How Do We Solve the World’s Spreadsheet Problem?

Alex Rasmussen

@alexras
Hi, I’m Alex!

@alexras
alexras.info

freenome
freenome.com

BITS ON DISK
bitsondisk.com
My Background

2009-2013: really fast sorting
2013-2016: data wrangling
2017-2018: cancer-fighting robots
I think/worry a lot about spreadsheets.
Today’s focus: spreadsheet data
(for compute, felienne.com)
This talk:

1. Spreadsheets are great
2. Spreadsheets are a problem
3. How we can fix it
This talk:

1. Spreadsheets are great
2. Spreadsheets are a problem
3. How we can fix it
What’s so great about spreadsheets?
Spreadsheets are Ubiquitous
1.2 billion Office users (~16% of humans)
1.2 billion Office users (~16% of humans)

60 million Office 365 customers
1.2 billion Office users (~16% of humans)

60 million Office 365 customers

>5 million businesses use Google Apps
Spreadsheets are Approachable
<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th></th>
<th>Multiply</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>17.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33.6</td>
<td>117.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>640</td>
<td>2240</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is a column!
Spreadsheets are Flexible
Data grids
Data grids
Graphs
Data grids
Graphs
Anything tabular
Data grids
Graphs
Anything tabular
Full-scale “apps”
Tatsuo Horiuchi (b. 1940)
*Kegon Falls*, 2007
AutoShape on canvas
So if spreadsheets are ubiquitous, approachable, and flexible, what’s the problem?
1. Spreadsheets are great
2. Spreadsheets are a problem
3. How we can fix it
Problem #1: Data Types
ME: *makes typo while entering a number*

EXCEL: WAS THAT A DATE

ME: no I meant t-

EXCEL: THAT WAS A 😡 DATE

ME: it doesn't even make sen-

EXCEL: MAY 12TH 1382. LOOK I EVEN FORMATTED IT. IT IS THIS FOREVER

9:00 AM - Feb 1, 2018
Automatic type conversion can cause serious problems.
RIKEN Identifier
2310009E13
RIKEN Identifier
2310009E13
2.31E+13
“We confirmed gene name errors in 987 supplementary files from 704 published articles (19.6% of all articles).”

False Equivalence

000123
= 00123
= 123

True if they’re integers, but what if they’re strings?
Enumerated Types
Enumerated Types

“Prostate Cancer”
Enumerated Types

“Prostate Cancer”
“prostate cancer”
Enumerated Types

“Prostate Cancer”
“prostate cancer”
“prostatecancer”
Enumerated Types

“Prostate Cancer”
“prostate cancer”
“prostatecancer”
“PC”
Enumerated Types

“Prostate Cancer”
“prostate cancer”
“prostatecancer”
“PC”
“prostate”
Enumerated Types

“Prostate Cancer”
“prostate cancer”
“prostatecancer”
“PC”
“prostate”
“prostrate”
List Validations? Sheet Protection?

😃 Easy to add
😊 Easy to remove by accident
😡 Hard to enforce
Data loss!
False equivalence!
Ontological chaos!
Mass hysteria!
Problem #2: Queryability
Inside a spreadsheet, things are pretty good!
Formulas!

Pivot Tables!

Filters!
What about querying across spreadsheets?
Get and Transform

![Excel screenshot showing the Data ribbon with Get & Transform highlighted.](image)
No Mac support.
Structure changes?
Type changes?
Column Renames?
Have fun re-loading.
And what about joins?
There’s VLOOKUP

=VLOOKUP(“Product 1”, Prices!$A$2:$B$9,2, FALSE)

... but, like, eww.
Data **inside** a spreadsheet is hard to connect to data **outside** that spreadsheet.
Summary:
Spreadsheets are bad at types and hard to query
This talk:

1. Spreadsheets are great
2. Spreadsheets are a problem
3. How we can fix it
What about databases?
Databases are great in ways that spreadsheets aren’t.
Databases are great at data type **definition** and **enforcement**.
## So Many Types of Types!

<table>
<thead>
<tr>
<th>Numeric</th>
<th>Enumerated</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monetary</td>
<td>Geometric</td>
<td>JSON</td>
</tr>
<tr>
<td>Character</td>
<td>Network Address</td>
<td>Arrays</td>
</tr>
<tr>
<td>Binary</td>
<td>Bit String</td>
<td>Composite</td>
</tr>
<tr>
<td>Date/Time</td>
<td>Text Search</td>
<td>Range</td>
</tr>
<tr>
<td>Boolean</td>
<td>UUID</td>
<td>Pseudo-Types</td>
</tr>
</tbody>
</table>
Databases are purpose-built for queries and joins.
BUT
Databases aren’t as approachable as spreadsheets.
$ psql -d postgres

psql (10.4, server 9.6.9)
Type "help" for help.

postgres=#
Databases aren’t as flexible as spreadsheets.
Databases are good at storing and querying data. But that’s it.
Spreadsheets and databases have complementary skillsets.
So, what do we do about it?
How to Solve Your Spreadsheet Problem
1. Identify the use case.
2. Stop the spread.
3. Backfill.
1. Identify the use case.

2. Stop the spread.

3. Backfill.
Every spreadsheet solves a problem. What is that problem?
What’s the business need?  
How much data is there?  
How fast does it change?  
How frequent are additions?
1. Identify the use case.
2. Stop the spread.
3. Backfill.
Give new data a structured home.
No custom apps.

At least at first.
Optimize for Speed
1. Identify the use case.
2. Stop the spread.
3. Backfill.
It’s time for some Data Wrangling. (yee-haw 😐)
Writing one-off scripts is sometimes the best option.
https://www.trifacta.com/start-wrangling/
Infer wrangle "recipe" from high-level actions.

https://www.trifacta.com/start-wrangling/
Another Option: Programming By Example
as hierarchical data, become complex geometric structures. While tools are not expressive enough to capture relational information, importing code, regular expressions and other string-matching techniques are another major source of semi-structured data. Without supporting ad-hoc encodings, data into relational tables, the design of a domain-specific language (DSL) called FlashRelate has enabled developers with scripting languages with regular expression capabilities, to extract relational data from a small number of user-provided examples. The strategy we use to achieve this goal has two parts.

1) We present a domain-specific language (DSL) called FlashRelate for extracting relevant data from ad hoc structures into relational ones. The strategy we use to achieve this goal has two parts. 1) The spreadsheet author’s encoding, while perhaps easy to read, makes this task a challenge. The first thought might be to convert the data into a relational table shown in Fig. 1(b). One approach would be to use a scripting language like Perl or Visual Basic to de-

2) For example, one could match country names with an alphanumeric pattern, years with 4-digit patterns, etc. Having done this, matches would be spatially collated with an edge that automatically generates a directed constraint graph over the spreadsheet. Vertices consist of cell constraints while edges consist of spatial constraints. Data that matches these constraints are automatically converted into relational tables.

The cells matched by a traversal of this graph produce a set of cell constraints while edges consist of spatial constraints. Data that matches these constraints are automatically converted into relational tables. Recall that the user would like to access to rich data or have to resort to manual copy-and-paste. The expertise required to use these tools is often particular to specific document types. Second, and more significantly, solutions to data extraction suffer two key limitations. First, they require knowledge of programming. The first aspect puts these solutions out of reach of the vast majority of users, who do not have programming skills, to convert ad hoc structures into relational ones. The second aspect is that, when translated over a spreadsheet, geometric structure that, when translated over a spreadsheet, is inspired declaratively by scripting languages like FlashRelate is a complementary technology that can concisely extract the necessary relational table. Recall that the user would like to have access to rich data or have to resort to manual copy-and-paste. We also present an algorithm that automatically generates a directed constraint graph over the spreadsheet. Vertices consist of cell constraints while edges consist of spatial constraints. Data that matches these constraints are automatically converted into relational tables. Recall that the user would like to have access to rich data or have to resort to manual copy-and-paste. We also present an algorithm that automatically generates a directed constraint graph over the spreadsheet. Vertices consist of cell constraints while edges consist of spatial constraints. Data that matches these constraints are automatically converted into relational tables.

An extracted relational table with the same two tuples highlighted as in Fig. 1a is shown in Fig. 1(b). One approach would be to use a scripting language like Perl or Visual Basic to de-

Synthesis Algorithm

Fig. 3. Intuitively, one can think of this graph as an invariant layout transformation. The cells matched by a traversal of this graph produce a set of cell constraints while edges consist of spatial constraints. Data that matches these constraints are automatically converted into relational tables.

Provide example rows, synthesize layout transformations.

https://github.com/microsoft/prose
Foofah

Provide input/output sample, synthesize layout and syntactic transformations.

https://github.com/umich-dbgroup/foofah
How to **Solve** Your Spreadsheet Problem

1. Identify the use case.
2. Stop the spread.
3. Backfill.
What about the future?
Spreadsheets aren’t going anywhere, for good reason.
Learn from the spreadsheet.
Meet the users where they are.
Thank you.

@alexras

Consulting Inquiries: contact@bitsondisk.com