Agile Database Development?

Yes we can!

Ron Ballard

What do you think of when you hear the words:

Relational database

?

What do you think of when you hear the words:

Relational database

SQL

?

What do you think of when you hear the words:

Relational database

SQL

Agile Development

?

What I Learned

Building database systems:

- Small teams: exploring methodologies and tools

Developing database applications:

- Weight of methodologies
- Peopleware (Tom DeMarco & Tim Lister)
- Extreme Programming (Kent Beck - Paul Beckford)
- Ruby on Rails
- Several successful projects
- Agile database development

How Heavy Are The Methodologies?

A major finance company spent **£400 million** between 2012 and 2016 developing conceptual and logical database models that were incomplete, riddled with errors and never used!

How Heavy Are The Methodologies?

A major finance company spent **£400 million** between 2012 and 2016 developing conceptual and logical database models that were incomplete, riddled with errors and never used!

They had some big diagrams on the wall though.

Database vendors and major consultancies
have used overweight methodologies
to make fortunes
out of business and government
while failing to deliver good systems
over and over again.

Database vendors and major consultancies
have used overweight methodologies
to make fortunes
out of business and government
while failing to deliver good systems
over and over again.

Is This Contentious?

The Damage Done

- Methodologies got heavier
  - "Things are going wrong; we need more control!"
- The tools from this era were tarred with the same brush - some validly; some not
- Existing vendors made tools more complex with more layers of "easy-to-use" administration tools
- Larry Ellison bought fighter jets and yachts with the proceeds
- New vendors piled in with allegedly simpler tools.
- Scott Ambler's approach - "more UML" - is just wrong.

Time to Stop Moaning

Agile

The Agile Manifesto

- Working code is delivery; everything else is overhead
- Users and developers must work together
- Change is what we do; do not fear change
- People deliver; processes and tools do not

Team Structure

- All necessary skills inside the team
- Includes users as part of the team
- No external DBAs (Database Administrators) – database design by team members
- Analysis and development by the same people
- Testing by the same people
- Documentation by the same people

Database Design

- The data model is a direct model of the real objects being processed by the application
- Understand the data and the model becomes obvious; no designers' ivory tower needed
- No conceptual model; no logical model; the model in the database is the only one
- Always design concrete objects; no abstract or generic objects
- Follow the checklist

Database Design Checklist

- Every column is atomic
- Every row has a primary key
- Every column in a row depends on the primary key and nothing else
- No repeating groups in a row

Database Design Checklist

- Every column is atomic
- Every row has a primary key
- Every column in a row depends on the primary key and nothing else
- No repeating groups in a row
- Strong data-types
- Concrete, specific, plain-English names
- Restrict nulls wherever possible
- Define constraints to protect links

Sneaky?

- If you are aware of "Database Normalisation" you will spot that I snuck it into the checklist.
- "Normalisation" is useful because it makes your database easier to use.
- Read about helicopter controls in The Pragmatic Programmer.
- "Normalisation" has too much obscure jargon
- I'll give you one example of normalisation to explain why it is useful.

Every Column Atomic

<vehicle>Mini John Cooper Works RH62XRG</vehicle>

Every Column Atomic

<vehicle>Mini John Cooper Works RH62XRG</vehicle>

<vehicle>Land Rover Discovery HIP 5TR</vehicle>

Every Column Atomic

<vehicle>Mini John Cooper Works RH62XRG</vehicle>

```
<vehicle>
  <make>Mini</make>
  <model>John Cooper Works</model>
  <registration>RH62XRG</registration>
</vehicle>
```

<vehicle>Land Rover Discovery HIP 5TR</vehicle>

```
<vehicle>
  <make>Land Rover</make>
  <model>Discovery</model>
  <registration>HIP 5TR</registration>
</vehicle>
```

Normal Forms

Normal Form	**In English (sort-of)**	**When To Use It**
First	Atomic fields	Always
Second	Every field part of same object	Always
Third	No internal relationships	Usually
Bryce-Codd	Overlapping candidate keys	You Ain't Gonna Need It!
Fourth	Multi-valued dependencies	
Fifth	Join dependencies	

So What is ***Agile*** Database Design?

- Design ***only*** what is needed for this iteration (sprint).
- The first migration will just be some “create table” statements.
- On subsequent iterations, if the database does not change do nothing. This often happens.
- If the database does change, build a migration to add what is new, and, if necessary, migrate data as well as definitions.
- If you are migrating data then build tests to ensure that nothing has unintentionally been lost, duplicated or shuffled.
- Refactor the design whenever it delivers a benefit.
- Store all migrations in your version control system. Organise and name them so that you can replay them.

Agile Database Development

If "You Ain't Gonna Need It", don't do it.

For example:

- No indexes unless they solve an actual performance problem, and then only if they actually solve it.
- No partitioning unless absolutely necessary
- No "spare" columns at all, ever!
- No database "views"

Agile Database Development
Separation of Responsibilities

Use the database only for what it is good for:

- No dodgy data-types: "blobs", XML, arrays, etc.
- No nested tables
- No "stored-procedures" (limited exceptions)
- SQL is the only interface to the database
- SQL is non-procedural; don't subvert it
- Do use strong data-types to keep data clean
- No string longer than it needs to be

Agile Database Development
Separation of Responsibilities

Use the application code for what it is good for

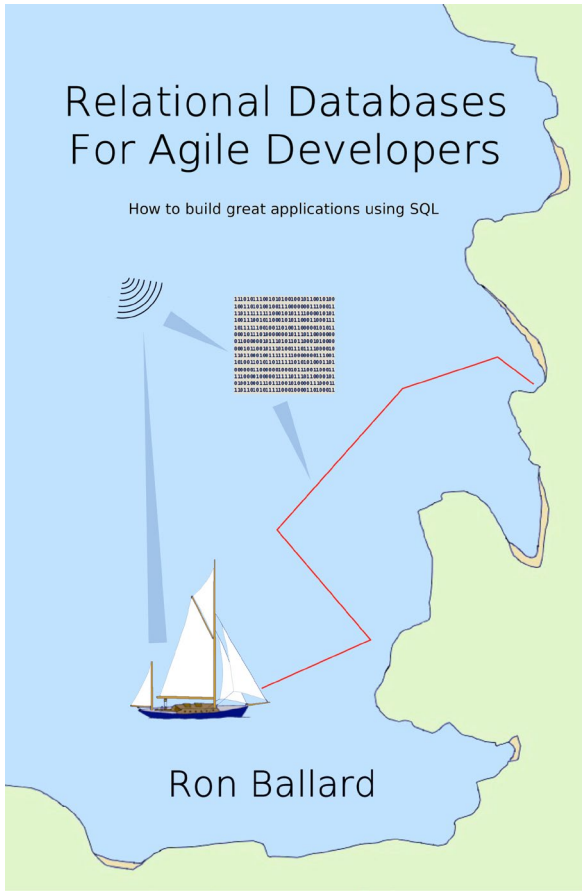
- No database navigation in the application
- Never use Object-Relational Mapping systems
- SQL is the interface to the database
- JDBC is a good implementation of the SQL interface (not perfect but the best available)

Agile Database Development
Test Driven

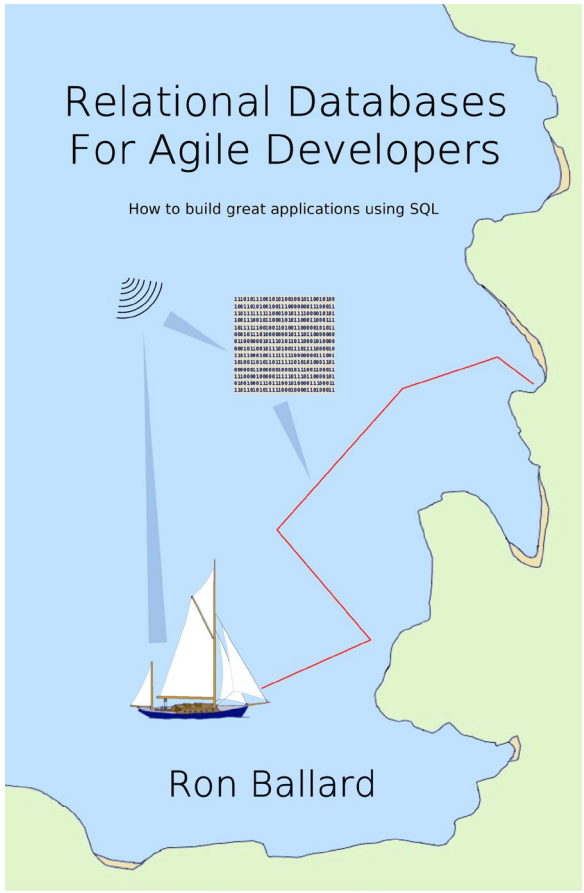
- Automated tests must include database tests
- Test database migrations and application releases
- SQL scripts are a good tool to check that the database changed as expected
- Automate running of scripts and reporting of results. Must be able to make the screen turn red if the release is broken.

The End

- Relational Databases and SQL are powerful when used for appropriate applications
- Relational Database development can be Agile and rigorous
- Relational Databases can be big and fast


Relational Databases
For Agile Developers
How to build great applications using SQL
Ron Ballard

Questions?


Relational Databases
For Agile Developers
How to build great applications using SQL
Ron Ballard