Excuse me, but is your data dead?

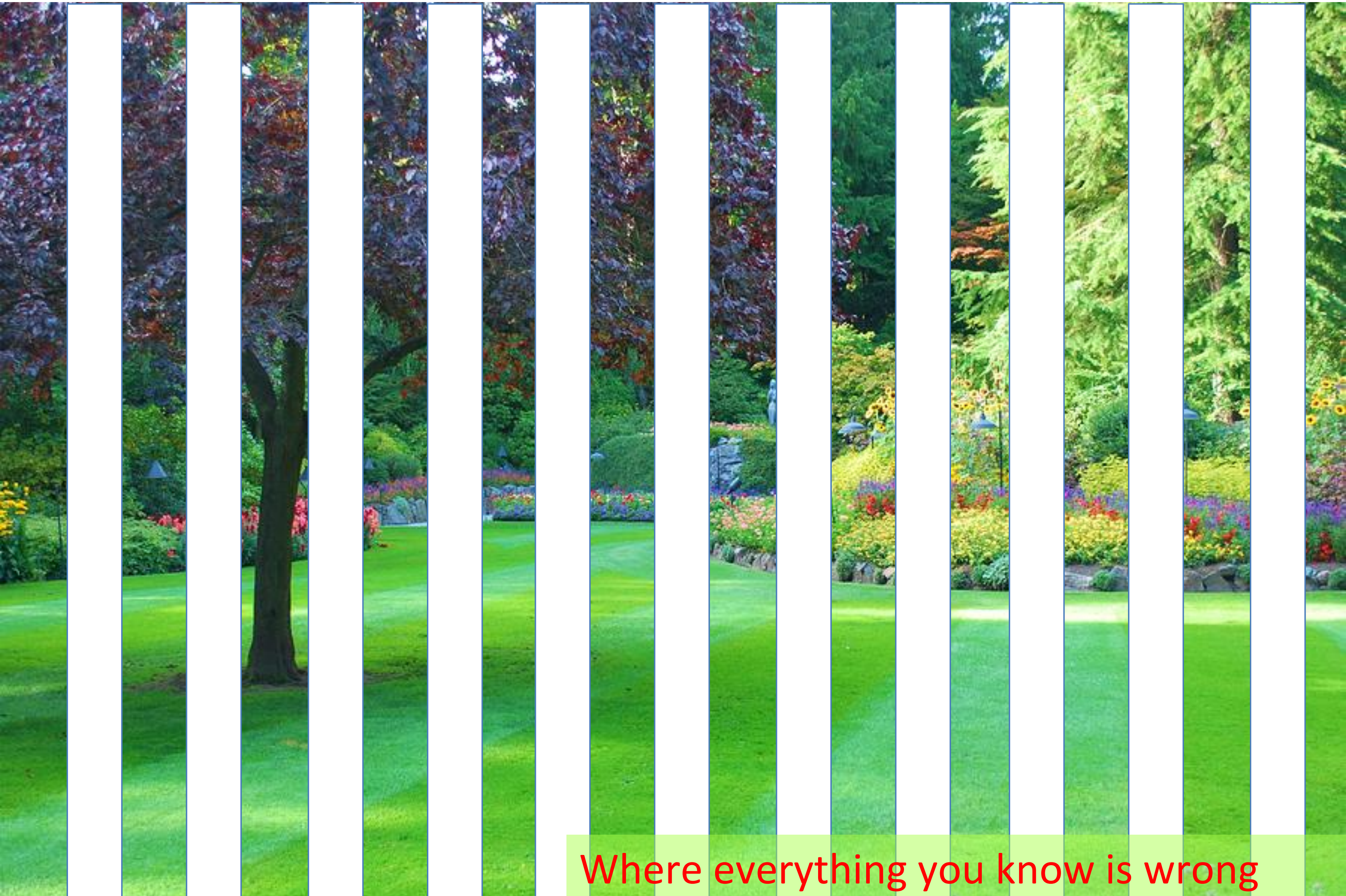
Toby Lehman

Aerospike

The Relational Landscape

Live Data
Rich, fertile Data Structures
Powerful Operators
Massive Data Operations

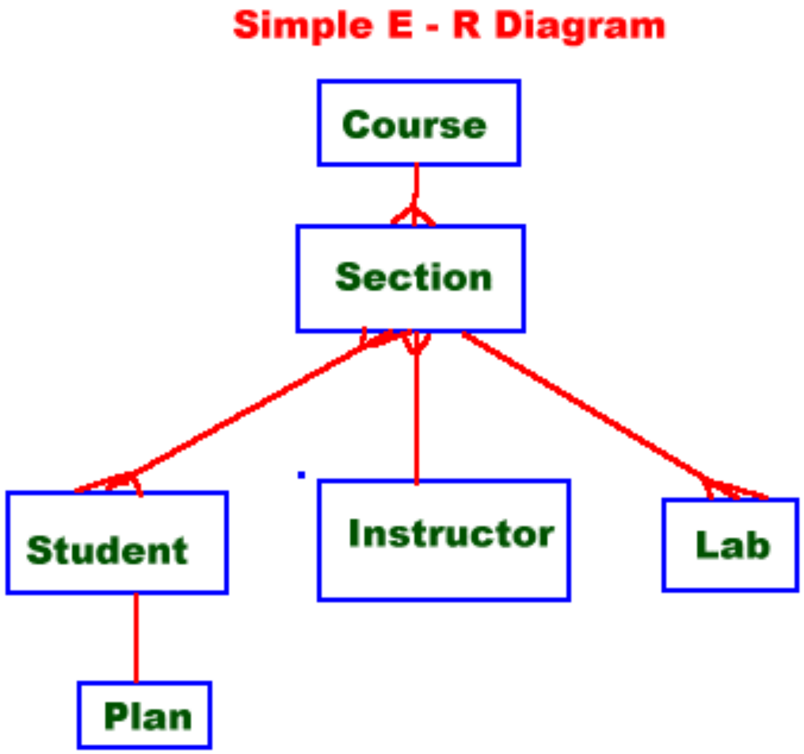
Stonebraker Land



Where everything you know is wrong

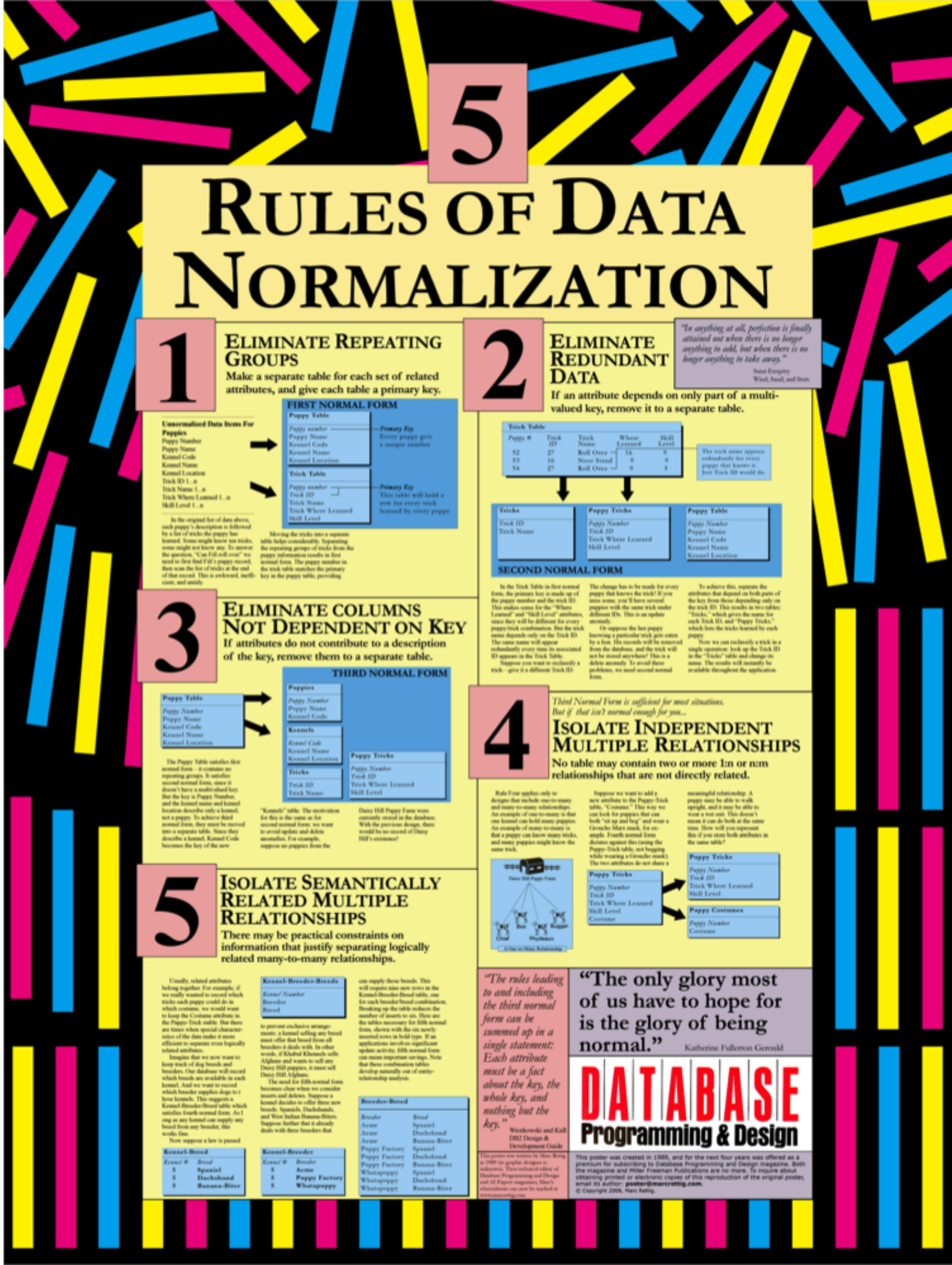
The Joy of Data Modeling

- Section
- Course #
- Course Name
- # Seats
- # Enrolled
- Room
- Time
- Days
- Instructor
- Students



Embrace Normalization

- The Famous Puppy Poster shows the complexities of normalization
- Suffer the penalties of joining the data back again.
- Why Normalize? Because LIVE data may have Update or Delete problems if copied.



5 Rules of Data Normalization

1 Eliminate Repeating Groups

Make a separate table for each set of related attributes, and give each table a primary key.

Unnormalized Data Items For Puppies
Puppy Number
Puppy Name
Kennel Code
Kennel Name
Kennel Location
Trick ID 1...n
Trick Name 1...n
Trick Where Learned 1...n
Skill Level 1...n

FIRST NORMAL FORM

Puppy Table
Puppy number — *Primary Key* Every puppy gets a unique number
Puppy Name
Kennel Code
Kennel Name
Kennel Location

Trick Table
Puppy number
Trick ID — *Primary Key* This table will hold a row for every trick learned by every puppy
Trick Name
Trick Where Learned
Skill Level

In the original list of data above, each puppy's description is followed by a list of tricks the puppy has learned. Some might know ten tricks, some might not know any. To answer the question, "Can Fifi roll over?" we need to first find Fifi's puppy record, then scan the list of tricks at the end of that record. This is awkward, inefficient, and untidy.

Moving the tricks into a separate table helps considerably. Separating the repeating groups of tricks from the puppy information results in first normal form. The puppy number in the trick table matches the primary key in the puppy table, providing

2 Eliminate Redundant Data

"In anything at all, perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away."
Saint-Exupéry
Wind, Sand, and Stars

If an attribute depends on only part of a multi-valued key, remove it to a separate table.

Trick Table

Puppy #	Trick ID	Trick Name	Where Learned	Skill Level
52	27	Roll Over	16	9
53	16	Nose Stand	9	9
54	27	Roll Over	9	5

The trick name appears redundantly for every puppy that knows it. Just Trick ID would do.

SECOND NORMAL FORM

Tricks
Trick ID
Trick Name

Puppy Tricks
Puppy Number
Trick ID
Trick Where Learned
Skill Level

Puppy Table
Puppy Number
Puppy Name
Kennel Code
Kennel Name
Kennel Location

In the Trick Table in first normal form, the primary key is made up of the puppy number and the trick ID. This makes sense for the "Where Learned" and "Skill Level" attributes, since they will be different for every puppy/trick combination. But the trick name depends only on the Trick ID. The same name will appear redundantly every time its associated ID appears in the Trick Table.

Suppose you want to reclassify a trick – give it a different Trick ID. The change has to be made for every puppy that knows the trick! If you miss some, you'll have several puppies with the same trick under different IDs. This is an update anomaly.

Or suppose the last puppy knowing a particular trick gets eaten by a lion. His records will be removed from the database, and the trick will not be stored anywhere! This is a delete anomaly. To avoid these problems, we need second normal form.

To achieve this, separate the attributes that depend on both parts of the key from those depending only on the trick ID. This results in two tables: "Tricks," which gives the name for each Trick ID, and "Puppy Tricks," which lists the tricks learned by each puppy.

Now we can reclassify a trick in a single operation: look up the Trick ID in the "Tricks" table and change its name. The results will instantly be available throughout the application.

3 Eliminate Columns Not Dependent on Key

If attributes do not contribute to a description of the key, remove them to a separate table.

Puppy Table
Puppy Number
Puppy Name
Kennel Code
Kennel Name
Kennel Location

THIRD NORMAL FORM

Puppies
Puppy Number
Puppy Name
Kennel Code

Kennels
Kennel Code
Kennel Name
Kennel Location

Tricks
Trick ID
Trick Name

Puppy Tricks
Puppy Number
Trick ID
Trick Where Learned
Skill Level

The Puppy Table satisfies first normal form – it contains no repeating groups. It satisfies second normal form, since it doesn't have a multivalued key. But the key is Puppy Number, and the kennel name and kennel location describe only a kennel, not a puppy. To achieve third normal form, they must be moved into a separate table. Since they describe a kennel, Kennel Code becomes the key of the new "Kennels" table. The motivation for this is the same as for second normal form: we want to avoid update and delete anomalies. For example, suppose no puppies from the Daisy Hill Puppy Farm were currently stored in the database. With the previous design, there would be no record of Daisy Hill's existence!

4 Isolate Independent Multiple Relationships

Third Normal Form is sufficient for most situations.
But if that isn't normal enough for you...

No table may contain two or more 1:n or n:m relationships that are not directly related.

Rule Four applies only to designs that include one-to-many and many-to-many relationships. An example of one-to-many is that one kennel can hold many puppies. An example of many-to-many is that a puppy can know many tricks, and many puppies might know the same trick.

Suppose we want to add a new attribute to the Puppy-Trick table, "Costume." This way we can look for puppies that can both "sit up and beg" and wear a Groucho Marx mask, for example. Fourth normal form dictates against this (using the Puppy-Trick table, not begging while wearing a Groucho mask). The two attributes do not share a meaningful relationship. A puppy may be able to walk upright, and it may be able to wear a wet suit. This doesn't mean it can do both at the same time. How will you represent this if you store both attributes in the same table?

Daisy Hill Puppy Farm
Chief
Boo
Phydeaux
Bugger
A One-to-Many Relationship

Puppy Tricks
Puppy Number
Trick ID
Trick Where Learned
Skill Level
Costume

Puppy Tricks
Puppy Number
Trick ID
Trick Where Learned
Skill Level

Puppy Costumes
Puppy Number
Costume

5 Isolate Semantically Related Multiple Relationships

There may be practical constraints on information that justify separating logically related many-to-many relationships.

Usually, related attributes belong together. For example, if we really wanted to record which tricks each puppy could do in which costume, we would want to keep the Costume attribute in the Puppy-Trick table. But there are times when special characteristics of the data make it more efficient to separate even logically related attributes.

Imagine that we now want to keep track of dog breeds and breeders. Our database will record which breeds are available in each kennel. And we want to record which breeder supplies dogs to those kennels. This suggests a Kennel-Breeder-Breed table which satisfies fourth normal form. As long as any kennel can supply any breed from any breeder, this works fine.

Now suppose a law is passed to prevent exclusive arrangements: a kennel selling any breed must offer that breed from all breeders it deals with. In other words, if Khabul Khennels sells Afghans and wants to sell any Daisy Hill puppies, it must sell Daisy Hill Afghans.

The need for fifth normal form becomes clear when we consider inserts and deletes. Suppose a kennel decides to offer three new breeds: Spaniels, Dachshunds, and West Indian Banana-Biters. Suppose further that it already deals with three breeders that can supply those breeds. This will require nine new rows in the Kennel-Breeder-Breed table, one for each breeder/breed combination. Breaking up the table reduces the number of inserts to six. Here are the tables necessary for fifth normal form, shown with the six newly inserted rows in bold type. If an application involves significant update activity, fifth normal form can mean important savings. Note that these combination tables develop naturally out of entity-relationship analysis.

Kennel-Breeder-Breeds
Kennel Number
Breeder
Breed

Kennel-Breed

Kennel #	*Breed*
5	**Spaniel**
5	**Dachshund**
5	**Banana-Biter**

Kennel-Breeder

Kennel #	*Breeder*
5	**Acme**
5	**Puppy Factory**
5	**Whatapuppy**

Breeder-Breed

Breeder	*Breed*
Acme	Spaniel
Acme	Dachshund
Acme	Banana-Biter
Puppy Factory	Spaniel
Puppy Factory	Dachshund
Puppy Factory	Banana-Biter
Whatapuppy	Spaniel
Whatapuppy	Dachshund
Whatapuppy	Banana-Biter

"The rules leading to and including the third normal form can be summed up in a single statement: Each attribute must be a fact about the key, the whole key, and nothing but the key."
Wiorkowski and Kull
DB2 Design & Development Guide

"The only glory most of us have to hope for is the glory of being normal." Katherine Fullerton Gerould

DATABASE
Programming & Design

This poster was written by Marc Rettig in 1989 (its graphic designer is unknown). Then technical editor of Database Programming and Design and AI Expert magazines, Marc's whereabouts can now be tracked at www.marcrettig.com.

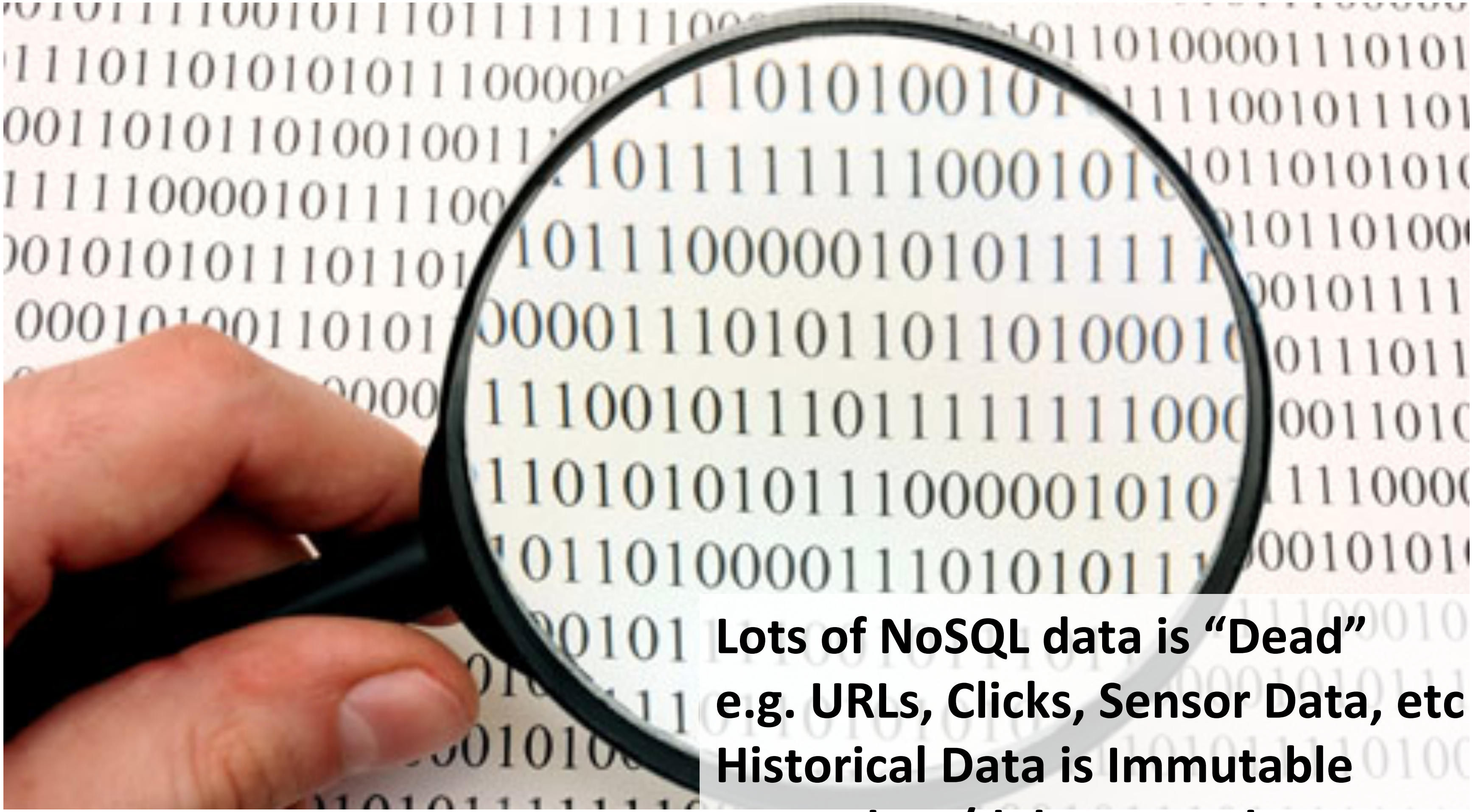
This poster was created in 1989, and for the next four years was offered as a premium for subscribing to Database Programming and Design magazine. Both the magazine and Miller Freeman Publications are no more. To inquire about obtaining printed or electronic copies of this reproduction of the original poster, email its author: **poster@marcrettig.com**.
© Copyright 2006, Marc Rettig.

The NoSQL Landscape

Bare Bones
High Performance
Key-Value Store
Simple Data
Simple Operations
No Joins (mostly)

But … What about 1:N, N:M Relationships?

Let’s Look More Closely at the Data



Lots of NoSQL data is “Dead”
e.g. URLs, Clicks, Sensor Data, etc
Historical Data is Immutable
No update/delete penalty
➔ Promotes Collection Types

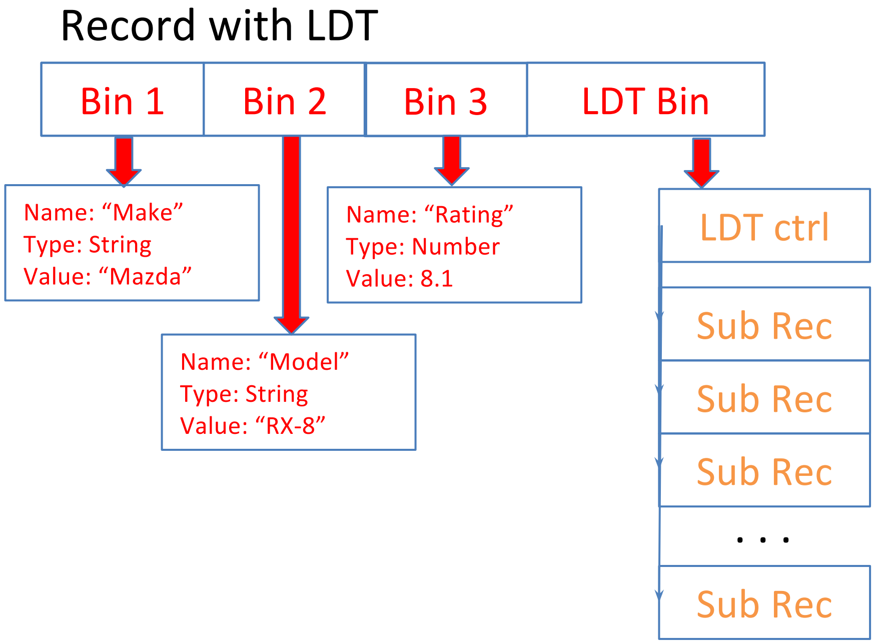
The NoSQL Landscape

Object Collections Can Satisfy Relationships

Complex Field Values
Large Stacks (time series)
Large Sets (unique objects)
Large Ordered Lists (B+ Tree)
Large Maps (name/value)

Aerospike Large Data Types

- Aerospike Record



Record with LDT

Bin 1

Bin 2

Bin 3

LDT Bin

Name: “Make”
Type: String
Value: “Mazda”

Name: “Model”
Type: String
Value: “RX-8”

Name: “Rating”
Type: Number
Value: 8.1

LDT ctrl

Sub Rec

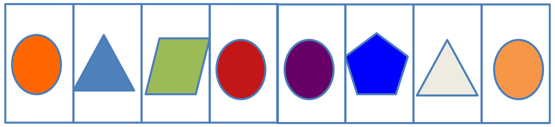
Sub Rec

Sub Rec

. . .

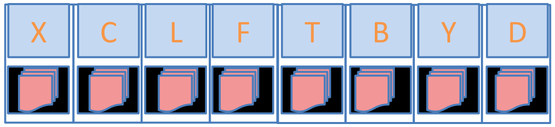
Sub Rec

- Set



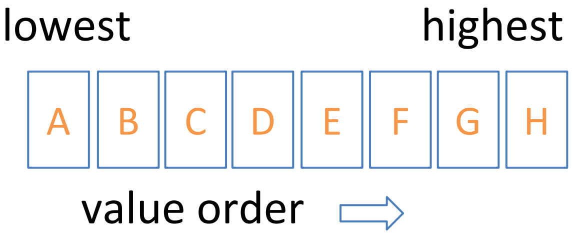
unique collection

- Map



Name/Value Pairs

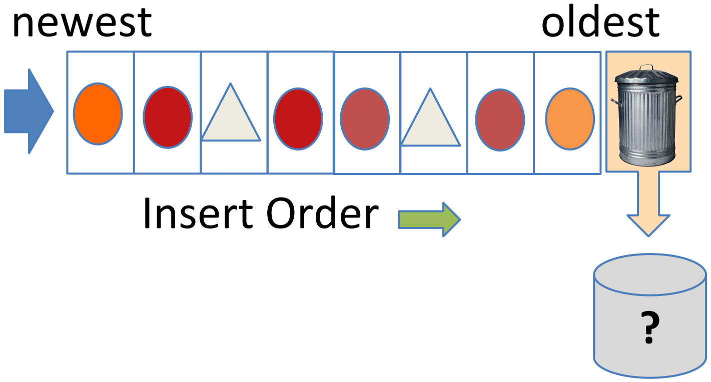
- List
- Stack



lowest

highest

value order



newest

oldest

Insert Order

AEROSPIKE

Summary

- In NoSQL Land, much of the data captured is historical (dead data)
- No penalty, other than increased storage, to keep copies
- Often a huge performance win to read a collection as part of a record (much like a pre-computed materialized join)
- Different collection types (stack, list, map, set) match the different application access patterns

- **Questions: toby@aerospike.com**