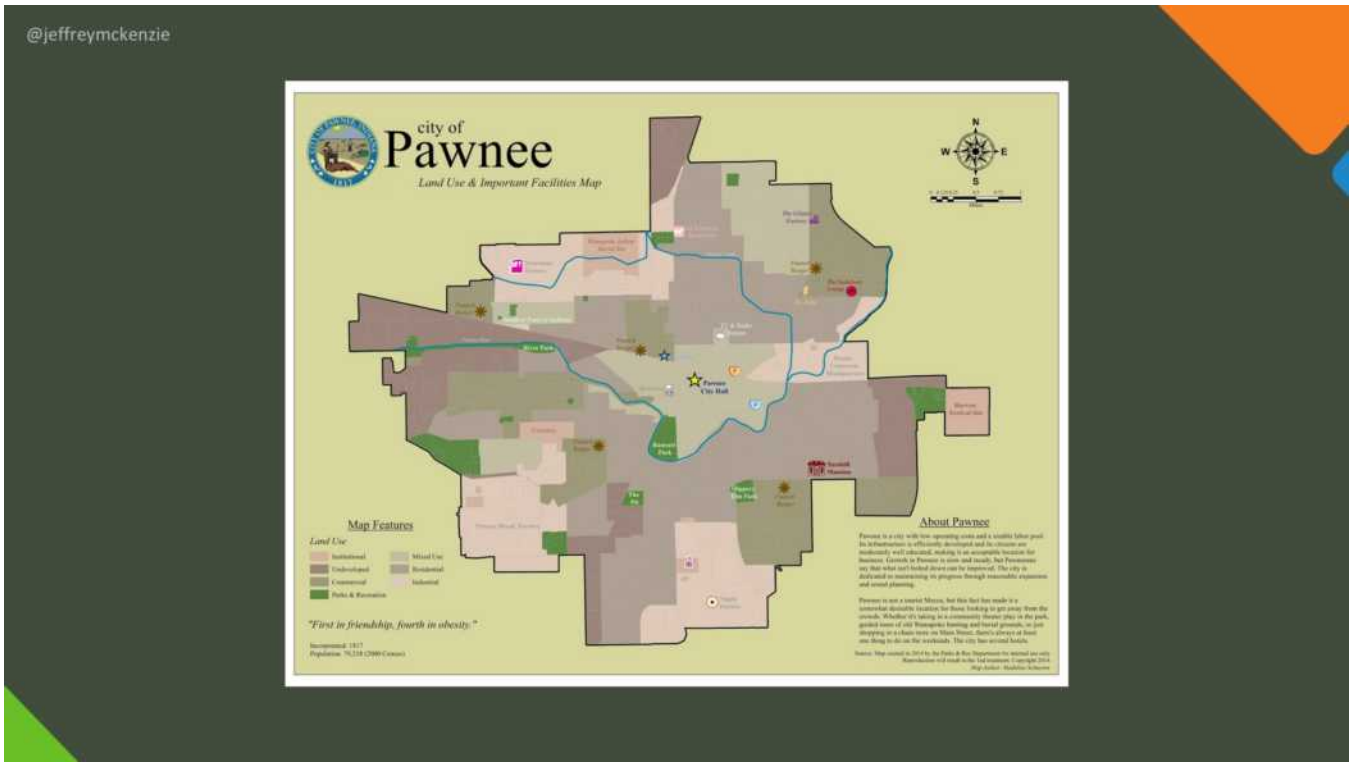
he's been using… ***

=========
https://vignette.wikia.nocookie.net/parksandrecreation/images/0/06/Food_and_Stuff_2.png/revision/latest?cb=20120730155117
http://parksandrecreation.wikia.com/wiki/File:Food_and_Stuff_2.png


Yeah, So no one ever accused Ron
of being a slave to technology.

He's also using this system to
 - take customer orders,
 - track inventory
 - manage his books,
and all that good stuff.
But Ron has a few problems. ***

=========
https://upload.wikimedia.org/wikipedia/commons/1/19/Zenith_Z-19_Terminal.jpg
By Jamie Cox from Melbourne, USA (Zenith Z-19 Terminal  Uploaded by Mewtu) [CC BY 2.0 (http://creativecommons.org/licenses/by/2.0)], via Wikimedia Commons

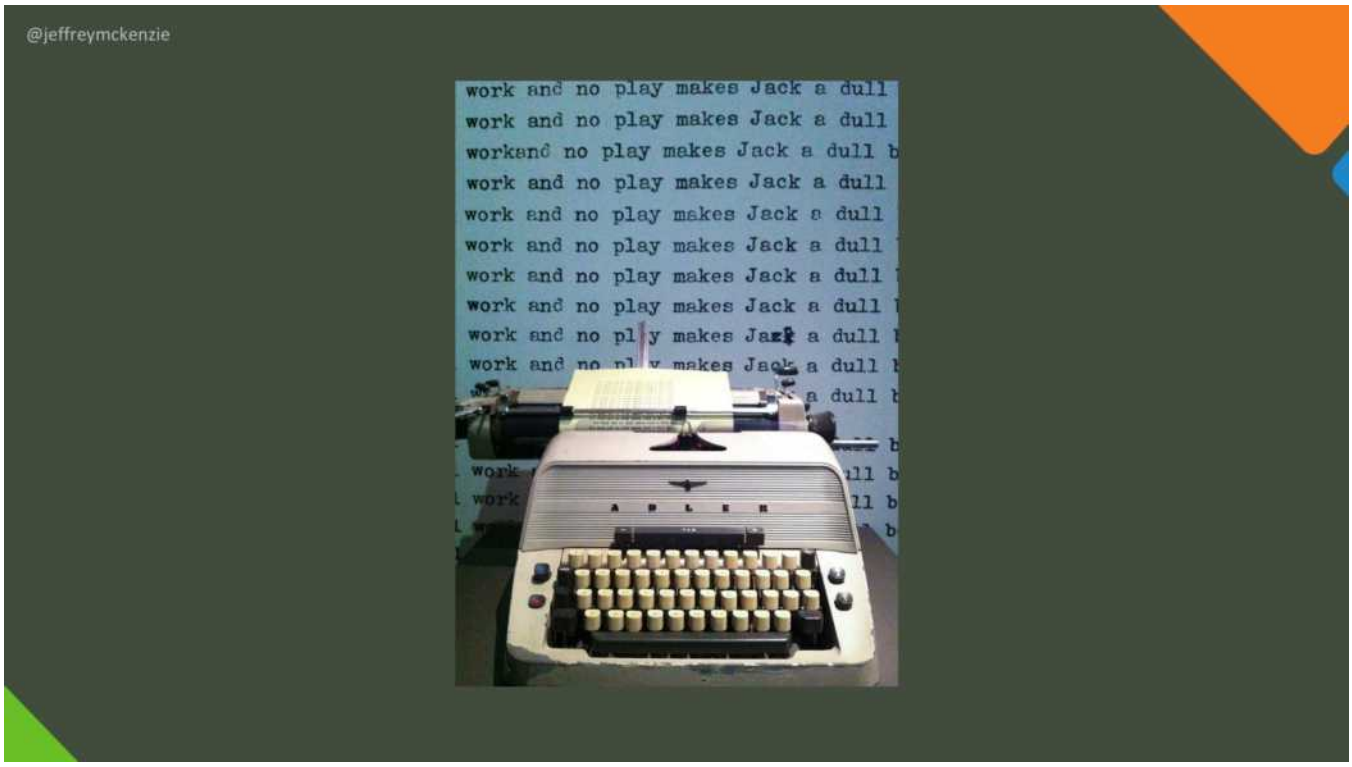



First, 3,000,000 new customers in DB

over last 12 months,

entire city of Pawnee, 80,000 people.

=========
http://1.bp.blogspot.com/-j9CbeOiNMLY/Vdr4b55bzcI/AAAAAAAAwOM/A_fjFcEBSJo/s1600/Pawnee.jpg
https://swimnova.com/map-of-pawnee-indiana.html




Runs customer reports,
See same names appear many times.

PoS system also allows Ron to mail
promo materials to customers, to remind
them when there are sales, or when new items
get in stock, like almond butter….
=========
https://upload.wikimedia.org/wikipedia/commons/3/3a/All_work_and_no_play_makes_Jack_a_dull_boy_%28The_Shining%29_%287957738500%29.jpg
By Marcel Oosterwijk from Amsterdam, The Netherlands [CC BY-SA 2.0 (https://creativecommons.org/licenses/by-sa/2.0)], via Wikimedia Commons


Or cattle prods….

=========
https://upload.wikimedia.org/wikipedia/commons/c/cf/Barney_butter_Jars.jpg
By Venomarv (Own work) [CC BY 1.0 (http://creativecommons.org/licenses/by/1.0)], via Wikimedia Commons


Ron, get complaints – 4-5 copies newsletter
Fortune on postage

So what is Ron's problem?
 - with system,
 - spec. with database?

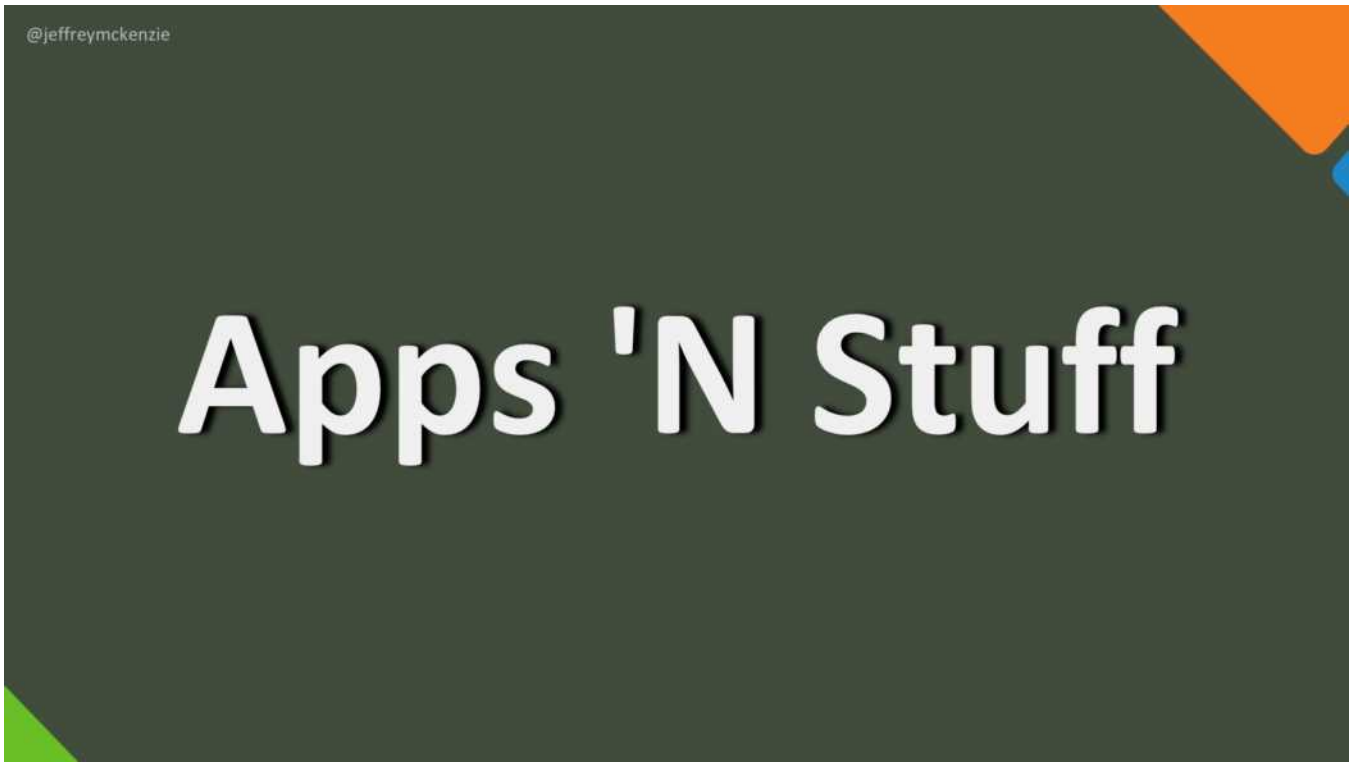
=========
https://upload.wikimedia.org/wikipedia/commons/d/d3/Electric_cattle_prod.jpg
Author Unknown


Yes, it's duplicate data.

Why might he?

[doesn't check for existing customers]
[no customer search]
[no DB cleaning]

Ron has asked all to solve
We are known as Apps N Stuff. ***




Ron wants us to fix his duplicate data problem.
He has 2 basic requirements for us.
First, he doesn't want to have to check
For duplicates up front.


No change UI for order process

(likes things way they are)

worried will slow,
Customers unhappy

Second, clean dupes himself***

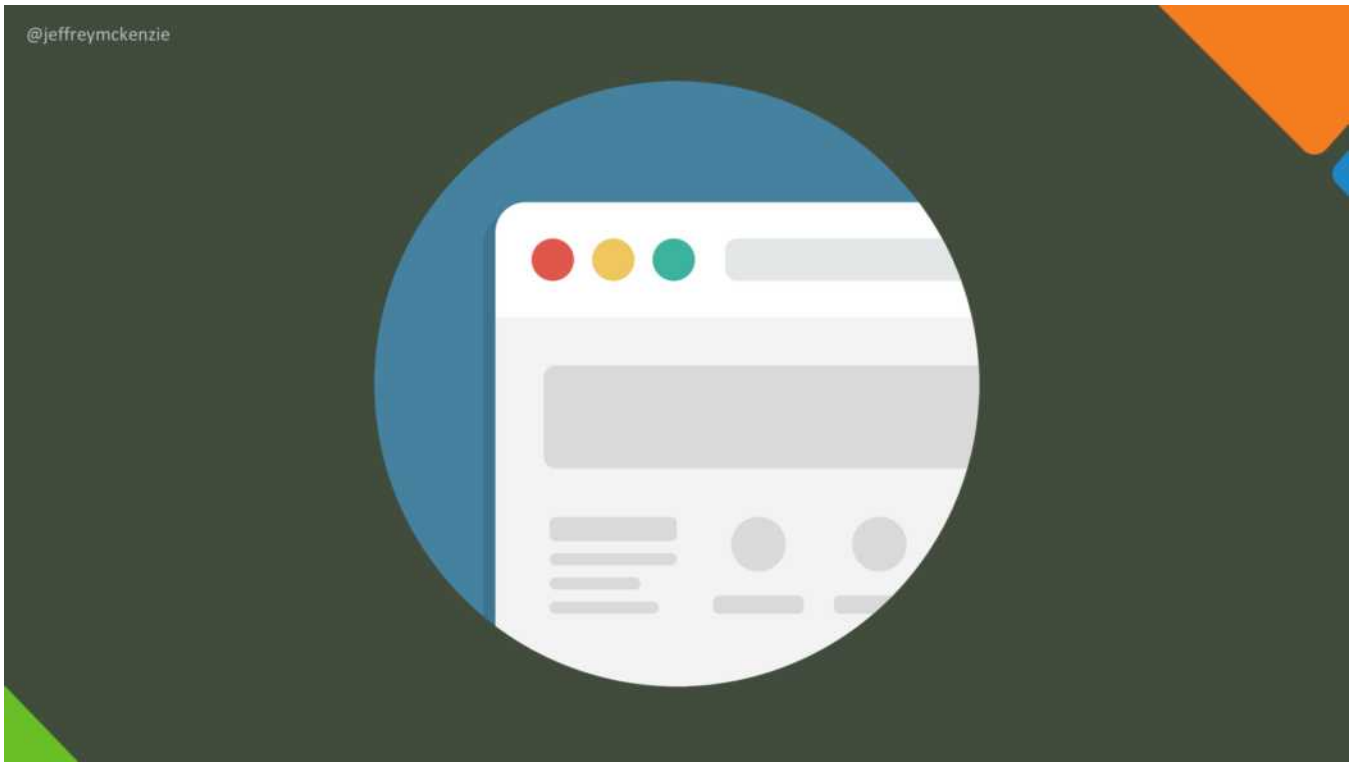

No Apps 'N Stuff visit every week,

No batch process clean,

-- doesn’t trust ***


build a solution
- allows Ron to access the data,
- find dupes,
- Merge together

First part of solution = UI comp. ***

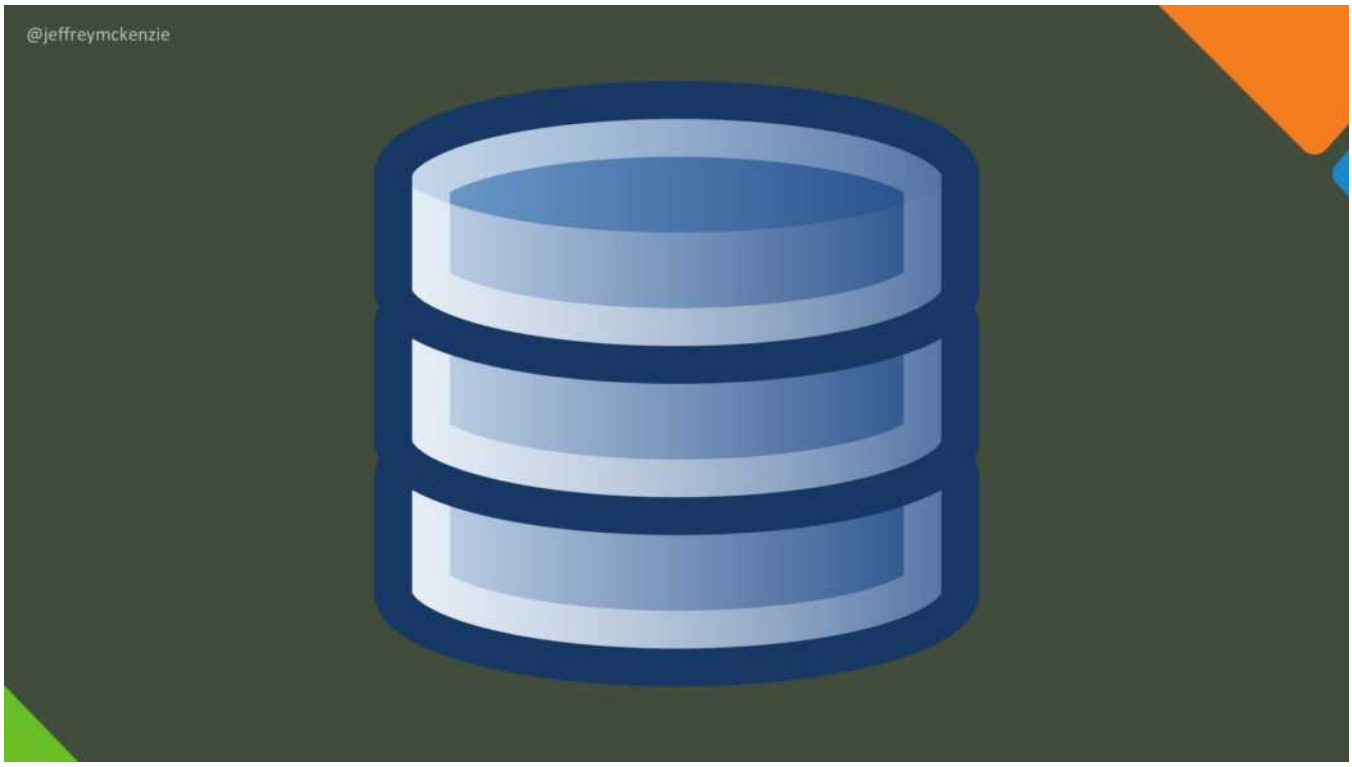



The actual screen that lets Ron find duplicates.

The second part of the solution

Is the backend database -- ***

=========
https://upload.wikimedia.org/wikipedia/commons/e/e0/Browser_ballon icon2.svg
By pixelbuddha [CC BY 3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons

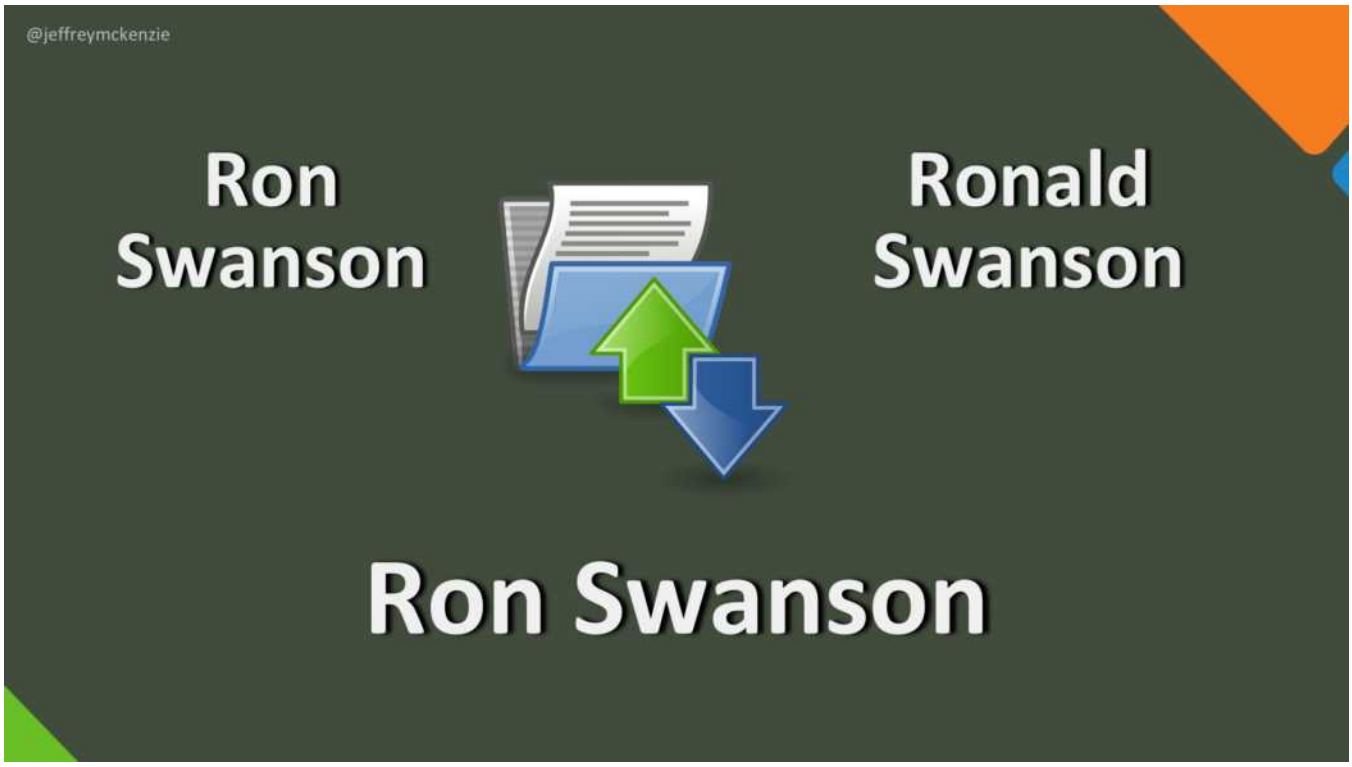



Focus effort today

write a stored procedure,

takes 2 duplicate customer records, ***

=========
https://commons.wikimedia.org/wiki/File:Applications-database.svg
By dracos (http://dracos.deviantart.com/#/d2y5ele) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons




merges → single customer.

UI = find the customers,

proc = take two records, combine

Complications = order history.

Next → data model. ***

=========
https://upload.wikimedia.org/wikipedia/commons/4/40/Data-transfer.svg
By RRZEicons (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

extremely simplified -- not recommend for prod

Customer = first name, last name, loyalty id
OrderDetail = item purchased
Order = joins Customer with an Order

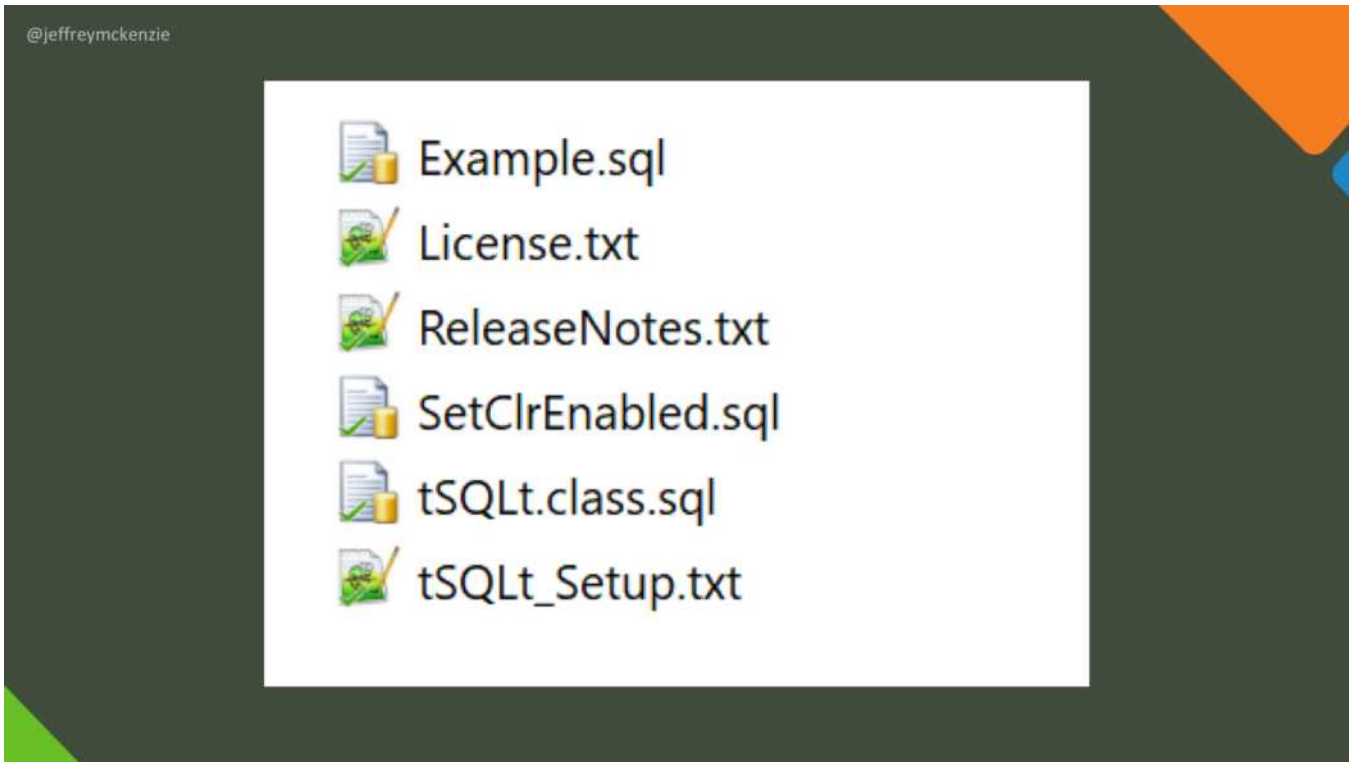
When done = single customer, with all orders,
duplicate customer deleted. ***


Before start write proc –

get TDD framework set up.

tSQLt framework is very small, and in fact,

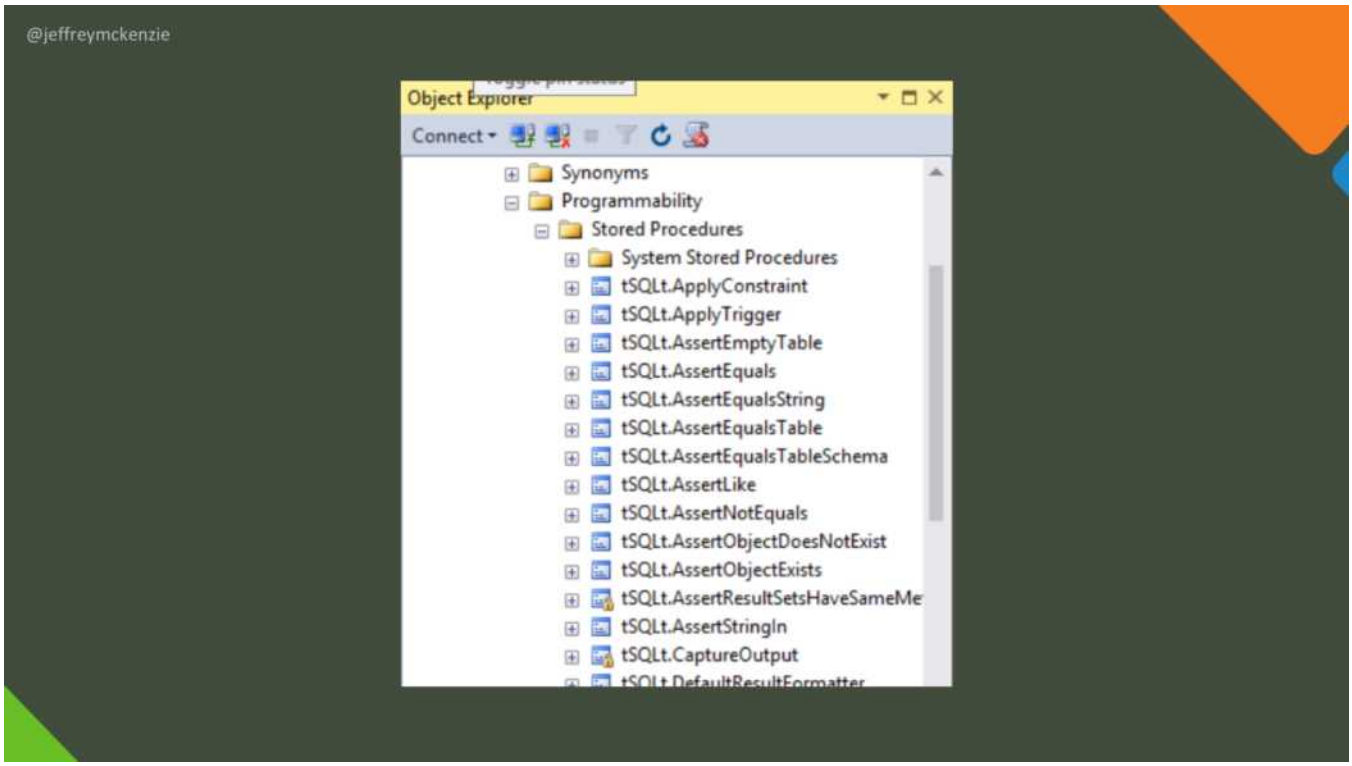
Contains only 6 files. ***




1. text file, explains the setup
2. License/release notes
3. Example tests

4. First, run Set CLR Enabled script – compiled assemblies tSQLt uses

5. execute tSqlt.class script….***




tSQLt.class installs all DBOs in own schema

Logically separated from dbo or any other

Now all tables, procs, functions,

-- Start working on our tests. ***


So we are going to do two things –

1. create a SQL Script installs tests in DB

2. create a SQL Script run tests

-- install script first. ***

@jeffreymckenzie

```sql
USE Food_And_Stuff
GO

EXEC tSQLt.NewTestClass
     @ClassName = N'FAS_Tests'
GO

------------- BEGIN [FAS_Tests].[InsertTestData] ----------
CREATE PROC FAS_Tests.InsertTestData
AS
  BEGIN
  END
------------- END [FAS_Tests].[InsertTestData] ------------
```

First, USE statement, FAS DB

Execute tSQLt statement = type schema name
Followed by command

Here, tSQLt New Test Class command,
create test class –
Actually = create new schema in DB,
tests run under***

@jeffreymckenzie

```sql
USE Food_And_Stuff
GO

EXEC tSQLt.NewTestClass
     @ClassName = N'FAS_Tests'
GO

------------- BEGIN [FAS_Tests].[InsertTestData] ----------
CREATE PROC FAS_Tests.InsertTestData
AS
  BEGIN
  END
------------- END [FAS_Tests].[InsertTestData] ------------
```

clear separation between
-- actual code,
-- tests
-- tSQLt framework.
Warning – delete test class of same name
Useful if changing/installing tests, good to script all

Next → create proc, inserts data needed for tests
First, create data for customer.

@jeffreymckenzie

```sql
---------------- BEGIN CUSTOMERS -------------------
      DECLARE @RonaldId INT = 1, @RonId INT = 2

      INSERT INTO dbo.Customer(
          CustomerId, FirstName, LastName, LoyaltyId
      )
      VALUES(@RonaldId, 'Ronald', 'Swanson', 1234)

      INSERT INTO dbo.Customer(
          CustomerId, FirstName, LastName, LoyaltyId
      )
      VALUES(@RonId, 'Ron', 'Swanson', 1234)
---------------- END CUSTOMERS --------------------
```

Want to test dupe = elim, so need 2

**Ron** Swanson = customer to keep
**Ronald** Swanson = duplicate

Insert customer ids (Ron is number 2),
first name, last name, and loyalty ID.

The loyalty ID = tracks who gets what discounts

@jeffreymckenzie

```sql
---------------- BEGIN ORDERS ----------------------
     DECLARE @RonaldOrderId INT = 11, @RonOrderId INT = 22

     -- customer 1 order
     INSERT INTO [dbo].[Order](OrderId, CustomerId)
     VALUES(@RonaldOrderId, @RonaldId)

     -- customer 2 order
     INSERT INTO [dbo].[Order](OrderId, CustomerId)
     VALUES(@RonOrderId, @RonId)
     ---------------- END ORDERS ------------------------
```

same for orders,

order ID, both Ron and Ronald –

Again, end of this process,

Ron = 2 orders under his name.

@jeffreymckenzie

```
---------------- BEGIN ORDER DETAIL ----------------
 DECLARE @RonaldDetailId INT = 111
 DECLARE @RonDetailId INT = 222

 INSERT INTO [dbo].[OrderDetail](OrderDetailId, OrderId, ItemName)
 VALUES(@RonaldDetailId, @RonaldOrderId,
        'T-Bone Steaks: Val-U 20 Pack')

 INSERT INTO [dbo].[OrderDetail](OrderDetailId, OrderId, ItemName)
 VALUES(@RonDetailId, @RonOrderId,
        'Bacon Strips: Val-U 100 Pack')
---------------- END ORDER DETAIL -------------------
```

Finally for order details.

Can see =

Ronald bought a value pack, 20 T-Bone
Ron, 100 bacon strips

Clearly same person. ***


Before continue with test setup,
Talk concept fakes, mocks in tests

All part, test isolation mentioned before

IF particular method test = webservice, DB call,

Create mock = test simple, focused***

@jeffreymckenzie

```
public class Button {
   public String Click() {
       return "That was easy.";
   }
}
```

back → EasyButton example.

implemented click method?

Service call instead ***

@jeffreymckenzie

```
public class Button {
   public String Click(MessageService) {
       return MessageService.Get();
   }
}
```

pseudocode = clearer

Now unit test, test both click and service

Longer, maybe unrelated failures

= create fake of service call ***

@jeffreymckenzie

```
public class FakeMessageService {
   public String Get() {
        return “When I eat,
                it is the food
                that is scared.
                    - Ron Swanson";
   }
}
```

Instead of passing real instance Message Service,

Create fake version, returns same message

Makes sure we get result From click method,

Also that message service will never fail. ***

@jeffreymckenzie

```
------------- BEGIN [FAS_Tests].[Setup] -------------------
CREATE PROC FAS_Tests.Setup AS
BEGIN

--create fakes for tests against tables
      EXEC tSQLt.FakeTable 'dbo.OrderDetail'
      EXEC tSQLt.FakeTable 'dbo.Order'
      EXEC tSQLt.FakeTable 'dbo.Customer'

-- populate fake tables
      EXEC FAS_Tests.InsertTestData
  END
GO
------------- END [FAS_Tests].[Setup] ---------------------
```

next part of install script, create setup proc

tSQLt , Proc called setup, runs before each test

Leverage to do prep work every test will need

First → create some Fakes for tables want to test***

@jeffreymckenzie

```
------------- BEGIN [FAS_Tests].[Setup] -------------------
CREATE PROC FAS_Tests.Setup AS
BEGIN

--create fakes for tests against tables
      EXEC tSQLt.FakeTable 'dbo.OrderDetail'
      EXEC tSQLt.FakeTable 'dbo.Order'
      EXEC tSQLt.FakeTable 'dbo.Customer'

-- populate fake tables
      EXEC FAS_Tests.InsertTestData
  END
GO
------------- END [FAS_Tests].[Setup] ---------------------
```

Fake table command in tSQLt =

Replaces actual table, Empty copy, no constraints.

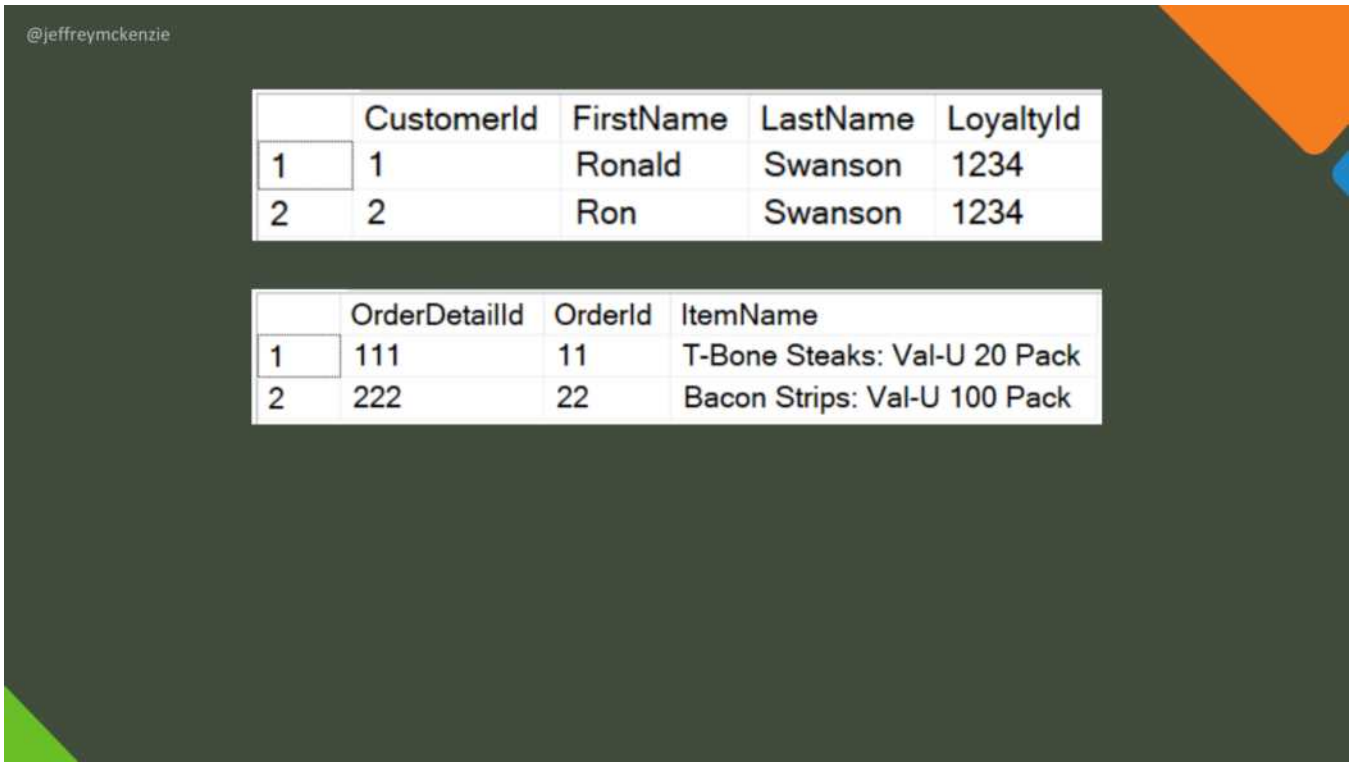
isolated from rest of DB,
can test operations on that table alone.

After fakes, proc just created, insert data
About Ronald and Ron***


| | CustomerId | FirstName | LastName | LoyaltyId |
|---|---|---|---|---|
| 1 | 1 | Ronald | Swanson | 1234 |
| 2 | 2 | Ron | Swanson | 1234 |

Here is our customer table,

With both Ron and Ronald in there…***




Then the Order detail table,

With its meat extravaganza… ***

@jeffreymckenzie

| | CustomerId | FirstName | LastName | LoyaltyId |
|---|---|---|---|---|
| 1 | 1 | Ronald | Swanson | 1234 |
| 2 | 2 | Ron | Swanson | 1234 |

| | OrderDetailId | OrderId | ItemName |
|---|---|---|---|
| 1 | 111 | 11 | T-Bone Steaks: Val-U 20 Pack |
| 2 | 222 | 22 | Bacon Strips: Val-U 100 Pack |

| | OrderId | CustomerId |
|---|---|---|
| 1 | 11 | 1 |
| 2 | 22 | 2 |

And finally the Order table,
Matches customer to order

Note: only happens when test is run,

After test return original state***

@jeffreymckenzie

```
----------- BEGIN GivenMerge-ThenCustomerIsCorrect --------------
CREATE PROC FAS_Tests.[test GivenMerge-ThenCustomersIsCorrect]
AS
  BEGIN

--Arrange
      DECLARE @RonaldId INT= 1
      DECLARE @RonId INT= 2  --CustomerId we want to keep

      DECLARE @ExpectedRonFirstName NVARCHAR(50) =
```

Ready,create first test.
tSQLt, each test a proc, starts “test”

This test =
--do merge
--ensure customer data correct

Start arrange section,
remember tables will be populated***

@jeffreymckenzie

```
----------- BEGIN GivenMerge-ThenCustomerIsCorrect --------------
CREATE PROC FAS_Tests.[test GivenMerge-ThenCustomerIsCorrect]
AS
  BEGIN

--Arrange
      DECLARE @RonaldId INT= 1
      DECLARE @RonId INT= 2  --CustomerId we want to keep

      DECLARE @ExpectedRonFirstName NVARCHAR(50) =
```

So to set the ExpectedRonFirstName variable,

What are a couple of ways we could do that?

Hardcode name –
Select based on ID --

***

@jeffreymckenzie

```sql
----------- BEGIN GivenMerge-ThenCustomerIsCorrect --------------
CREATE PROC FAS_Tests.[test GivenMerge-ThenCustomerIsCorrect]
AS
  BEGIN

--Arrange
      DECLARE @RonaldId INT= 1
      DECLARE @RonId INT= 2  --CustomerId we want to keep

      DECLARE @ExpectedRonFirstName NVARCHAR(50) = (
              SELECT FirstName FROM dbo.Customer
              WHERE CustomerId = @RonId
      )
```

If select value based on the ID,

That allows us to test other IDs

Without

Having to change a hardcoded string***

@jeffreymckenzie

```sql
DECLARE @ExpectedRonFirstName NVARCHAR(50) = (
        SELECT FirstName FROM dbo.Customer
        WHERE CustomerId = @RonId
)

DECLARE @ExpectedRonLastName NVARCHAR(50)= (
        SELECT LastName FROM dbo.Customer
        WHERE CustomerId = @RonId
)

DECLARE @ExpectedLoyaltyId INT = (
        SELECT LoyaltyId FROM dbo.Customer
        WHERE CustomerId = @RonId
)
```

--Same for all other values in Customer

How to compare expected/actual values?

table compare…***

@jeffreymckenzie

```sql
CREATE TABLE FAS_Tests.Expected(
    CustomerId          INT NOT NULL,
    FirstName           NVARCHAR(50) NOT NULL,
    LastName            NVARCHAR(50) NOT NULL,
    LoyaltyId           INT NOT NULL
)

-only Ron should exist
INSERT INTO FAS_Tests.Expected(
    CustomerId, FirstName, LastName, LoyaltyId)
VALUES(@RonId, @ExpectedRonFirstName,
  , @ExpectedRonLastName
  , @ExpectedLoyaltyId
)
```

First, create Expected table in FAS_Tests
(separate from app code

Insert expected values in table,
because should only see one record after merge.

That's it for the arrange section –

Let's finish the test: ***

@jeffreymckenzie

```
--Act
      EXEC dbo.MergeCustomer
           @DuplicateCustomer = @RonaldId,
           @CustomerToKeep = @RonId

--Assert
      EXEC tSQLt.AssertEqualsTable
           @Expected = N'FAS_Tests.Expected',
           @Actual = N'dbo.Customer'
           @FailMsg = N'Customer table has incorrect data.'

 END
GO
-------- END GivenMergeCustomers-ThenCustomerIsCorrect ------------
```

Act section = execute merge customer stored proc

Assert section, use tSQLt AssertEqualsTable assertion –

Verifies two tables have same data,

We are comparing expected table in test schema
With actual Customer table = Fake table


After finished tests –

1. Run install script to get latest tests into DB

2. Create test run script to exec those tests

It's pretty simple.***

@jeffreymckenzie

```sql
USE Food_And_Stuff

/*
EXEC tSQLt.RunAll
GO

EXEC tSQLt.Run
    N'FAS_Tests'
GO */

EXEC tSQLt.Run
     N'FAS_Tests.[test GivenMergeCustomers-ThenCustomerIsCorrect]'
GO
```

3 ways run test in tSQLt:

1. Run all
2. Run test class
3. Run test name

[ASK]
So if we run our test,
what's going to happen? ***

@jeffreymckenzie

```
(0 row(s) affected)
[FAS_Tests].[test GivenMergeCustomers...] failed: (Error) Could not
find stored procedure 'dbo.MergeCustomer'.[16,62]{MergeCustomer,49}

+----------------------+
|Test Execution Summary|
+----------------------+
|No|Test Case Name                  |Dur(ms)|Result|
+--+--------------------------------+-------+------+
|1 |[FAS_Tests].[test GivenMerge...]|    236|Error |
-----------------------------------------------------------------
Msg 50000, Level 16, State 10, Line 1: Test Case Summary: 1 test
case(s) executed, 0 succeeded, 0 failed, 1 errored.
-----------------------------------------------------------------
```

Output -- tSQLt shows us:

-- actual error, (MergeCustomer stored proc not found)
-- test summary:
    test name,
    duration,
    result

Red = failed***


| | CustomerId | FirstName | LastName | LoyaltyId |
|---|---|---|---|---|
| 1 | 1 | Ronald | Swanson | 1234 |
| 2 | 2 | Ron | Swanson | 1234 |

Customer table,

eliminate the Ronald record,

Least amount code need to pass

[Next → Show blank test] ***

@jeffreymckenzie

```sql
USE Food_And_Stuff
GO

CREATE PROCEDURE [dbo].[MergeCustomer]
  @DuplicateCustomer INT,
  @CustomerToKeep    INT
AS
    BEGIN




    END
GO
```

Here’s merge customer proc skeleton

[delete duplicate]

@jeffreymckenzie

```sql
USE Food_And_Stuff
GO

CREATE PROCEDURE [dbo].[MergeCustomer]
  @DuplicateCustomer INT,
  @CustomerToKeep    INT
AS
    BEGIN

        DELETE dbo.Customer
        WHERE CustomerId = @DuplicateCustomer

    END
GO
```

Yep, that's right –

let’s Run our test again….

@jeffreymckenzie

```
(1 row(s) affected)

+----------------------+
|Test Execution Summary|
+----------------------+

|No|Test Case Name                  |Dur(ms)|Result |
+--+--------------------------------+-------+-------+
|1 |[FAS_Tests].[test GivenMerge...]|     54|Success|
-------------------------------------------------------------------
Test Case Summary: 1 test case(s) executed, 1 succeeded, 0 failed,
0 errored.
-------------------------------------------------------------------
```

Now our test is passing.

Let's write our next test,

This time for the order table.***

@jeffreymckenzie

| | CustomerId | FirstName | LastName | LoyaltyId |
|---|---|---|---|---|
| 1 | 1 | Ronald | Swanson | 1234 |
| 2 | 2 | Ron | Swanson | 1234 |

| | OrderDetailId | OrderId | ItemName |
|---|---|---|---|
| 1 | 111 | 11 | T-Bone Steaks: Val-U 20 Pack |
| 2 | 222 | 22 | Bacon Strips: Val-U 100 Pack |

| | OrderId | CustomerId |
|---|---|---|
| 1 | 11 | 1 |
| 2 | 22 | 2 |

This test, a little different –
Remember, keep all order history.

What expect order table @ bottom
[ASK] To look like after the merge is done?

[CustomerId should be two for both orders]
Right, so let's write a test for that -- ***

@jeffreymckenzie

```
--------- BEGIN GivenMergeCustomers-ThenOrderIsCorrect ------------
CREATE PROC FAS_Tests.[test GivenMergeCustomers-ThenOrderIsCorrect]
AS
  BEGIN
--Assert
      DECLARE @RonaldId INT= 1
      DECLARE @RonId INT= 2  --Customer Id we want to keep

      CREATE TABLE FAS_Tests.Expected(
          OrderId           INT NOT NULL,
          CustomerId        INT NOT NULL
      )
```

We will need to write the test a little differently
For this one – for the customer test,
We only needed one record, so we could
Insert one record into the expected table.
Here, we could have any number of orders
Between the two customers –
So how should we fill the expected table here? ***

@jeffreymckenzie

```sql
CREATE TABLE FAS_Tests.Expected(
    OrderId           INT NOT NULL,
    CustomerId        INT NOT NULL
)

--only orders for Ron (CustomerId 2) should exist
INSERT INTO FAS_Tests.Expected(
    OrderId,
    CustomerId
)
SELECT OrderId, @RonId
FROM dbo.[Order]
WHERE CustomerId = @RonaldId
   OR CustomerId = @RonId
```

Essentially what we are doing here
Is selecting every record in the order table
For both customers, and inserting
The order id, as well as the customerId
We want to keep.
Let's finish with the act and assert
***

@jeffreymckenzie

```
--Act
      EXEC dbo.MergeCustomer
           @DuplicateCustomer = @RonaldId,
           @CustomerToKeep = @RonId

--Assert
      EXEC tSQLt.AssertEqualsTable
           @Expected = N'FAS_Tests.Expected',
           @Actual = N'dbo.Order'
           @FailMsg = N'Order table has incorrect data.'

 END
GO
-------- END GivenMergeCustomers-ThenOrderIsCorrect ------------
```

This is the same act and arrange as the last test,
Except we are now checking the Order table.

Let's add our new test to the script… ***

@jeffreymckenzie

```sql
USE Food_And_Stuff

EXEC tSQLt.Run
     N'FAS_Tests.[test GivenMergeCustomers-ThenCustomerIsCorrect]'
GO

EXEC tSQLt.Run
     N'FAS_Tests.[test GivenMergeCustomers-ThenOrderIsCorrect]'
GO
```

Now we will run both to make sure
The previous test still passes. ***

@jeffreymckenzie

```
+----------------------+
|Test Execution Summary|
+----------------------+

|No|Test Case Name                    |Dur(ms)|Result |
+--+----------------------------------+-------+-------+
|1 |[FAS_Tests].[test Given...Customer|     37|Success|
|2 |[FAS_Tests].[test Given...Order   |     33|Failure|
-------------------------------------------------------------------
Msg 50000, Level 16, State 10, Line 7, Test Case Summary:
2 test case(s) executed, 1 succeeded, 1 failed, 0 errored.
-------------------------------------------------------------------
```

So our customer test is passing,
But our order test is not – which is good
Because we haven't written the code yet.
When we are using the AssertEqualsTable
Assertion, we get an additional error message… ***

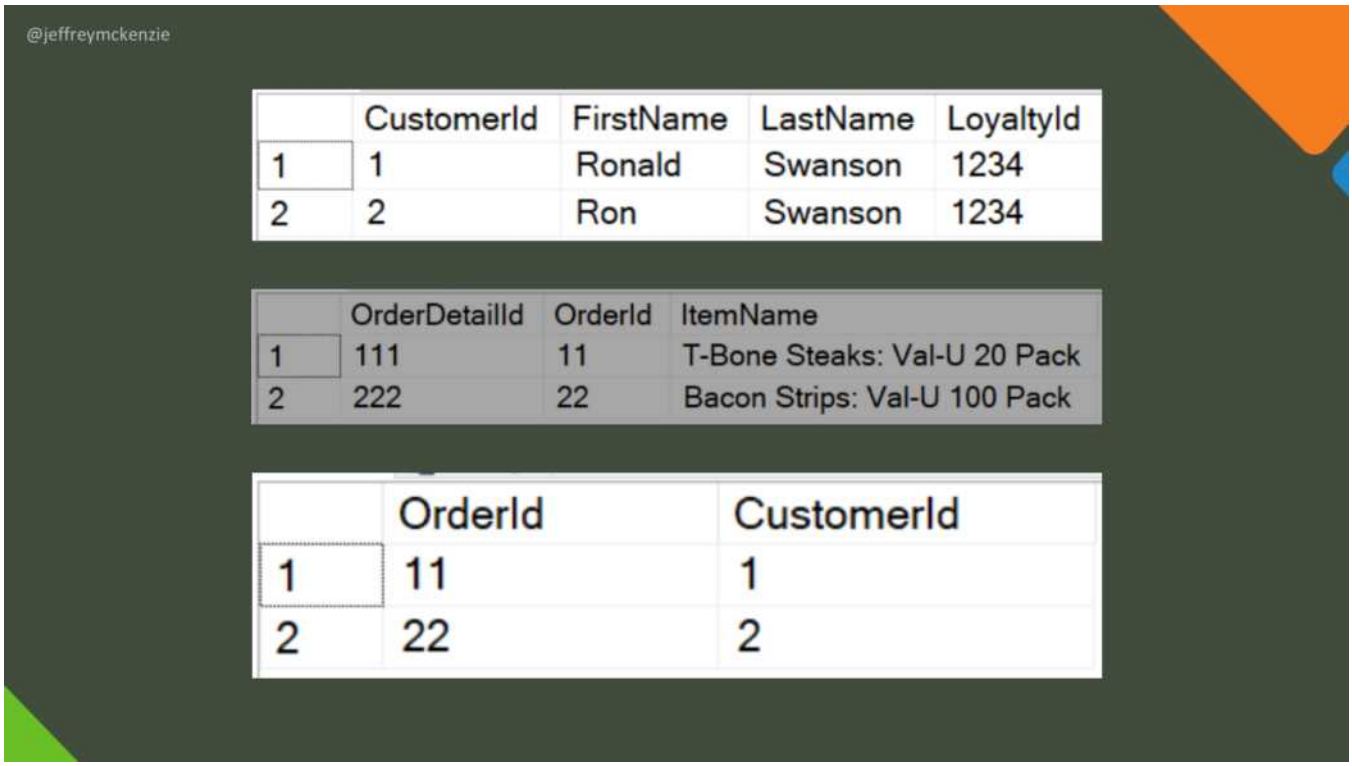
@jeffreymckenzie

```
(1 row(s) affected)

[FAS_Tests].[test GivenMergeCustomers-ThenOrderIsCorrect] failed:
(Failure) Order table has incorrect data.

|_m_|OrderId|CustomerId|
+---+-------+----------+
|<  |11     |2         |
|=  |22     |2         |
|>  |11     |1         |
```

This graphical table shows us
Exactly how the data is incorrect
A less than sign means that the record
Is in the expected table but not the actual table –
The greater than sign means the record
Is in the actual table but not the expected table,
And the equals sign means it's in both tables.
This output can help you troubleshoot data issues. ***




So now we want to make our order table
Have all the orders for Ronald and Ron
Belong to Ron – what's the least amount
Of code we need to accomplish this?
[UPDATE CustomerId column]
So let's update our procedure…. ***

@jeffreymckenzie

```sql
CREATE PROCEDURE [dbo].[MergeCustomer]
  @DuplicateCustomer INT,
  @CustomerToKeep    INT
AS
    BEGIN
        DELETE dbo.Customer
        WHERE CustomerId = @DuplicateCustomer

        UPDATE dbo.[Order]
        SET CustomerId = @CustomerToKeep
        WHERE CustomerId = @DuplicateCustomer
           OR CustomerId = @CustomerToKeep
    END
GO
```

We've added our update statement,
And we will run our tests again ***…

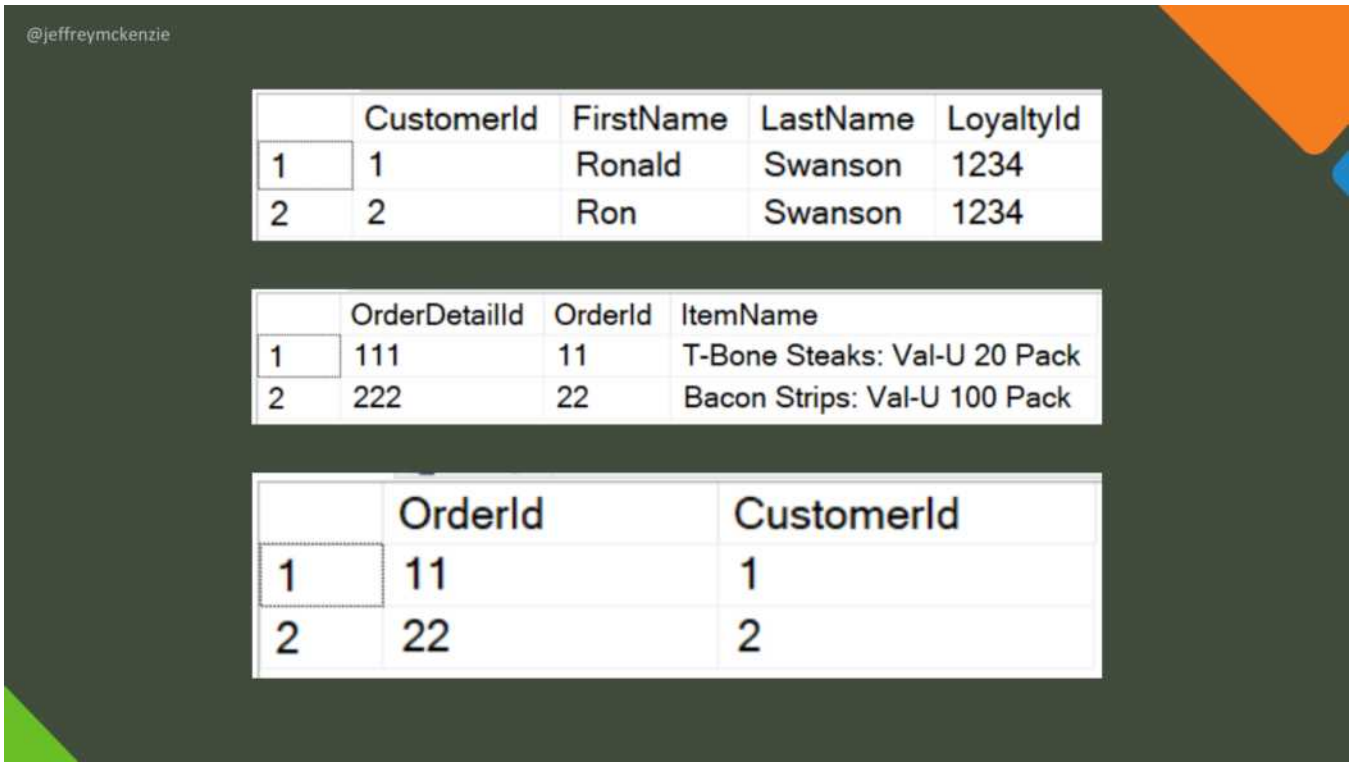
@jeffreymckenzie

```
(1 row(s) affected)

+----------------------+
|Test Execution Summary|
+----------------------+

|No|Test Case Name                                |Dur(ms)|Result |
+--+----------------------------------------------+-------+-------+
|1 |[FAS_Tests].[test GivenMerge...Customer]      |     37|Success|
|2 |[FAS_Tests].[test GivenMerge...Order]         |     40|Success|
-------------------------------------------------------------------
Test Case Summary: 2 test case(s) executed, 2 succeeded, 0 failed,
0 errored.
-------------------------------------------------------------------
```

And now both are passing. ***




Now because OrderDetail is tied to Order,
OrderDetail will stay the same,
So we don't need a test for that,
Because we are not performing any action
On that table.
Our tests are passing, so let's try this
On the live table – here's our script … ***

@jeffreymckenzie

```sql
USE Food_And_Stuff
GO
BEGIN TRAN
    DECLARE @RonaldId INT= 1
    DECLARE @RonId INT= 2  /* Customer Id we want to keep */
    EXEC FAS_Tests.InsertTestData
    EXEC dbo.MergeCustomer
        @DuplicateCustomer = @RonaldId,
        @CustomerToKeep = @RonId
    SELECT * FROM dbo.Customer
    SELECT * FROM dbo.[Order]
    SELECT * FROM dbo.OrderDetail
ROLLBACK TRAN
GO
```

We are going to run this in a transaction
So we can reset the tables and run it repeatedly.
First we have our regular customer IDs for
Ronald and Ron.
Then we run our insert data procedure,
Execute the merge procedure,
Then do some selects to see
What the data looks like….
***

```
Msg 547, Level 16, State 0
Procedure MergeCustomer, Line 7 [Batch Start Line 2]

The DELETE statement conflicted with the REFERENCE
constraint "FK_Order_Customer_CustomerId".

The conflict occurred in database "Food_And_Stuff",
table "dbo.Order", column 'CustomerId'.

The statement has been terminated.
```

And we have an error –
What's going on here?
[DELETED a customer record
still attached to an Order.]
Remember I said when you create
A fake table in tSQLt it creates
A blank copy without restraints?
We didn't get this error in our test
Because we had no constraints. ***

@jeffreymckenzie

```
------------- BEGIN [FAS_Tests].[Setup] -------------------
CREATE PROC FAS_Tests.Setup AS
BEGIN
--create fakes for tests against tables
      EXEC tSQLt.FakeTable 'dbo.OrderDetail'
      EXEC tSQLt.FakeTable 'dbo.Order'
      EXEC tSQLt.FakeTable 'dbo.Customer'
      EXEC tSQLt.ApplyConstraint N'dbo.Order',
          N'FK_Order_Customer_CustomerId'
-- populate fake tables
      EXEC FAS_Tests.InsertTestData
  END
GO
------------- END [FAS_Tests].[Setup] ---------------------
```

Fortunately we have a way
To add constraints to our tests,
Using the tSQLt.ApplyConstraint command.
After we create our fake table, all
Constraints are removed, but we can then
Apply individual constraints using the
Table name and constraint name – here
We are applying the foreign key constraint
Between the Order and Customer tables. ***

@jeffreymckenzie

```
[FAS_Tests].[test GivenMerge...CustomerIsCorrect] failed: (Error)
DELETE statement conflicted with "FK_Order_Customer_CustomerId".
The conflict occurred in database "Food_And_Stuff", table
"dbo.Order", column 'CustomerId'.[16,0]{MergeCustomer,7}
+----------------------+
|Test Execution Summary|
+----------------------+
|No|Test Case Name                  |Dur(ms)|Result|
+--+--------------------------------+-------+------+
|1 |[FAS_Tests].[test GivenMerge...]|    236|Error |
-----------------------------------------------------------------
Msg 50000, Level 16, State 10, Line 1: Test Case Summary: 1 test
case(s) executed, 0 succeeded, 0 failed, 1 errored.
-----------------------------------------------------------------
```

When we run the test again,
We get the same error we did when running live.
So we now have a failing test that we need to fix.
Let's take a look at our
MergeCustomer procedure… ***

@jeffreymckenzie

```sql
CREATE PROCEDURE [dbo].[MergeCustomer]
  @DuplicateCustomer INT,
  @CustomerToKeep    INT
AS
    BEGIN
        DELETE dbo.Customer
        WHERE CustomerId = @DuplicateCustomer

        UPDATE dbo.[Order]
        SET CustomerId = @CustomerToKeep
        WHERE CustomerId = @DuplicateCustomer
           OR CustomerId = @CustomerToKeep
    END
GO
```

Again , what's the simplest change we can make
To get our test to pass?

[flip the statements…] ***

@jeffreymckenzie

```sql
CREATE PROCEDURE [dbo].[MergeCustomer]
  @DuplicateCustomer INT,
  @CustomerToKeep    INT
AS
    BEGIN
        UPDATE dbo.[Order]
        SET CustomerId = @CustomerToKeep
        WHERE CustomerId = @DuplicateCustomer
           OR CustomerId = @CustomerToKeep

        DELETE dbo.Customer
        WHERE CustomerId = @DuplicateCustomer
END
GO
```

Right, we make the update statement first,
So when we do the customer delete,
That customer will have no relationship
With the Order table.
Let’s run our tests again…. ***

@jeffreymckenzie

```
(1 row(s) affected)

+----------------------+
|Test Execution Summary|
+----------------------+

|No|Test Case Name                                |Dur(ms)|Result |
+--+----------------------------------------------+-------+-------+
|1 |[FAS_Tests].[test GivenMerge...Customer]      |     37|Success|
|2 |[FAS_Tests].[test GivenMerge...Order]         |     40|Success|
-------------------------------------------------------------------
Test Case Summary: 2 test case(s) executed, 2 succeeded, 0 failed,
0 errored.
-------------------------------------------------------------------
```

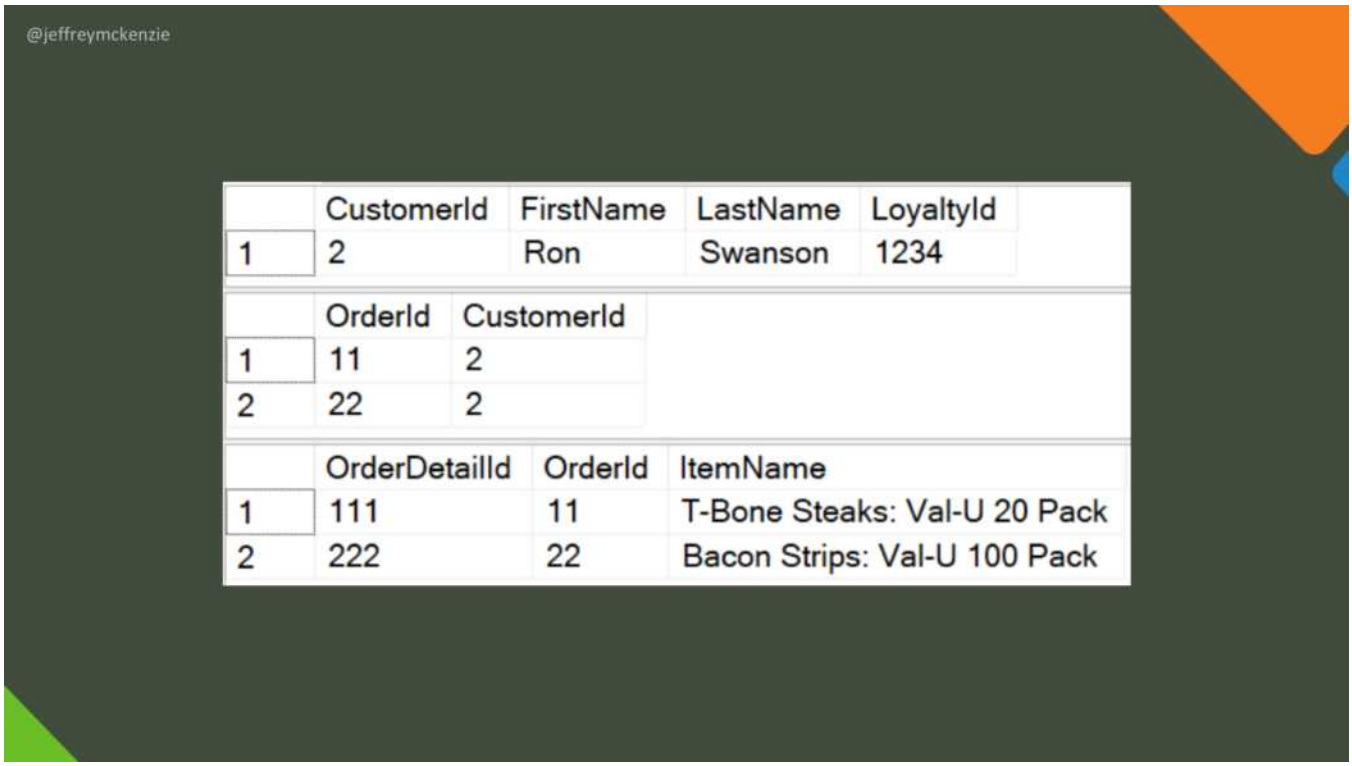
And now we are back to passing tests.

***

@jeffreymckenzie

```sql
USE Food_And_Stuff
GO
BEGIN TRAN
    DECLARE @RonaldId INT= 1
    DECLARE @RonId INT= 2  /* Customer Id we want to keep */
    EXEC FAS_Tests.InsertTestData
    EXEC dbo.MergeCustomer
        @DuplicateCustomer = @RonaldId,
        @CustomerToKeep = @RonId
    SELECT * FROM dbo.Customer
    SELECT * FROM dbo.[Order]
    SELECT * FROM dbo.OrderDetail
ROLLBACK TRAN
GO
```

Let’s try our live test again and see
What that returns….
Again we are inserting test data
And performing the merge
against the actual table….
***




Here's the output – no errors
And it has what we expect,
Just the one record for Ron,
Who is attached to all of the orders now.
As we continue to add features
To this, we now have a set of tests
We can use to make sure everything
Is working properly.
***


So congratulations everybody –
Ron is very happy with our work,
And has offered everyone
A five dollar gift card to Food and Stuff.

=========
https://vignette.wikia.nocookie.net/parksandrecreation/images/0/06/Food_and_Stuff_2.png/revision/latest?cb=20120730155117
http://parksandrecreation.wikia.com/wiki/File:Food_and_Stuff_2.png


That’s all I have….

=========
http://parksandrecreation.wikia.com/wiki/Food_and_Stuff
https://vignette.wikia.nocookie.net/parksandrecreation/images/1/15/Food_and_Stuff.png/revision/latest?cb=20120730155051


Good morning everyone –
Thanks for being here to learn
About SQL Server testing using TDD

My name is Jeff McKenzie,
And I am a Practice Manager for App Dev and Infrastructure
At Insight Digital Innovation in Columbus Ohio
We used to be Cardinal Solutions
But acquired in August 2018 by Insight