Who Cares About Relational Database Design?

Louis Davidson

Editor of 

SQLSATURDAY
#1068 - Jacksonville, FL - May 4th, 2024

JACKSONVILLE SQL SERVER USERS GROUP

Thank you to ALL of our sponsors! - Be sure to stop by all tables!

AI Platinum

NLP LOGIX
AI REALIZED

Gold

VyStar Credit Union

vmware by Broadcom

Platinum

UNF School of Computing
College of Computing, Engineering and Construction

SIOS

((CENTRIC))

IDERA

Silk

DBHawk

Silver

COZYROC

DBeaver

onecall

SQLGrease

Quest

Bronze

Simple Talk

Octopus Deploy

Microsoft

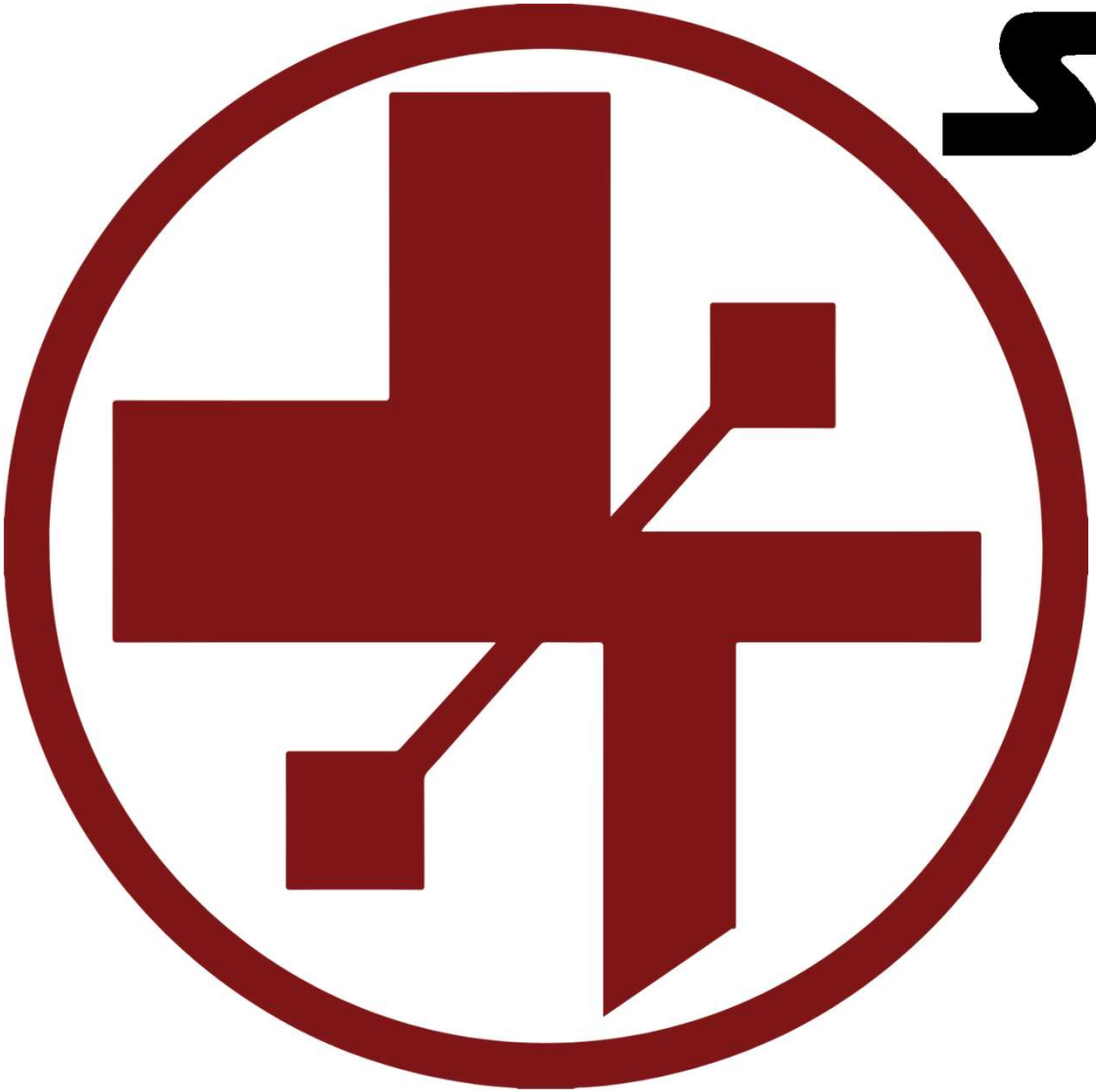
In-Kind

O'REILLY

drsg.ion

Monthly Meetings
3rd Wednesday of each month
jssug.org

#SQLSatJax



SQL CLINIC

Get your SQL Questions answered here
by our SQL Experts

(Across from Registration)

501 Legion Charitable Donation & LEGO Drive

Thank the 501 Legion for Supporting Our Event!

JSSUG Will Match Donations up to $1000

Donation Bucket on Registration Table

LEGO donation on V for Victory Table

Costume Contest Rules

1. Take a picture with the SQL Saturday Backdrop during the event

2. Post the picture to Twitter/X and include the hashtag #SQLSatJax24CC

3. The tweet using the hashtag that has the most "likes" wins a prize!

Session Evaluations

Your feedback is important to us!

Please fill out and hand to speaker after the session!

Event Evaluation

Fill out event evaluation card in your bag and visit all sponsors to be entered to win an Xbox Series X – (Must be present to win)


SQL
SATURDAY

Who Cares About Relational Database Design?

Louis Davidson

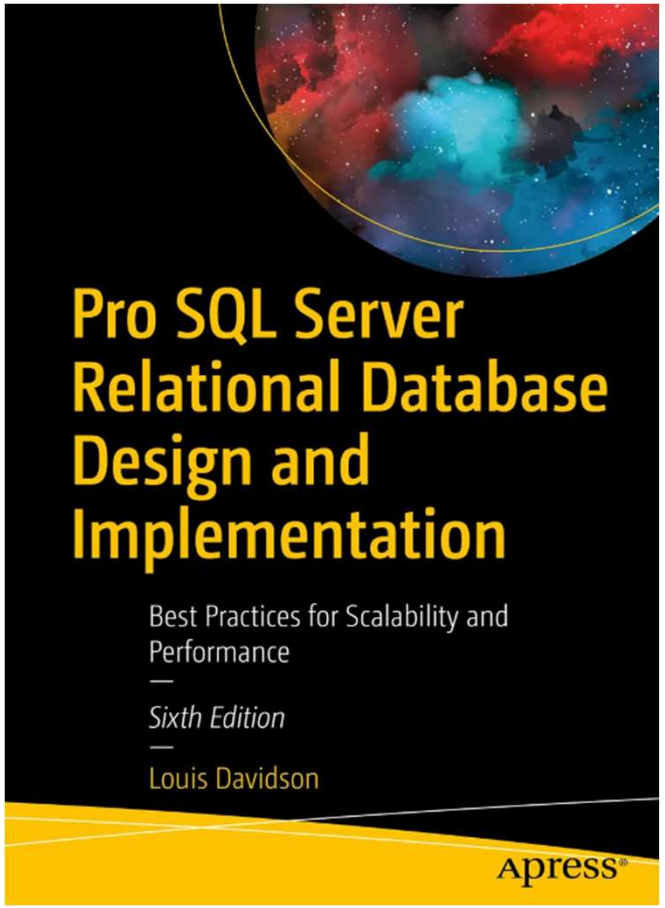
Editor of 

I do, but who am I?

- Database programmer/architect > 20 years
- Microsoft MVP 19 Times
- Simple-Talk.com Editor
- Written 6 books on database design
  - They were technically all versions of the same book.
  - They at least had slightly different titles each time
  - Seventh Edition? Who knows

- Super Brief Contact Info: DRSQL
  - Twitter, Website (.org), Email (@hotmail.com)

- Article writing interest?
  - editor@simple-talk.com



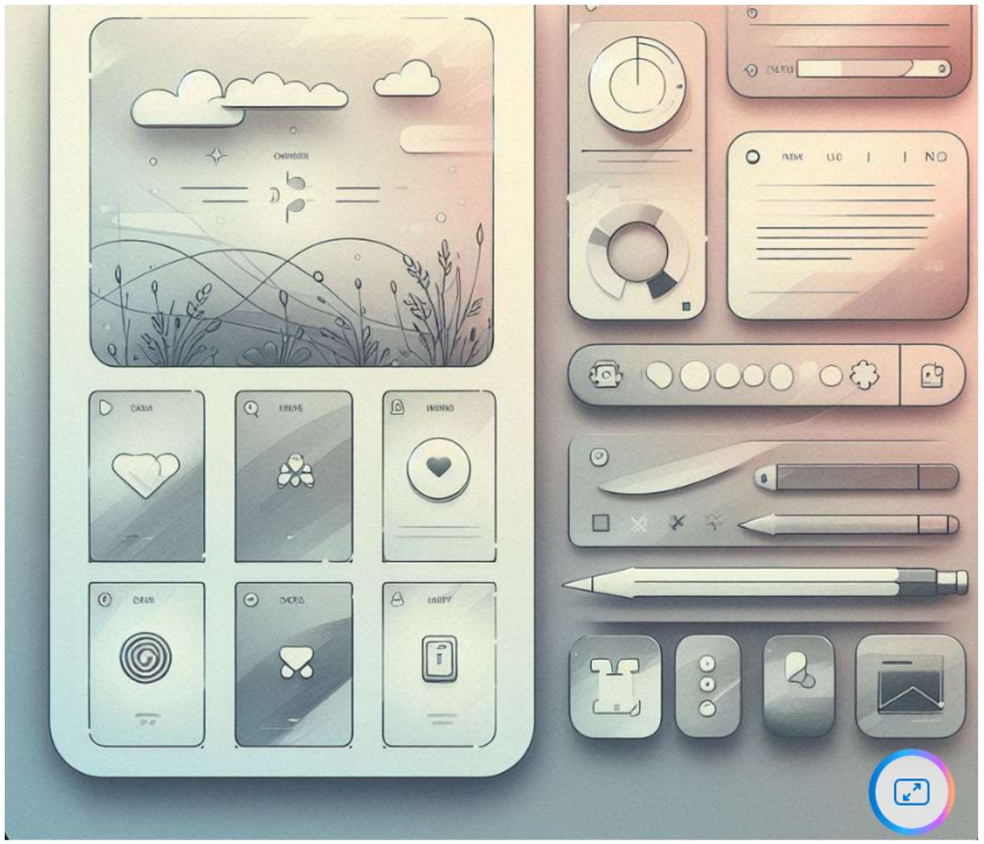
Well? Who cares?

What about people who create application databases?

Bad interfaces do suck

We all like a nice UI

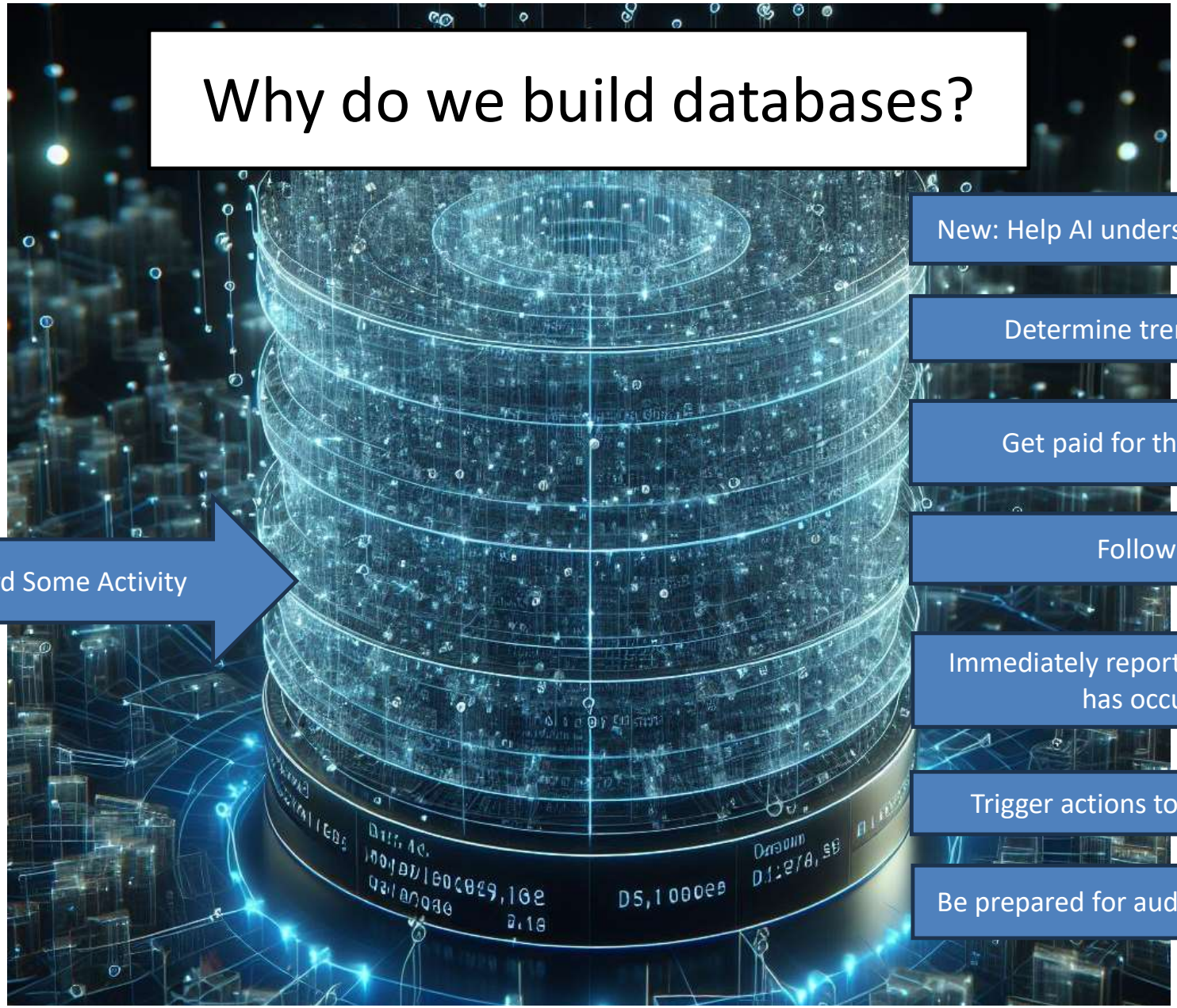
- We just want it to do what we told it to do…
- And more importantly store the data we expect it to.
- In ONE SUCCESSFUL TRY.



So, who is affected by poorly designed databases?

In short

Why do we build databases?



Record Some Activity

New: Help AI understand our needs

Determine trends

Get paid for that action…

Follow Up

Immediately report that this action has occurred

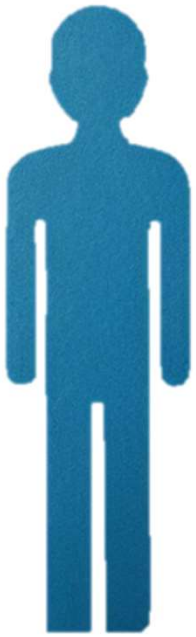
Trigger actions to occur

Be prepared for audits

So like I said… all of us should care about db design when we are building a new solution that stores data..

Or altering an existing app.

Or adding new features to an existing app.



So what can I do? I am just one person!

Start by caring early…

- **Think**: What problem are we really trying to solve?
- **Communicate**: Get agreement from all involved that you are solving the right problem
  - Users
  - Management
    - Project
    - Client
  - Programmers
  - Anyone else who might disagree with you and cause your design harm later. (other than your significant other, unless you work together.)
- **Document**: Write it down so it cannot be disputed in a few weeks (or years)

- The common term for what you need is Requirements

Consider the consequences of bad requirements…

If everyone decided on Lake Eerie (instead of Erie), then everyone is to blame

WRITE REQUIREMENTS DOWN

- WRITE IT DOWN. Seriously.
- True for Agile, Extreme, or Full Blown Committee Driven Waterfall
- Use to provide the target for the rest of the project
  - Design
  - Coding
  - Testing
- Make certain you get sign-off from all decision makers
- Anything not written down can and will change without notice
- WRITE IT DOWN. Before you code. I mean it.

Design goal

- A database that works well, and meets your needs, and doesn't lie to you more than you expect it to…
- And you go from there…
  - Chose a data store that meets your needs
  - All your needs…

Prerequisites

- There are no variations except for those who know a norm, and no subtleties for those who have not grasped the obvious.
  - C. S. Lewis, An Experiment in Criticism

Prerequisites.Relational History/Theory

- Codd's Rules
  - 13 Rules that qualified a system as a "relational" database system, written back in 1985 (I was in high school)
  - Will help you see the thought process behind how products like SQL Server are implemented
  - Outlines aspects of RDBMS, including: Catalog; Data Access (and a lack of direct physical access); NULL values; Integrity controls; Set based operations
- Basic relational theory
  - Trade books
  - Textbooks
  - College classes

Database Design Process

- Conceptual
  - Early model, generally representing the tables or things being modeled and their relationship to each other
- Logical
  - Ideal representation of the problem that is trying to be solved. Generally the “complete” data requirements
- Physical - Code
  - Represents what is actually implemented
- Physical – Maintenance and Tuning
  - Adjusting on-disk structures (engine (on disk, in mem, etc) indexes, partitions, distribution, etc) over time to handle changes in the usage, without altering the meaning of the system

- Code is largely developer based, maintenance and tuning is largely DBA focused…
- These layers correspond loosely to deliverables called data models

What does it mean to data model?

- Design and capture the semantic details of the database
- Including
  - Structure
  - Predicates
  - Documentation
- May include more than can be implemented
- Most modeling languages have a graphical representation that makes communication easier
- A picture **is** worth a thousand words when communicating with non-modelers

Start with the Conceptual Model

- Start by understanding the requirements
- Build a Conceptual Model
  - Tables (Nouns)
  - Relationships (Connective sentences/phrases)

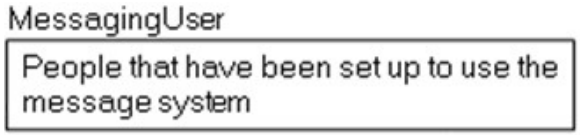
- Validate (Test) the model against the requirements
  - If YES (they do) Move On To Logical Model
  - If NO, Keep Trying

Example Model

- The next slides walk us through the stages of a small modeling project
- High Level Requirements:
  - Messaging system for conference attendees
  - Can send message to everyone or one person
  - Messages can have multiple topics chosen from a fixed set of topics (but you can create a custom topic as well)
  - No duplicate messages in the same hour
  - Attendees can be connected to other attendees
  - Messages must not include bad language or “hate” speech

Example Model

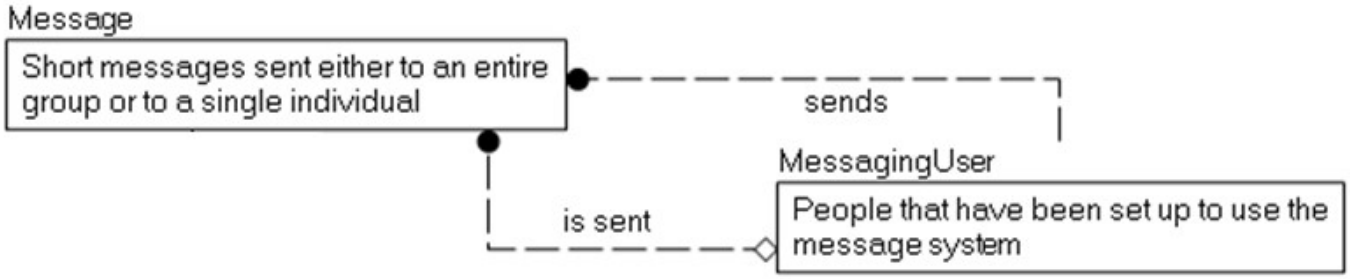
– Messaging system for conference attendees



Example Model

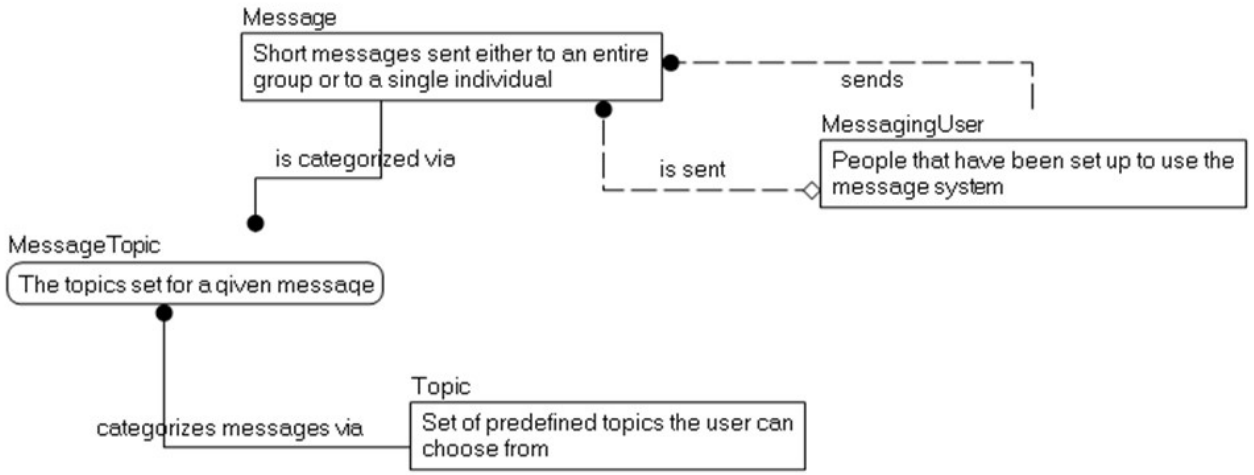
– Messaging system for conference attendees

– Can send <u>message</u> *to everyone or one person*



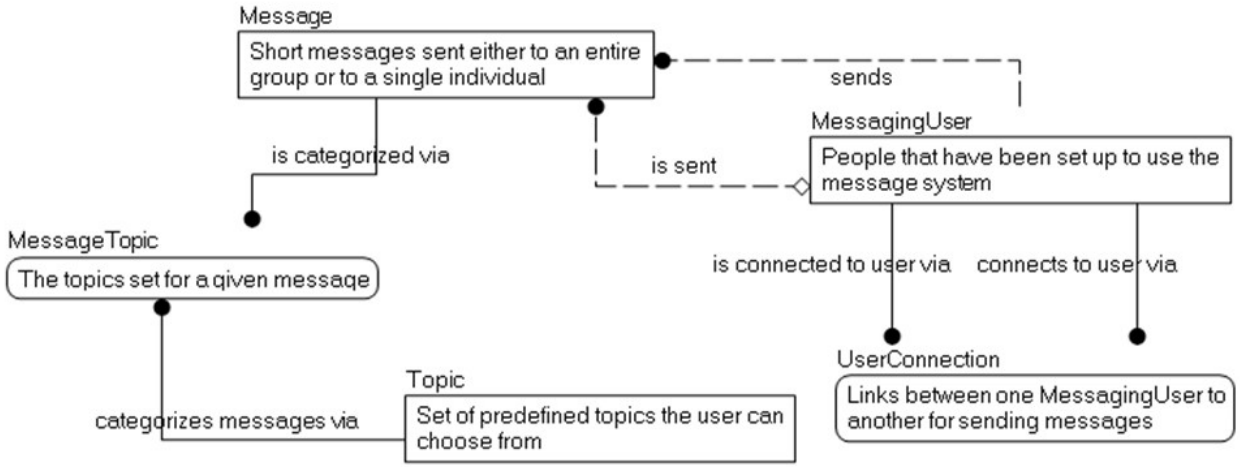
Example Model

- Messaging system for conference attendees
- Can send message to everyone or one person
- *Messages can have multiple* topics chosen from a fixed set of topics (*but you can create a* custom topic as well)



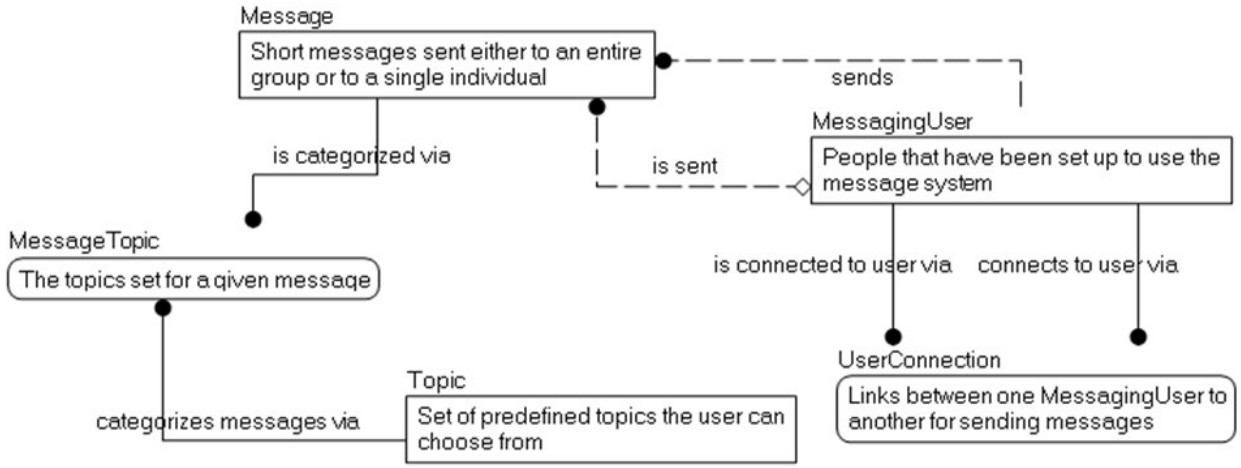
Example Model

- Messaging system for conference attendees
- Can send message to everyone or one person
- Messages can have multiple topics chosen from a fixed set of topics (but you can create a custom topic as well)
- No duplicate messages in the same hour
- Attendees *can be connected to other* attendees



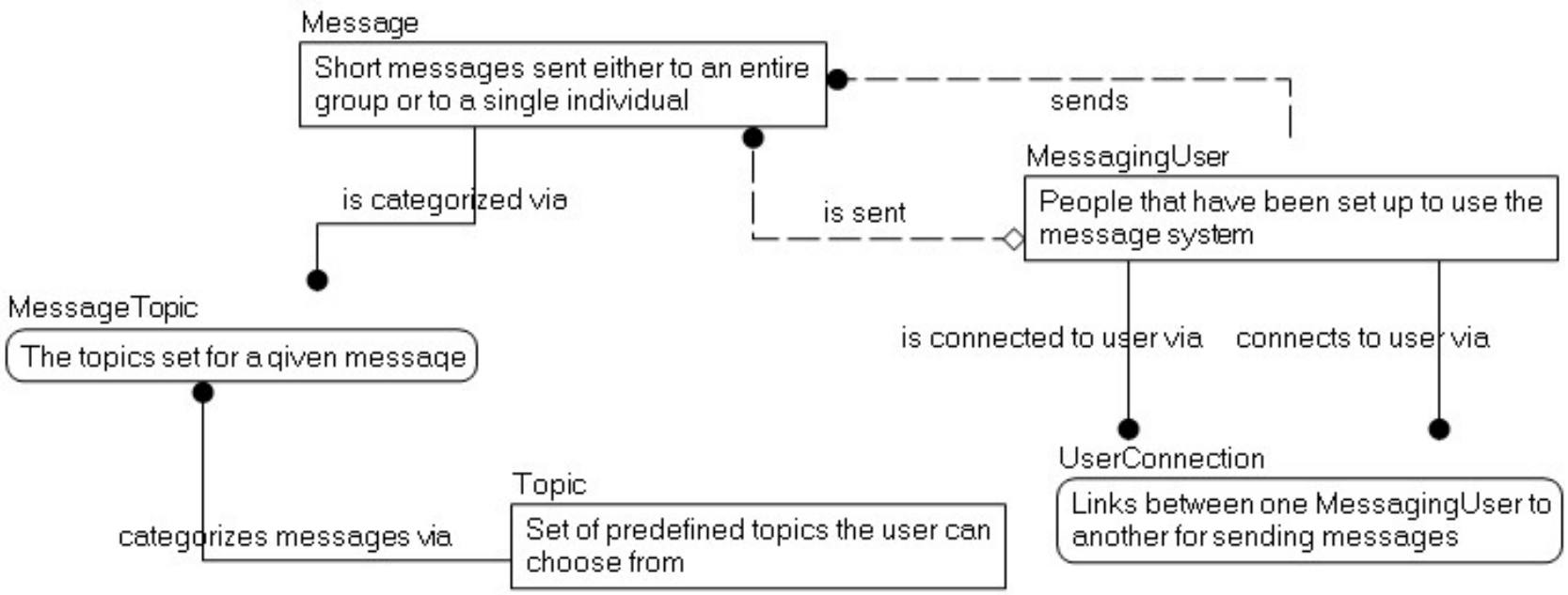
Example Model

- Messaging system for conference attendees
- Can send message to everyone or one person
- Messages can have multiple topics chosen from a fixed set of topics (but you can create a custom topic as well)
- No duplicate messages in the same hour
- Attendees can be connected to other attendees



- Messages must not include bad language or “hate” speech

Conceptual Model



- Tables, with descriptions and relationships
- Test against the requirements (Looks like the process we just went through, with edits)

Continue to the Logical Model

- Columns (With legal values-domain)
- Uniqueness conditions (Keys)
- System Predicates

- Basically: Understand what is good data, what is NOT good data, and what probably isn't good data.
- It doesn't matter how any of this can be implemented with or without SQL

- Continue to test your design against the requirements
  - If YES (they do), start moving towards building tables (Woo hoo!)
  - If NO, keep refining the model

Tip – Start Out Naming Consistently

- There are lots of naming standards out there.
- Most importantly, name consistently
  - Names should be as specific as possible
  - Data should rarely be represented in the name
  - Don't use the English language as a guide (I before E except after C, unless?)
- Early in the process, avoid abbreviations unless it is very apparent to everyone
  - When you build the physical model, if you must abbreviate, use a data dictionary to make sure abbreviations are always the same
- Tables
  - Never prefixed to says that this is a table (~~tbl_~~TableName)
  - Singular or Plural (either one)
    - I prefer singular

Column Naming

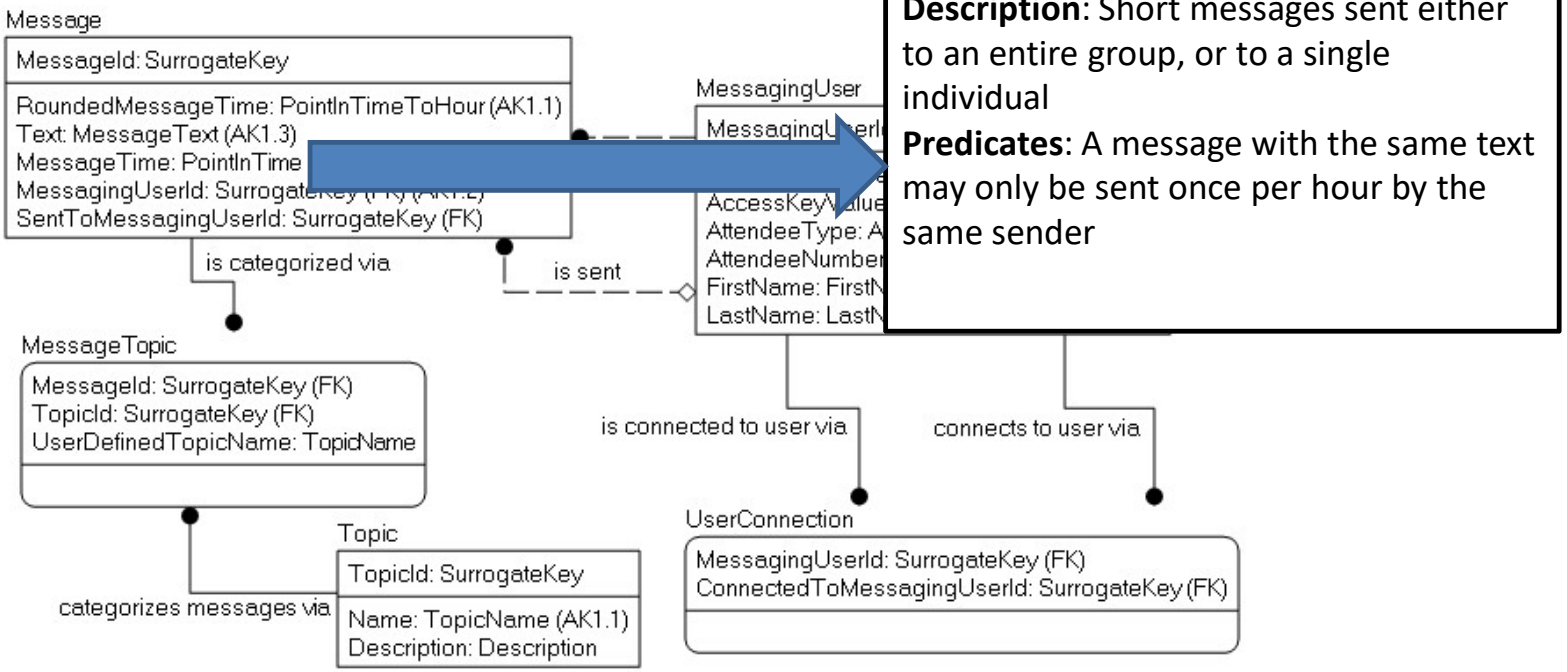
- Column names should be singular - Columns should (at least eventually) represent a scalar value
- Avoid overly specific prefixes/suffixes
- Follow a standard format for names
- An example that I have seen documented in various places (often attributed to ISO 11179) is to have names that include something along the following:
  - RoleName – Optional for when you need to explain the purpose of the attribute
  - Attribute – The primary purpose of the column being named. Optionally can be omitted, meaning it refers to the entity
  - Classword – a suffix that identifies the usage of the column, in non-implementation specific terms
  - Scale – Optional to tell the user what the scale of the data is, like minutes, seconds, dollars, euros, etc

  - RoleName_Attribute_Classword_Scale

Column Naming Examples

- **Name** - a textual string that names the row value, but whether or not it is a varchar(30) or nvarchar(128) is immaterial (prefix is implied. Example Company.Name)
- **UserName** - a more specific use of the name classword that indicates it isn't a generic usage
- **AdminstratorUserName** – A user name, but specifically for the admin user.
- **PledgeAmount** - an amount of money (using a numeric(12,2), or money, or any sort of types)
- **PledgeAmountEuros** - an amount of money (using a numeric(12,2), or money, or any sort of types), but with an atypical scale
- **TickerCode** - a short textual string used to identify a ticker row
- **EndDate** - the date when something ends. Does not include a time part
- **SaveTime** - is the point in time when the row was saved
- **WaitingOnMailPickupFlag** – the mail delivery person has put the flag up, and your mail is here (your IOT device has discovered and told you)

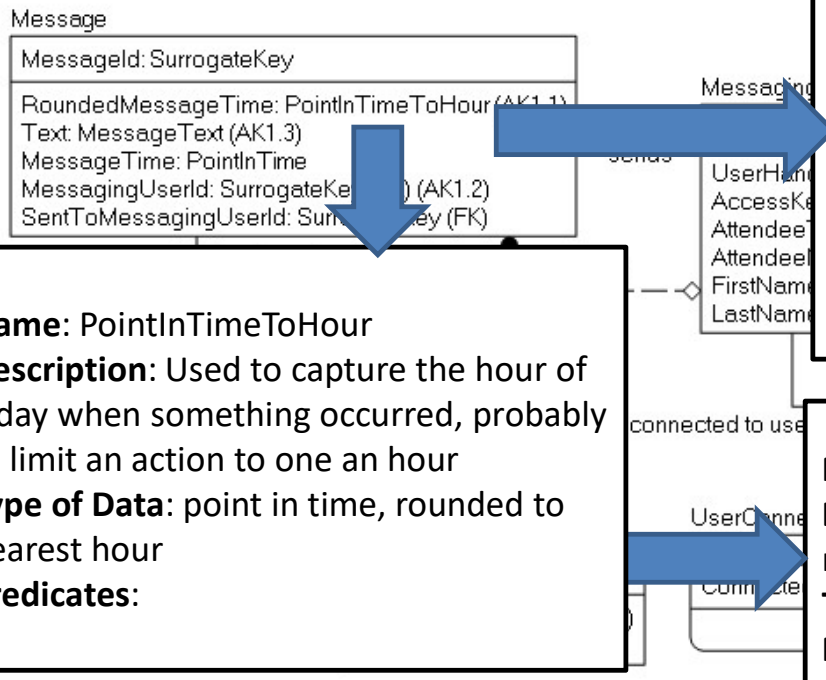
Logical Mo



Name: Message
Description: Short messages sent either to an entire group, or to a single individual
Predicates: A message with the same text may only be sent once per hour by the same sender

- The “ideal” version of the database requirements
- Implementation non-specific

Logical Model Ba



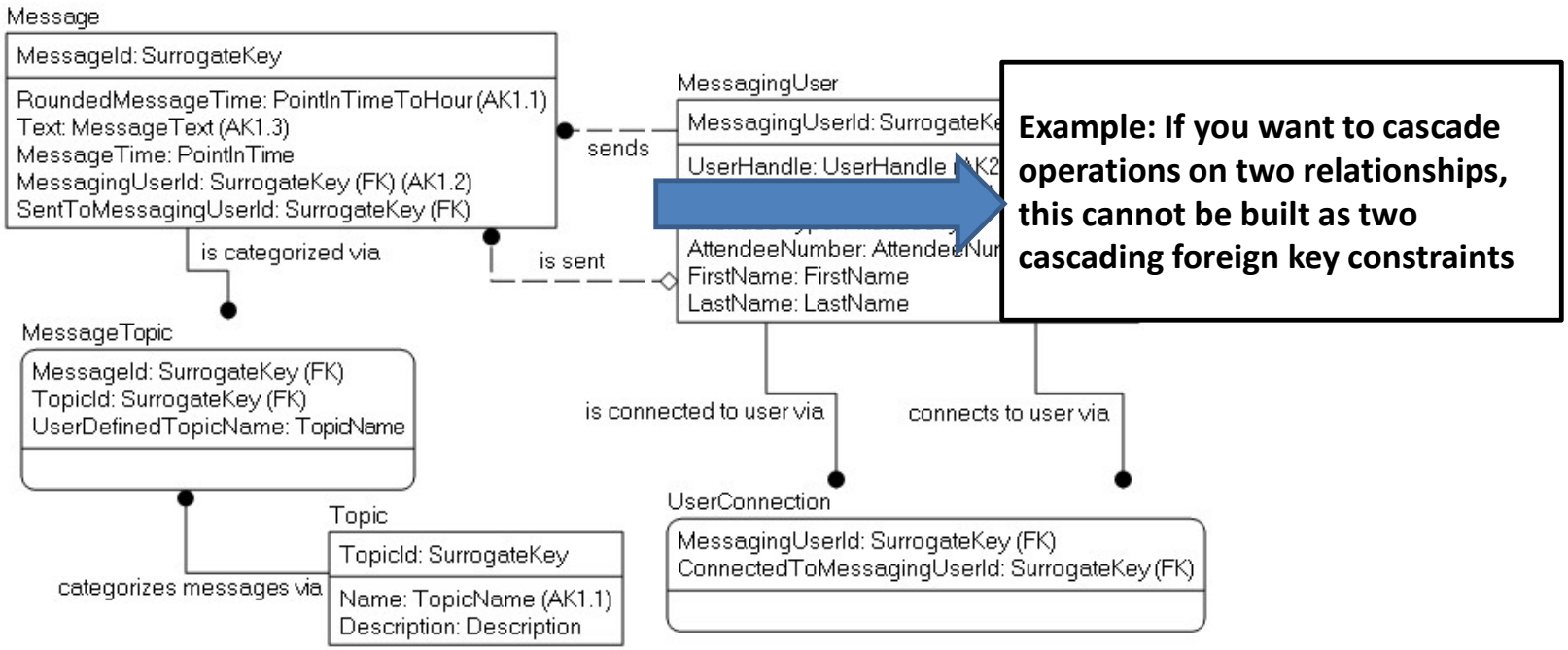
Name: MessageText
Description: Used to present messages to users
Type of Data: unformatted values, less than or equal to 200 characters
Predicates: Must not be empty or Null, Must not include hate speech, bad language, etc.

Name: PointInTimeToHour
Description: Used to capture the hour of a day when something occurred, probably to limit an action to one an hour
Type of Data: point in time, rounded to nearest hour
Predicates:

Name: SurrogateKey
Description: Used as a stand-in for the natural key
Type of Data: Any
Predicates: The data should be of some format that is small, yet still sufficiently capable of being unique

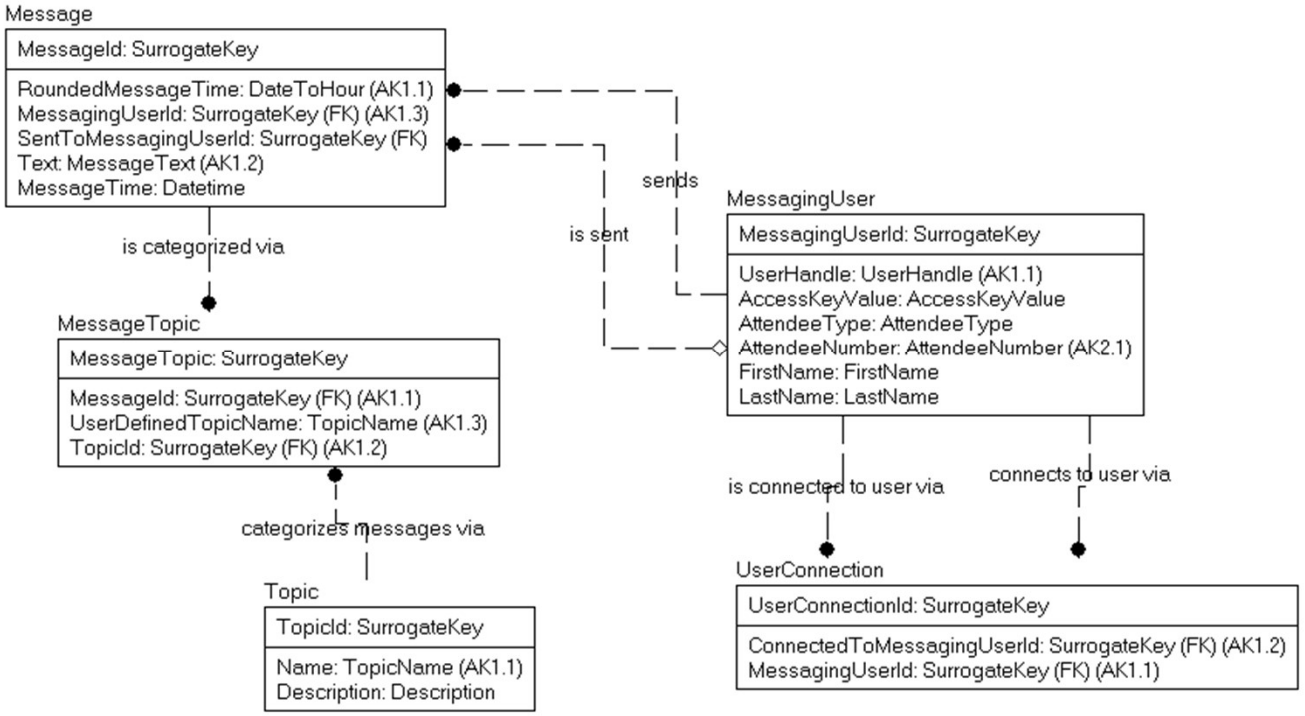
- Domains instead of Data types
  - Domains should define the domain of values tha
  - Data types will specify part of how the domain is
  - Check constraints, triggers etc may also be needed to do the implementation

Logical Model Basics - Relationships



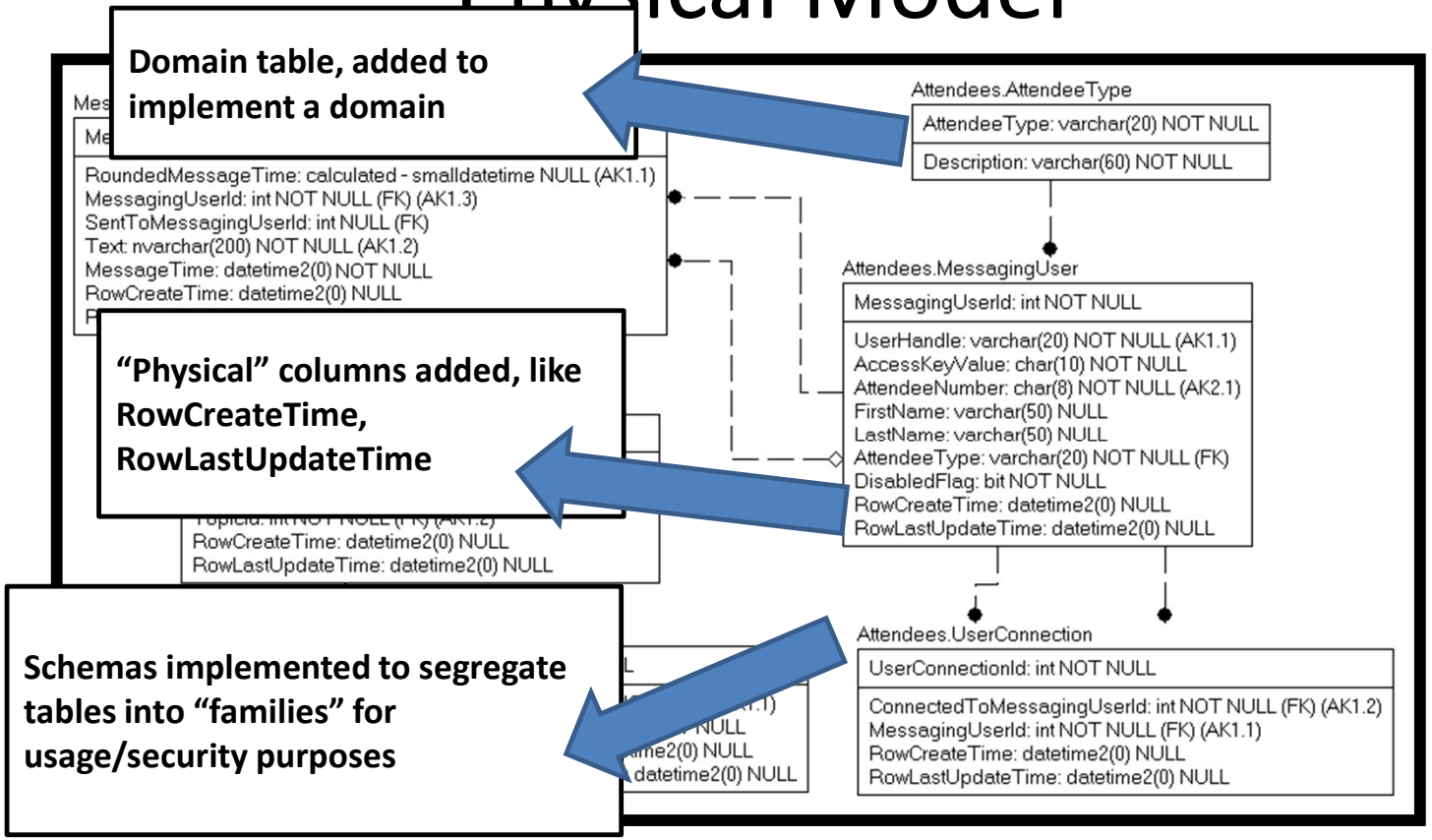
- Document all relationship requirements, no matter how you will need to implement them

Surrogate Keys on all Tables?



- This has a few effects, positive and negative
  - Positive: it is very similar to what UI tools often expect
    - Keeps personal information out of keys (that might end up on a web page call!)
  - Negative: Uniqueness and lineage are more difficult to see

Physical Model

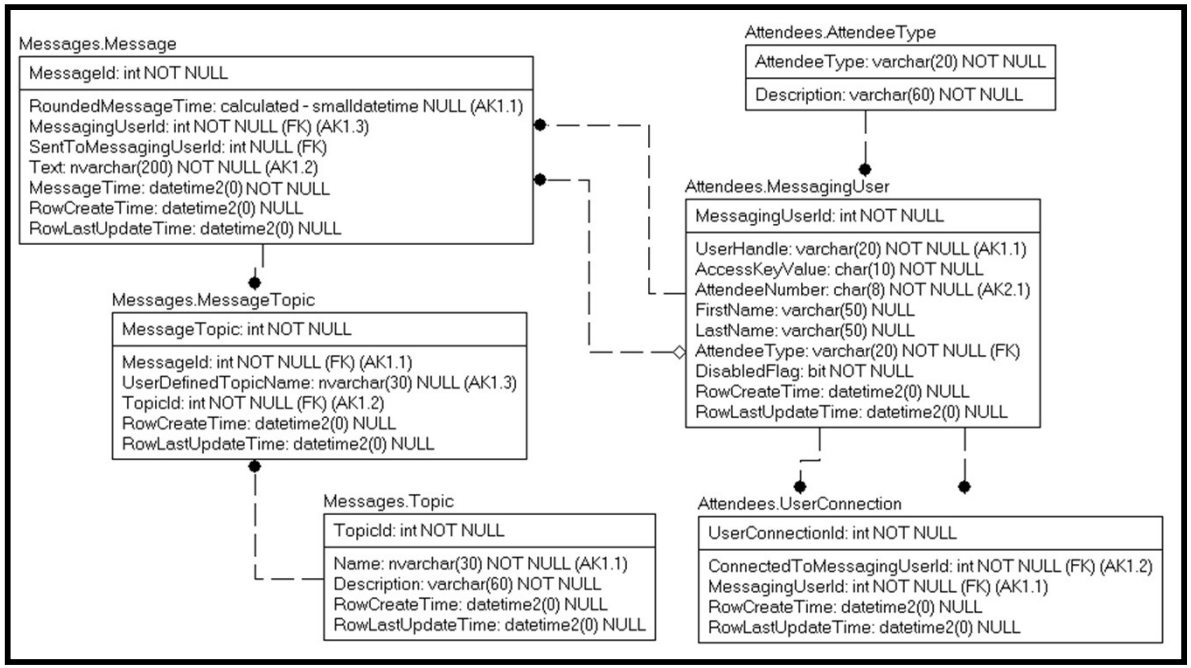


- Some domains become tables
- Best data types chosen

Document

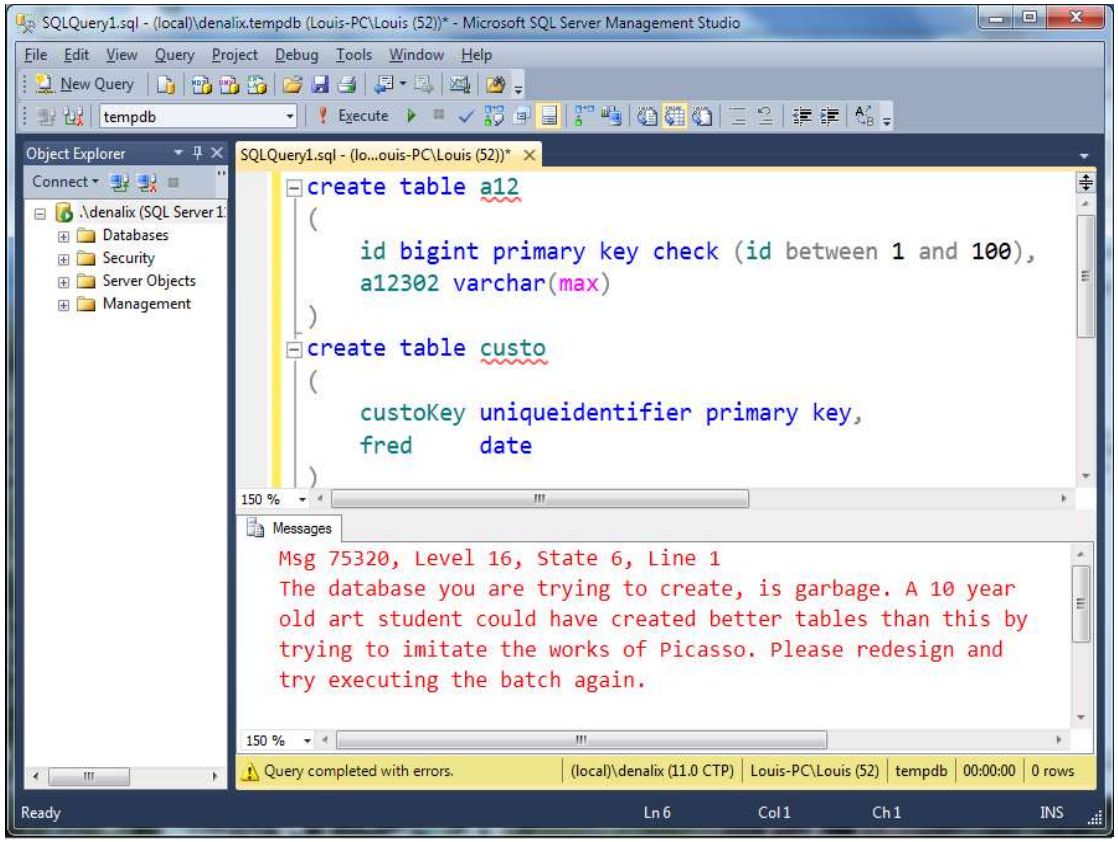
- Every table and column should have a succinct description
- Then expand complex situations with documents, examples, etc, with the knowledge that will likely not be maintained…
- Try to avoid too many examples, as data can change
- Ideally, the documentation will be accessible by programmers and end users alike

Are we done finished designing yet?



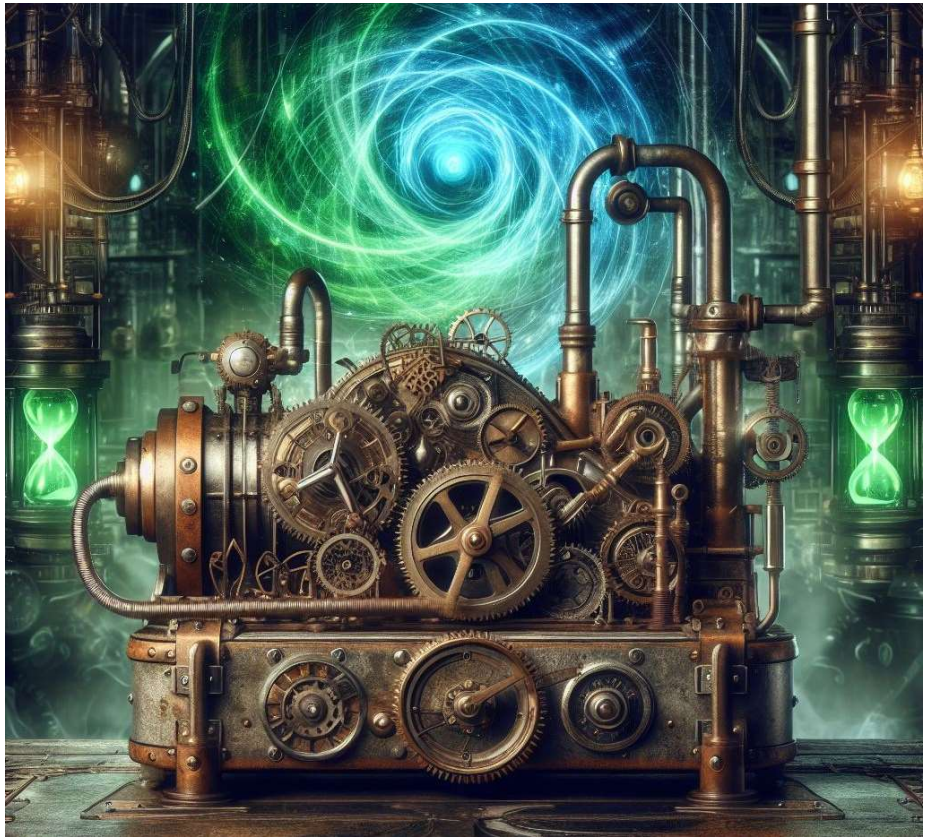
- Perhaps
  - At this point, it is important to check your model against a standard
  - For a relational database, the standards are the Normal Forms

New SQL Server ’22 AI Assisted Design Warning Message



- Press Ctrl-Y, Ctrl-R
- Not yet, maybe some day when AI can read your requirements, your data model, and your mind.

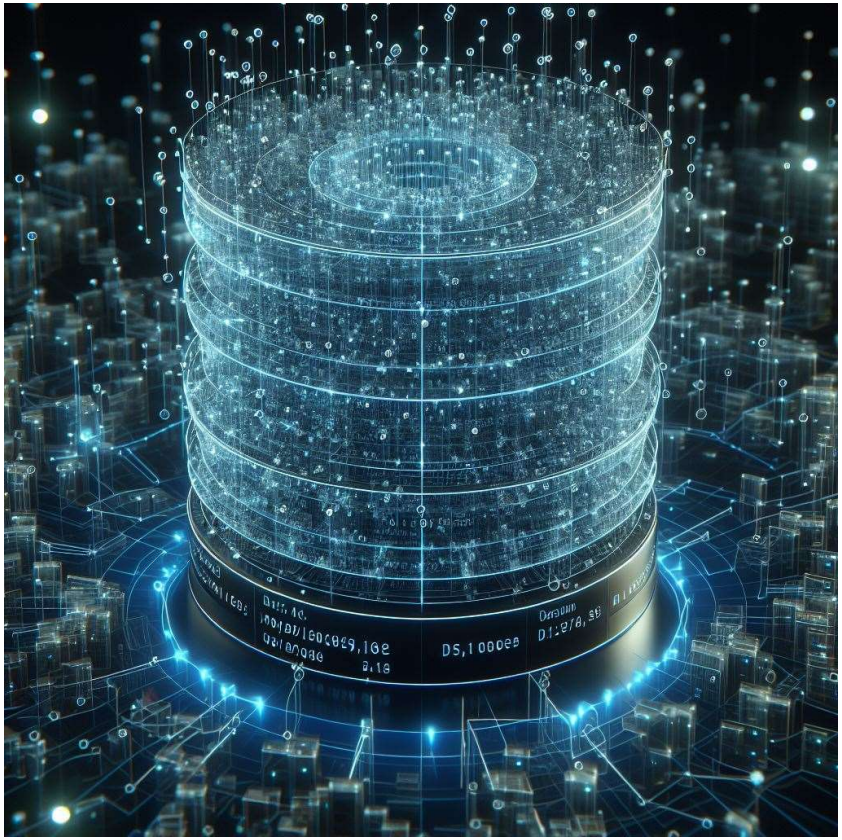
Skipping ahead in time



- Pretend you have chosen a database management system…
  - Which means you have requirements, a budget, meetings scheduled, your whiteboards covered in ink and marked “do not erase” on every corner, lunch ordered in for the next month, and you basically have your ducks all sitting in a row ready to get quacking

- *And I don’t care which one*
- *And I don’t care why you chose it*
  - *I do hope you can change your mind if necessary.*

Now what?



Controversial opinion time…

- It is time to iteratively design…
- It doesn't matter if your system requires structured, semi-structured, or unstructured data
- You want to know the structure…
  - Even if the structure is strictly "no structure"
  - I can't imagine a true.. Zero structure database

Structure because… you have to use the data

And trust the data…

Normal Forms/Normalization

- A process to understand the shape of your data
  - Probably to work with a relational engine
  - And even if not, to help you understand how your data is used
- Specified as a series of forms that signify compliance
- Practically speaking:
  - A set of standards to work towards along the way
  - Somewhat done instinctively
  - Important to understand even if you cannot achieve perfection

  - Utter nonsense if you do not understand the requirements!

Why I am talking about normalization?

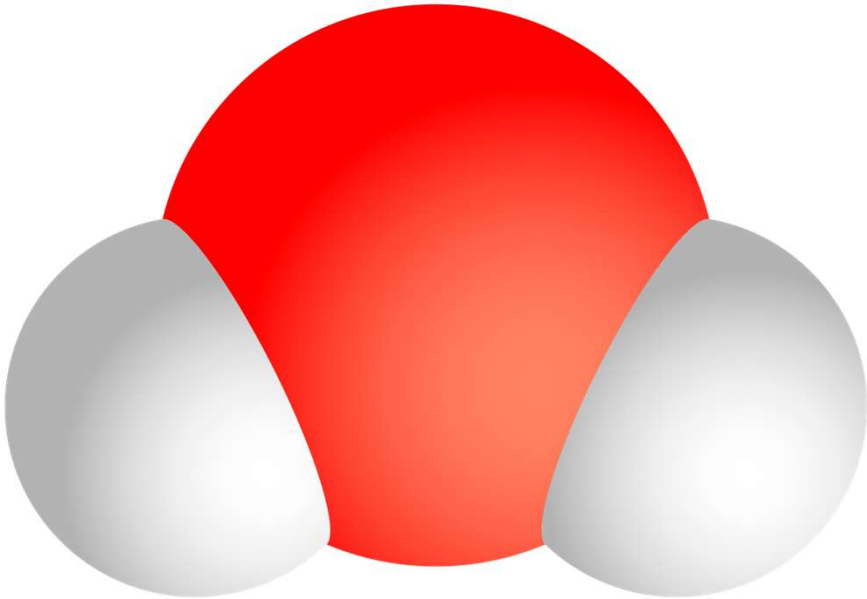
Because I used to do reporting ETL for a living…

Normalization Main Programming Purpose

- Eliminate DML modification anomalies and partial column operations
- Meaning, you need to update one piece of information:
  - One entire piece of data is changed
  - No other piece of data *needs* changing
- Normalization is focused around protecting the integrity of your data by
  - Making every column and table an *atomic* unit
  - Making modifications *atomic* operations

Atomicity

- At the lowest level possible without losing the original characteristics
  - Similar to context of physics as we know it in the 21st century
  - Break down H^20 into Hydrogen and Oxygen, no big deal
  - Break down Hydrogen it becomes a different sort of matter and you are going to need a new laboratory



Example: Go get Aunt Granny's Mud Pie Recipe

You know to go to the Recipe Box!

Mud Pie

A classic twist on a timeless classic.

Often referred to as sweet revenge pie, this pie is awful and inedible, but at least not lethal. Easy to make, using ingredients you almost certainly have around your house... around the outside of it anyhow.

Ingredients

2 cups dirt
2 cups water
1 cup sand
1 frozen pie crust
1 cup chopped nuts (acorns will do)

Directions

1. Put the 2 cups of the dirt and 1 cup of the water in a bowl and stir (be certain that the dirt is actually dirt. You don't want to make that mistake)

2. Add some of the sand, again making sure you have pure sand.

3. Add some more of the water, and the chopped nuts to the mixture and continue to stir.

4. Once the mixture has a wet firm feel to it, pack it into the pie crust

5. Put on campfire until it is a little singed on the top. Let it cool and serve it to friend who broke the head off your Han Solo action figure just to see how easy it would be and now you have a headless Han and it makes you so mad.

There is a reason we don’t use scrolls anymore…

Whose parent didn’t do this?

Mud Pie

A classic twist on a timeless classic.

Often referred to as sweet revenge pie, this pie is awful and inedible, but at least not lethal. Easy to make, using ingredients you almost certainly have around your house... around the outside of it anyhow.

Ingredients
2 cups dirt
2 cups water
1 cup sand
1 frozen pie crust
1 cup chopped nuts (acorns will do)

Directions
1. Put the 2 cups of the dirt and 1 cup of the water in a bowl and stir (be certain that the dirt is actually dirt. You don't want to make that mistake)

2. Add some of the sand, again making sure you have pure sand.

3. Add some more of the water, and the chopped nuts to the mixture and continue to stir.

4. Once the mixture has a wet firm feel to it, pack it into the pie crust

5. Put on campfire until it is a little singed on the top. Let it cool and serve it to friend who broke the head off your Han Solo action figure just to see how easy it would be and now you have a headless Han and it makes you so mad.

Mud Pie

A classic twist on a timeless classic.

Often referred to as sweet revenge pie, this pie is awful and inedible, but at least not lethal. Easy to make, using ingredients you almost certainly have around your house... around the outside of it anyhow.

Ingredients

2 cups dirt

2 cups water

1 cup sand

1 frozen pie crust

1 cup chopped nuts (acorns will do)

Directions

1. Put the 2 cups of the dirt and 1 cup of the water in a bowl and stir (be certain that the dirt is actually dirt. You don't want to make that mistake)

2. Add some of the sand, again making sure you have pure sand.

3. Add some more of the water, and the chopped nuts to the mixture and continue to stir.

4. Once the mixture has a wet firm feel to it, pack it into the pie crust

5. Put on campfire until it is a little singed on the top. Let it cool and serve it to friend who broke the head off your Han Solo action figure just to see how easy it would be and now you have a headless Han and it makes you so mad.

And then as you are making one piece of paper per ingredient, you ask the question…

Does this make any sense for what my I need?

Mud Pie

A classic twist on a timeless classic.

Often referred to as sweet revenge pie, this pie is awful and inedible, but at least not lethal. Easy to make, using ingredients you almost certainly have around your house... around the outside of it anyhow.

Ingredients

2 cups dirt

2 cups water

1 cup sand

1 frozen pie crust

1 cup chopped nuts (acorns will do)

Directions

1. Put the 2 cups of the dirt and 1 cup of the water in a bowl and stir (be certain that the dirt is actually dirt. You don't want to make that mistake)

2. Add some of the sand, again making sure you have pure sand.

3. Add some more of the water, and the chopped nuts to the mixture and continue to stir.

4. Once the mixture has a wet firm feel to it, pack it into the pie crust

5. Put on campfire until it is a little singed on the top. Let it cool and serve it to friend who broke the head off your Han Solo action figure just to see how easy it would be and now you have a headless Han and it makes you so mad.

But a database can make great use of this!

Keep Your Design Simple

- Meet the user's requirements to the Normal Form's requirements
  - One table per user's concept
  - One column per user's attributes of that concept
- Things it is not:
  - Religion
  - Complicated
  - Easy
- Proper designs answer all user queries in straightforward SQL

Normal Forms Overview - 1NF

- Basic shaping of data for the engine
- Data broken down to its lowest form
  - Column Values are atomic
  - No duplicate rows
  - All rows must represent the same number of values (Sometimes referenced as “no repeating groups”)

Your database may not be in 1NF if..

- You have string data that contains separator-type characters. Example, patterns using commas, pipes, tildes, etc (even spaces can qualify)
- Bitmasks (ew!)
- Attribute names with numbers at the end
  - Payment1, Payment2, …
- Tables with no or poorly defined keys
  - `CustomerId int identity PRIMARY KEY`

First Normal Form Example 1

- Requirement: Table of school mascots

MascotId	Name	Color	School
1	Smokey	Black/Brown	UT
112	Smokey	Black/White	Central High
4567	Smokey	Smoky	Less Central High
979796	Smokey	Brown	Southwest Middle

- To truly be in the spirit of 1NF, some manner of uniqueness constraint needs to be on a column that has meaning
- It is a good idea to unit test your structures by putting in data that looks really wrong and see if it stops you, warns you, or something!

Uniqueness isn't always naturally attainable

- Design for all possible cases, even if you will not be able to implement solely in SQL Server
- Some common uniqueness requirements
  - Bulk Uniqueness – Inventory of Canned Goods, Parts, etc.
    - One row per type of object
  - Selective Uniqueness – Unique when filled in: Driver's License Number, SSN/Work Number, Union Card Number
    - Use a unique filtered index (2008+), indexed view (2000- 2005) or triggers (earlier) to implement
  - Likely Uniqueness – Data condition where a human should make the decision about uniqueness: Employee names; Customer Information, etc.

- Bottom Line: Design all uniqueness situations, enforce as much as possible (and reasonable).

First Normal Form Example 2

- Requirement: Store information about books

BookISBN	BookTitle	BookPublisher	Author
111111111	Normalization	Apress	Louis
222222222	T-SQL	Apress	Michael
333333333	Indexing	Microsoft	Kim
444444444	DB Design	Apress	Jessica, and Louis
444444444-1	DB Design	Apress	Louis

- What is wrong with this table?
  - Lots of books have > 1 Author.
- What are common way users would "solve" the problem?
  - Any way they think of!
- What's a common programmer way to fix this?

First Normal Form Example 2

- Add a repeating group?

```
BookISBN    BookTitle       BookPublisher   …
==========  ------------    --------------
111111111   Normalization   Apress          …
222222222   T-SQL           Apress          …
333333333   Indexing        Microsoft       …
444444444   Design          Apress          …

Author1     Author2     Author3
----------- ----------- -----------
Louis
Michael
Kim
Jessica     Louis
```

- What is so wrong?

First Normal Form Example 2

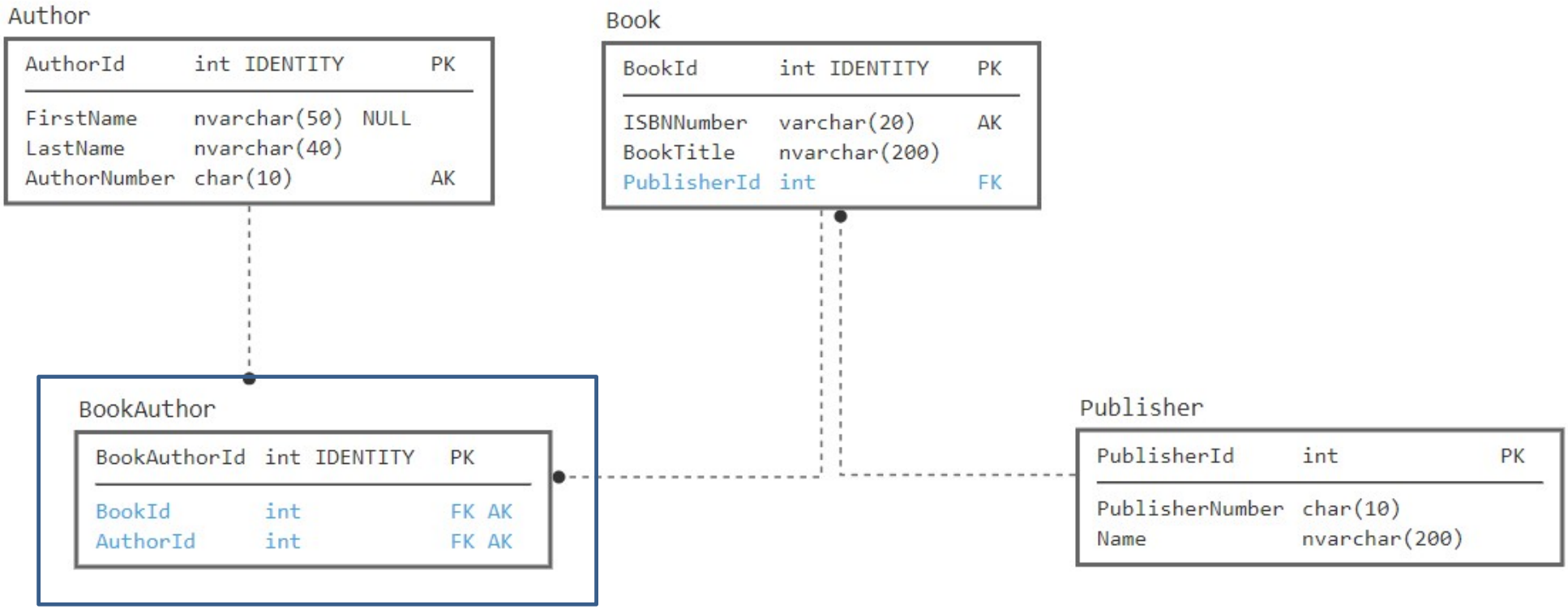
- Two tables!

BookISBN	BookTitle	BookPublisher
111111111	Normalization	Apress
222222222	T-SQL	Apress
333333333	Indexing	Microsoft
444444444	DMV Book	Simple Talk

BookISBN	Author	*ContributionType*
111111111	Louis	***Principal Author***
222222222	Michael	***Principal Author***
333333333	Kim	***Principal Author***
444444444	Jessica	***Contributor***
444444444	Louis	***Principal Author***

- And it gives you easy expansion

First Normal Form Example 2



First Normal Form Example 2.1

```
Email1    Email2    Email3
--------- --------- -----------
```

But inevitably, requirements change, and the structures are inadequate… and unchangeable… So you do something like this (and yes, I had to do this recent enough it still makes me a bit ill…that db has more columns than this!):

```
Email1Status Email1Type   Email1PrivateFlag
------------ ------------ --------------------

Email2Status Email2Type   Email2PrivateFlag
------------ ------------ --------------------

Email3Status Email3Type   Email3PrivateFlag
------------ ------------ --------------------
```

First Normal Form Example 3

- Requirement: Store users and their names

UserId	UserName	PersonName
1	Drsql	Louis Davidson
2	Kekline	Kevin Kline
3	Datachix2	Audrey Hammonds
4	PaulNielsen	Paul Nielsen

- How would you search for someone with a last name of Nielsen? David? Davidson?
- What if the name were more realistic with Suffix, Prefix, Middle names?

First Normal Form Example 3

- Break the person's name into individual parts

UserId	UserName	PersonFirstName	PersonLastName
1	Drsql	Louis	Davidson
2	Kekline	Kevin	Kline
3	Datachix2	Audrey	Hammonds
4	PaulNielsen	Paul	Nielsen

- This optimizes the most common search operations
- It isn't a "sin" to do partial searches on occasion, just don't make it habitual:
  - I know the last name ended in "son" or "sen"
- If you also commonly need the full name available, let the engine manage this using a calculated column:
  - PersonFullName AS CONCAT(PersonFirstName + ' ', PersonLastName)

Normal Forms Overview – 2NF, 3NF and Boyce-Codd (BCNF) Normal Forms

- Eliminate incorrect data dependencies in your tables
  - All attributes are either a key, or fully dependent on a key (the whole key, and nothing but the key)
  - Violations usually manifest themselves as multiple column, row-wise repeating groups
- In other words…
  - All keys for a table are identified
  - All columns describe that “thing” that the table is modeling

Intrarow Dependency

- If you can determine the value of one attribute X given a different attribute Y, then Y is functionally dependent on X. X is considered the determinant.

  Example:

X	Y	Z
1	1	2
2	2	4
3	2	4

- Assuming this is the entire known universe. X is unique key:
  - Y and Z are functionally dependent on X
  - But, is Z is functionally dependent on Y (or vice versa)?

Keeping it simple

- I want to demonstrate how the normal forms help you without getting too deep
- Quickly:
  - 2NF deals with partial-key dependencies
  - 3NF deals with non-key dependencies
  - BCNF deals with cases where an attribute is dependent on both key and non-key values

Your database may not be BCNF if..

- There are multiple columns with the same prefix
- Multiple tables have the exact same complex columns
  - Example: Three tables have MessageSentDate, MessageText columns
- There are repeating groups of data
  - Particularly if > 1 column shows the repeats
- There are triggers with modification statements
  - Some trigger use to trigger workflow can make sense, but too often it is a matter or maintaining summary/status data

2NF Example

- Requirement: Defines the types of car(s) that a driver likes

Driver	Car Style	Height	EyeColor	MaxWeight
Louis	Station Wagon	6′0″	Blue	2900
Louis	Hatchback	6′0″	Blue	2500
Ted	Coupe	5′8″	Brown	2200

- Check the attributes against the meaning of the table
  - Height and EyeColor define the attributes of the driver alone
  - MaxWeight? The weight of vehicle for that style it is acceptable for the style of car? Or the driver? Naming is important!
    - Defined as: Vehicle weight for car style

2NF Example 1

- Solution: 3 independent tables, 1 for driver, 1 for driver's car style preference, 1 for driver and car style

Driver	Car Style
Louis	Station Wagon
Louis	Hatchback
Ted	Coupe

Driver	Height	EyeColor
Louis	6′0″	Blue
Ted	5′8″	Brown

Car Style	MaxWeight
Station Wagon	2900
Hatchback	2500
Coupe	2200

3NF Example

- Requirement: Driver registration for rental car company

Driver	PrimaryVehicleOwned	Height	EyeColor	WheelCount
Louis	Hatchback	6′0″	Blue	4
Ted	Coupe	5′8″	Brown	4
Rob	Tractor trailer	6′8″	NULL	18

- Column Dependencies
  - Height and EyeColor, check
  - VehicleOwned, check
  - WheelCount, <buzz>, driver's do not have wheelcounts

3NF Example

- Two tables, one for driver, one for type of vehicles and their characteristics

Driver	PrimaryVehicleOwned (FK)	Height	EyeColor
Louis	Hatchback	6′0″	Blue
Ted	Coupe	5′8″	Brown
Rob	Tractor trailer	6′8″	NULL

VehicleOwned	WheelCount
Hatchback	4
Coupe	4
Tractor trailer	18

BCNF Example

- Requirement: Define cars ready to be rented

VehicleId	VehicleModel	ParkedInArea
XCV101	Ford Escape 2022	A
XR3DA1	Toyota Corolla 2023	A
TRE302	Dodge RAM Pickup 2019	B
E20E32	Ford Lightning 2024	B

  - Note: Consider VehicleModel to actually be in 1NF for this example. In reality it would be a multi-part key in a table defining types of vehicles.
- Column Dependencies
  - VehicleModel - Yes, this would be dependent on the actual vehicle
  - ParkedInArea – Yes, this would be dependent on the vehicle
  - Hence, the table is in 3NF

BCNF Example – But…

- Dig a little deeper:

VehicleId	VehicleModel	ParkedInArea
XCV101	Ford Escape 2022	A
XR3DA1	Toyota Corolla 2023	A
TRE302	Dodge RAM Pickup 2019	B
E20E32	Ford Lightning 2024	B

- Note that all cars are in area A, and all trucks are in area B.
  - Assuming this is an actual business rule, this table is not in BCNF.
  - Otherwise it is a proper design

BCNF Example 3

- Three tables. One to define the type of vehicle, another to define the parking location, and another for the actual vehicles.

VehicleId	VehicleModel
XCV101	Ford Escape 2022
XR3DA1	Toyota Corolla 2023
TRE302	Dodge RAM Pickup 2019
E20E32	Ford Lightning 2024

VehicleModel	VehicleType
Ford Escape 2022	Car
Toyota Corolla 2023	Car
Dodge RAM Pickup 2019	Pickup Truck
Ford Lightning 2024	Pickup Truck

VehicleType	ParkingLocation
Car	A
Pickup Truck	B

Important!

- Don’t forget to get whatever you were supposed to pick up on the way home.
- But more importantly:

  Remember that ALL of this is requirements driven.

Natural Keys

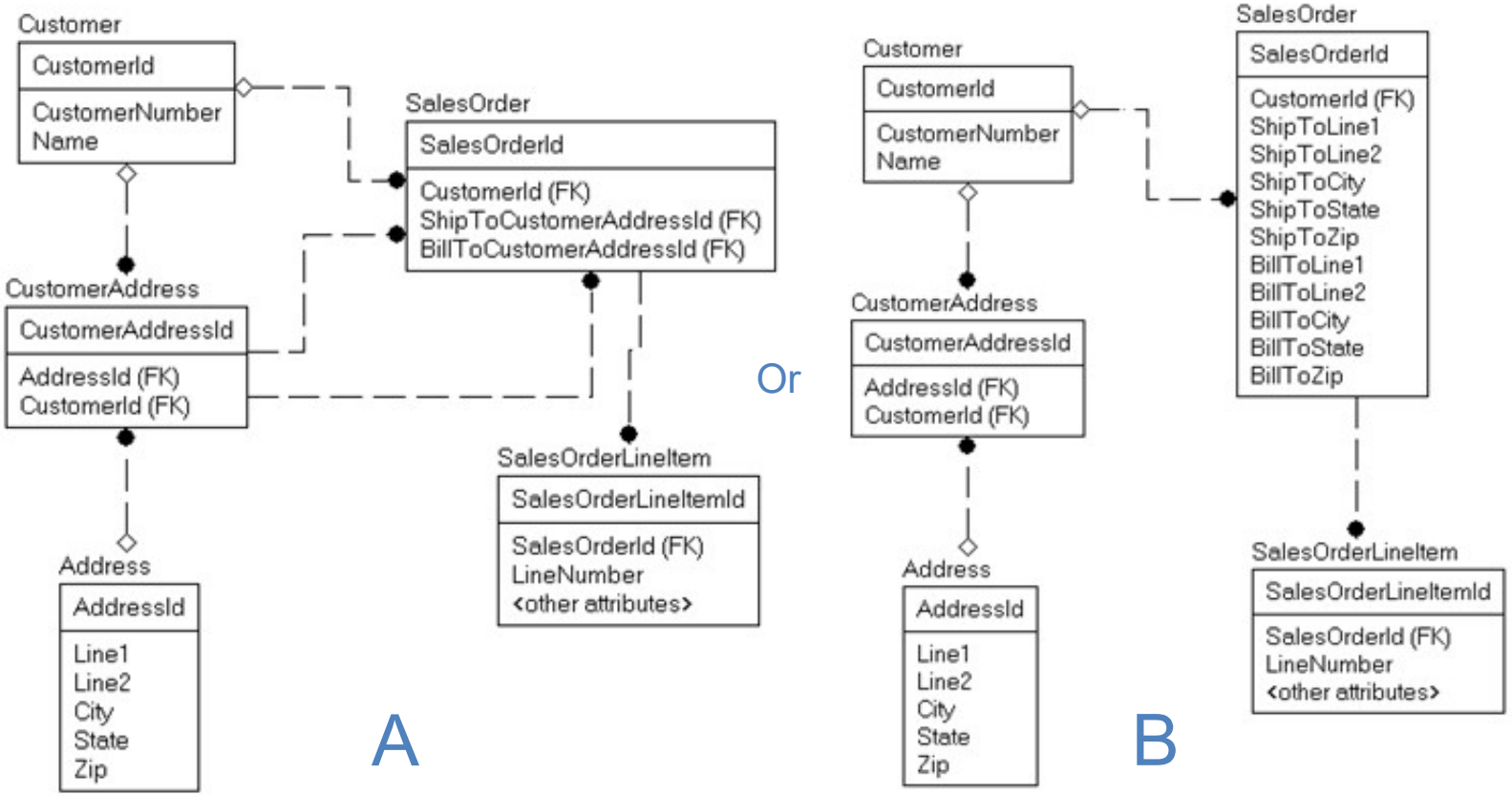
```
create table driversLicense
(
        driversLicenseStateCode char(2) not null,
        driversLicenseNumber varchar(40) not null,
        classCode char(2) not null,
        PRIMARY KEY (driversLicenseStateCode, driversLicenseNumber)
)
create table employee
(
        employeeNumber char(10) not null primary key
        driversLicenseStateCode char(2) null,
        driversLicenseNumber varchar(40) null,
        firstName varchar(30) not null,
        middleName varchar(30) null,
        lastName varchar(30) not null,
        UNIQUE (driversLicenseStateCode, driversLicenseNumber)
        FOREIGN KEY(driversLicenseStateCode, driversLicenseNumber)
                REFERENCES driversLicense(driversLicenseStateCode, driversLicenseNumber)
)
```

Composite Keys – The Surrogate Effect

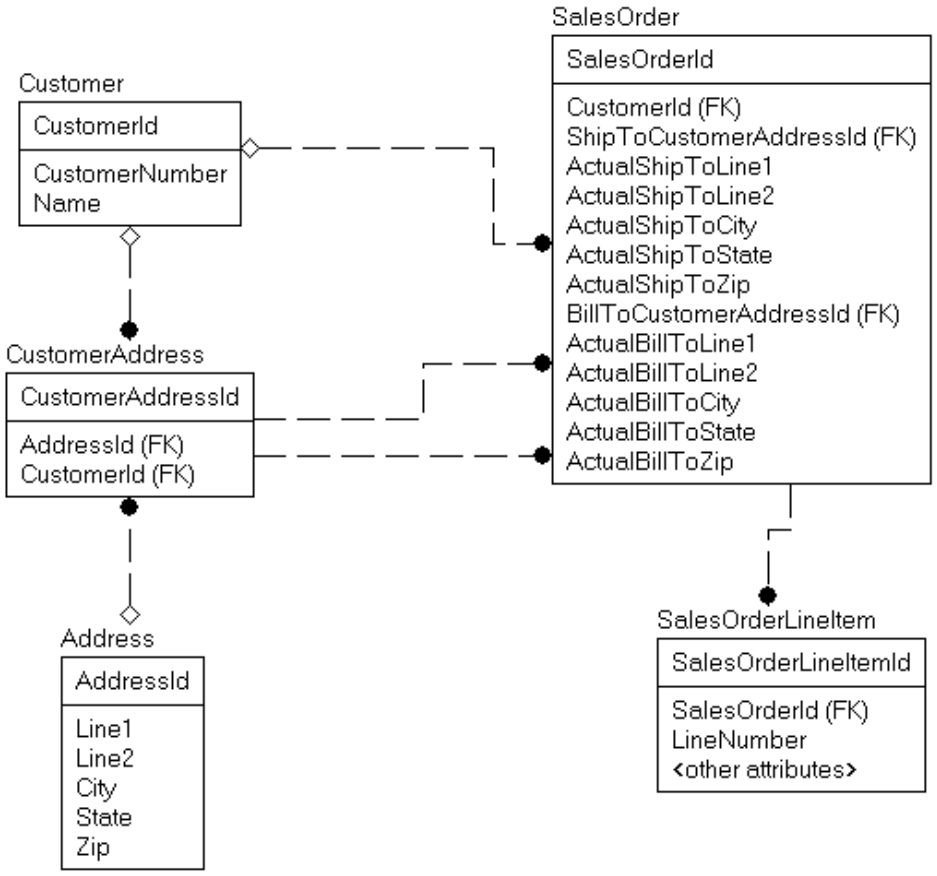
This is a mathematically equivalent representation

```
create table driversLicense
(
        driversLicenseId int primary key --might be identity or even GUID
        driversLicenseStateCode char(2),
        driversLicenceNumber varchar(40),
        classCode char(2),
        unique (driversLicenseStateCode, driversLicenseNumber)
)
create table employee
(
        employeeId int identity primary key, -might be identity or even GUID
        employeeNumber char(10) not null UNIQUE,
        driversLicenseId int null UNIQUE
                                REFERENCES driversLicense(driversLicenseId)
        firstName varchar(30) not null,
        middleName varchar(30) null,
        lastName varchar(30) not null
)
```

Quiz – Which Model is Correct?



Quiz – Answer “It depends…perhaps both”



Fourth and Fifth Normal Forms

- Deals with the relationships within key attributes
- In practical terms, it deals when a single row actually has multiple meanings that are not immediately obvious
- What makes it challenging that the same table may or may not be in Fourth or Fifth Normal Form depending on subtle differences in requirements
- If a table is properly in Third Normal Form, and EFFECTIVELY has no three or more part composite keys, it is already in Fifth Normal Form

Fourth Normal Form

- The key of every table should represent no more than one independent multi-valued relationship
- In other words, the combination of key attributes should represent one thing only

Is Fourth Normal Form relevant?

- A 1992 paper by Margaret S. Wu notes that the teaching of database normalization typically stops short of 4NF, perhaps because of a belief that tables violating 4NF (but meeting all lower normal forms) are rarely encountered in business applications. This belief may not be accurate, however. Wu reports that in a study of forty organizational databases, over 20% contained one or more tables that violated 4NF while meeting all lower normal forms.
- http://en.wikipedia.org/wiki/Fourth_normal_form

Fourth Normal Form Example

- Requirement: define the classes offered with teacher and book

Trainer	Class	Book
Louis	Normalization	DB Design & Implementation
Chuck	Normalization	DB Design & Implementation
Fred	Implementation	DB Design & Implementation
Fred	Golf	Topics for the Non-Technical

- Dependencies
  - Class determines Trainer (Based on qualification)
  - Class determines Book (Based on applicability)
  - Trainer does not determine Book (or vice versa)
- If trainer and book are related (like if teachers had their own specific text,) then this table is in 4NF

Fourth Normal Form Example

Trainer	Class	Book
Louis	Normalization	DB Design & Implementation
Chuck	Normalization	DB Design & Implementation
Fred	Implementation	DB Design & Implementation
Fred	Golf	Topics for the Non-Technical

Question: What classes do we have available and what books do they use?

```
SELECT DISTINCT Class, Book
FROM   TrainerClassBook
```

Class	Book
Normalization	DB Design & Implementation
Implementation	DB Design & Implementation
Golf	Topics for the Non-Technical

Doing a very slow operation, sorting your data unnecessarily… please wait

Fourth Normal Form Example

- Break Trainer and Book into independent relationship tables to Class

Class	Trainer
Normalization	Louis
Normalization	Chuck
Implementation	Fred
Golf	Fred

Class	Book
Normalization	DB Design & Implementation
Implementation	DB Design & Implementation
Golf	Topics for the Non-Technical

Fifth Normal Form

- A general rule that breaks out any data redundancy that has not specifically been called out by additional rules
- Like Fourth Normal Form, deals with the relationship between key attributes
- Basically, if you can break a table with three (or more) independent keys into three individual tables and be guaranteed to get the original table by joining them together, the table is not in Fifth Normal Form

- An esoteric rule that is only occasionally violated (but still interesting!)
- *Examples in hidden slides*

Fifth Normal Form Example

- Requirement: Store types of cars driver willing to rent

Driver	Car Style	Car Brand
Louis	Station Wagon	Ford
Louis	Hatchback	Hyundai
Ted	Coupe	Chevrolet

- Table is in 5NF if this represents:
  - Louis is strictly willing to drive any Ford Station Wagon or Hyundai Hatchback
  - Ted is willing to drive any Coupe from Chevrolet
- Because:
  - Driver determines Car Style
  - Driver determines Car Brand
  - Car Brand determines Car Style
  - Driver determines Car Style and Car Brand

Fifth Normal Form Example

- Requirement: Store types of cars driver willing to rent

Driver	Car Style	Car Brand
Louis	Station Wagon	Ford
Louis	Hatchback	Hyundai
Ted	Coupe	Chevrolet

- Table is not in 5NF if this represents:
  - Louis is willing to drive any Station Wagon or Hatchback from Ford or Hyundai
  - Ted is willing to drive any Coupe from Chevrolet
- Still 4th Because:
  - Driver determines Car Style
  - Driver determines Car Brand
  - Car Brand determines Car Style

Fifth Normal Form Example

- Solution: Three independent tables

Driver	Car Style
Louis	Station Wagon
Louis	Hatchback
Ted	Coupe

Driver	Car Brand
Louis	Ford
Louis	Hyundai
Ted	Chevrolet

Car Style	Car Brand
Station Wagon	Ford
Hatchback	Hyundai
Coupe	Chevrolet

Fifth Normal Form Example

Driver	Car Style	Car Brand
Louis	Station Wagon	Ford
Louis	Hatchback	Hyundai
Ted	Coupe	Chevrolet

- Alternative: Table is in 5NF if this represents:
  - Louis is willing to drive Ford Station Wagons and Hyundai Hatchbacks
  - Ted is willing to drive a Chevrolet Coupe
  - Ford only makes (or we only stock) a Station Wagon, Hundai only makes a hatchback and Chevrolet only makes a coupe
- Because: Driver determines Car Style + Car Brand
- In a well designed system, with these requirements…
  - The intersection of Style and Brand would have formed it's own table
  - Car Style/Car Brand would have been recognized as an independent object with a specific key (often a surrogate).

Book, editor, author

- Requirement: Represent books with their authors and editors

Book	Author	Editor
Design	Louis	Tony
Design	Jeff	Leroy
Golf	Louis	Steve
Golf	Fred	Tony

- Table is in not in 4NF if this represents:
  - Book “Design” has authors Louis and Jeff and Editor Tony
  - Book “Golf” has authors Louis and Fred and Editors Steve and Tony
- Table is in 5NF if this represents
  - For book “Design”, Editor Tony edits Louis’ work and Leroy edits Jeff’s work
  - For book “Golf”, Editor Steve edits Louis’ work, and Tony edits Fred’s work.

Normal Review

- The normal forms should govern the design for the models you create
- First Normal Form is for the engine
  - Even data warehouses are largely in First Normal Form!
- BCNF, Fourth, and Fifth are for data consistency
- In the end you get a programming surface that is resilient to change and works like SQL Server expects
- This isn't hard stuff!

Can you over-normalize?

- Short answer: sort of
- Long answer: no
  - Breaking objects down beyond user needs is not productive
  - Lots of joins are not always that costly
  - Over-normalization is usually over-engineering, ignoring what the user needs

Normalization Final Scan
(The Normal way to Normalize)

- Columns - One column, one value
- Table/row uniqueness – Tables have independent meaning, rows are distinct from one another.
- Proper relationships between columns – Columns either are a key or describe something about the row identified by the key.
- Data dependent directly on data in the same table, or a foreign key
- Scrutinize intra-key dependencies - Make sure relationships between three values in a key. Reduce all relationships to binary relationships if possible.

Normalization is done when..

- Data can be used programmatically without parsing
- Users have exactly the right number of places to store the data they need
- Users stop changing their needs

- Pretty much when a system is completely retired

Denormalization

- 100% an implementation concept
- Stepping back from strict normalization for a good reason
- Common reasons
  - Performance
  - Usability
  - But mostly
    - FUD (Fear, uncertainty, doubt)
    - Try the normalized version first if possible
    - Learn how to tune your data platform
- Not knowing what normalization is not denormalization

Denormalization

- Adjusting a design that has been normalized in a manner that has caused some level of problem
- Usually this is sold as having to do with performance or usability
- Common saying
  - ~~Normalize 'til it hurts, Denormalize 'til it works~~
  - Normalize 'til it works.
    - In reality, there is very little reason to denormalize when Normalization has been done based on requirements and user need.

– There are common exceptions…

Typically acceptable denormalization

- When read/write ratio approaches infinity
- Examples
  - Balances/Inventory as of a certain date (summing activity after that date for totals)
    - Allows you to query a LOT LESS data
  - Calendar table
    - November 15, 2006 with always be a Wednesday
  - Table of integers
    - Prime Numbers
    - Odd Numbers

Final Exam: Data Model For a House

- A company needs a database to model houses on a block

- Unless I tell you what the company does, you will not get the answer right… Perspective is everything!

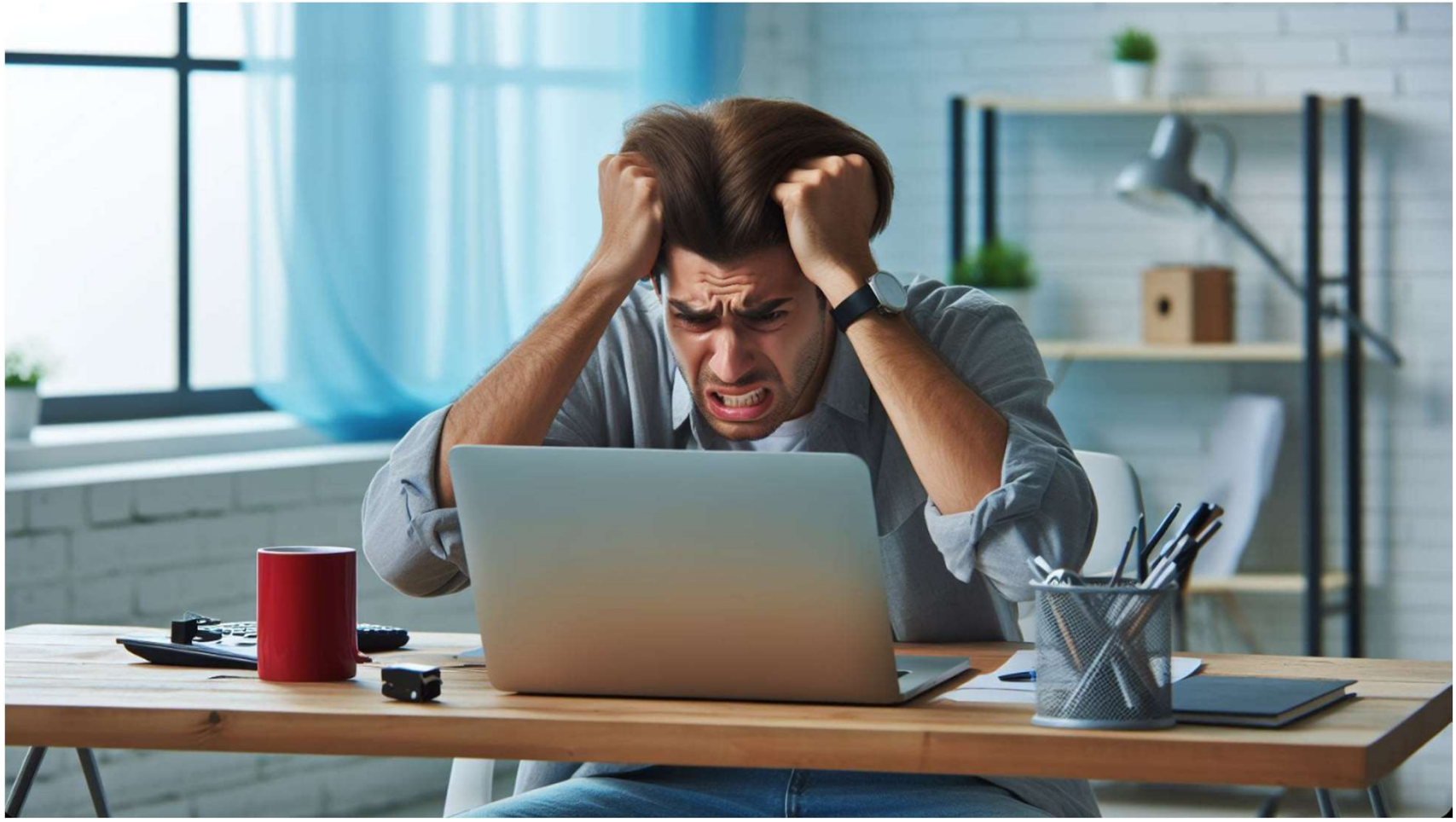
“Daydream” Practice

- A good way to get better is to pick out scenarios in real life and mentally model them
- Such as:
  - Grocery list management
  - DMV
  - Theme park operations
- Build models in your spare time to reinforce your daydreams and your modeling skills
- Warning: telling non-nerdy people you do this will open you up to teasing.

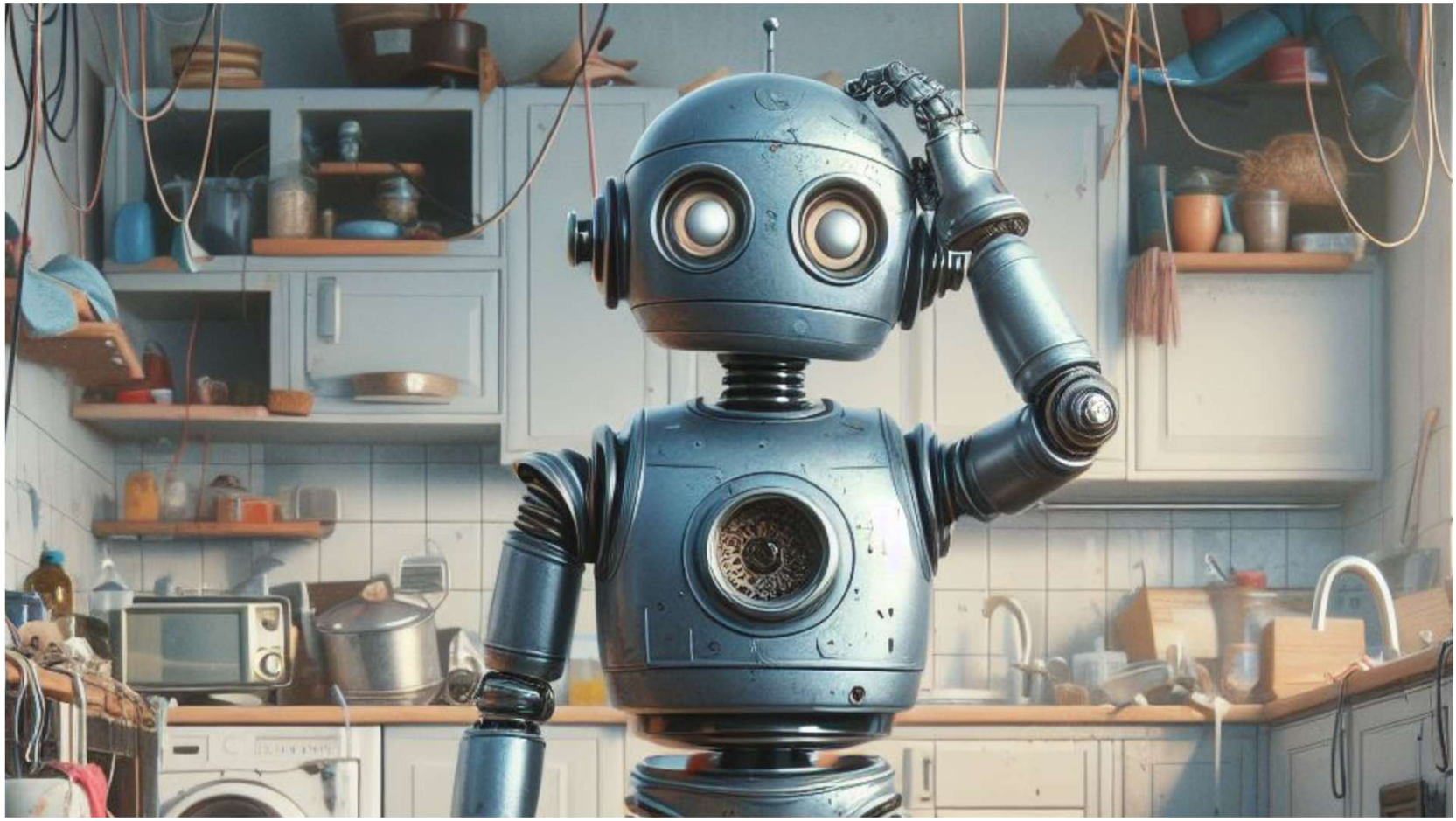
So you have a design…build!

- I usually build to a SQL Server that contains model databases
- Strongly consider using SQL Server (or any RDBMS) to implement
  - Data checks (check constraints)
  - Column domains (data types and check/null constraints)
  - Table relationships (foreign key constraints)
- Minimally, make certain that the data is protected at a level lower than a user can get access
- Ideally:
  - You should trust that your data is valid to the immutable business rules at all times
  - The only data cleansing you should ever do is to look for users doing dumb stuff

The better you design…the less your users make this face



Perhaps more importantly… nor does AI



Test…test…test

- Start building unit tests in during conceptual modeling that you can run to make sure that your design works
- Remember the requirements? Develop the test plan directly from them
- Throw tons of bad data at the design, especially places where you thought data should have been designed better
- Try to get an error for every constraint implemented
- Try to get an error for every constraint NOT implemented

Questions? Contact info..

- Louis Davidson - louis@drsql.org
- Website – http://drsql.org
  - Get slides here: https://drsql.org/presentations
- Twitter – http://twitter.com/drsql

- Editor/Blogger at 

https://www.red-gate.com/simple-talk/author/louis-davidson/
- (Always looking for new writers! editor@simple-talk.com)

Thank you!