Thinking about databases

Travis C. Porco

Francis I. Proctor Foundation

June 8, 2016

Databases

- Relational databases, and the common operations on them, form a useful abstraction of common patterns.
- Relational databases aren't the only type of database, but they are common, clear, and give us a vocabulary for talking about how to organize our data.
- Experience suggests most of your time analyzing data is spent organizing, cleaning, and manipulating data.
- A little knowledge is a good thing!

Example

Data on patients by age, drug treatment, and visual acuity status:

Patient_ID	Age	Natamycin	VA_3M
1	38	0	0.02
2	37	0	0.26
3	36	1	0.48
4	66	1	0.7
5	73	1	0.48
6	24	0	0
7	66	0	0.34
8	68	1	0.4
9	41	0	0.18
10	56	1	0.5

Table

- Here, the data are organized by rows (subjects).
- Each column (*attribute*) is a variable measured on a subject.

Example

Another example: data on patients, by eye, visit time, and region within eye; each region is graded by different people.

Patient_ID	Eye	Visit	ROI	Grader	Outcome
1	Right	1	I_1	A	1.4
1	Right	1	I_1	B	1.3
1	Right	1	I_2	A	1.1
1	Right	2	I_1	A	1.5
1	Right	2	I_1	B	1.4
1	Left	1	LL_1	A	1.1
2	Right	1	I_1	A	1.2
2	Right	1	I_1	C	1.8

Key

- The values of some column, or combination of columns, serve to uniquely identify each unique subject or observation.
- If there are columns in a table that no two rows can share, those columns comprise a *superkey*.
- If you cannot delete any columns without losing this uniqueness, then the columns comprise a *key*.

Example

In this table, each row corresponds to a patient. The key consists of a single column.

Patient_ID	Age	Natamycin	VA_3M
1	38	0	0.02
2	37	0	0.26
3	36	1	0.48
4	66	1	0.7
5	73	1	0.48
6	24	0	0
7	66	0	0.34
8	68	1	0.4
9	41	0	0.18
10	56	1	0.5

Example

Here, it takes five columns to fully specify the key.

Patient_ID	Eye	Visit	ROI	Grader	Outcome
1	Right	1	I_1	A	1.4
1	Right	1	I_1	B	1.3
1	Right	1	I_2	A	1.1
1	Right	2	I_1	A	1.5
1	Right	2	I_1	B	1.4
1	Left	1	LL_1	A	1.1
2	Right	1	I_1	A	1.2
2	Right	1	I_1	C	1.8

Key

- Whether columns comprise a key or not does not depend on the specific data you have at any given time; it has to be true of all possible tables you could have.

Functional relationships

- A table represents a general functional relationship between the key and various attributes or variables.

Example

You are familiar with an identity matrix, such as this one in 3 dimensions:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example

This same information can be represented as a relational database, as a functional dependence between the row and column indices, and the entry.

i	j	I_{ij}
1	1	1
1	2	0
1	3	0
2	1	0
2	2	1
2	3	0
3	1	0
3	2	0
3	3	1

Example

- A matrix does not have to be represented in the usual two dimensional layout you see in math books.
- This different tabular representation expresses a matrix as a function of its indices.

Another Example

- But a relational database can represent higher order objects just as easily.
- The two dimensionality of a relational database table reflects the mapping of the arguments of a function to its values.

Another Example

Here is an interesting one:

i	j	k	a_{ijk}
1	1	1	0
1	1	2	1/4
1	2	1	1/4
1	2	2	0
2	1	1	1/4
2	1	2	0
2	2	1	0
2	2	2	1/4

Another Example

Here is the same information, represented in a different order:

i	j	k	a_{ijk}
1	1	1	0
1	2	2	0
2	1	2	0
2	2	1	0
1	1	2	$1/4$
1	2	1	$1/4$
2	1	1	$1/4$
2	2	2	$1/4$

Features

- Each cell contains one and only one value.
- The order of the rows does not matter.
- The order of the columns does not matter.
- There are no duplicate rows.

Example

Here is an example data table, keyed by patient. We record the outcome and the medications the person was taking.

Patient_ID	Outcome	Medications
10010	0	AB
10011	1	AB,CD
10020	1	CD
10013	0	CD,AB
10014	1	CD, AB
10015	1	
10016	0	CD
10041	1	AB CD

Example

After we at least canonicalize the representation of multiple drugs...

Patient_ID	Outcome	Medications
10010	0	AB
10011	1	AB,CD
10020	1	CD
10013	0	AB,CD
10014	1	AB,CD
10015	1	
10016	0	CD
10041	1	AB,CD

Why is this bad?

- We want to analyze medications, but all the medications together are stuffed into a single cell.
- If we are interested in a given medication, we have to break apart the possible values in the cell to find out what we want to know. (The values are not *atomic* with respect to what we want to do.)
- Therefore the table is irritating to maintain and query.
- It is hard to tell missing records from the absence of medication use.

Improvement

This is somewhat better. Still, we might need to anticipate the need for extra columns (such as medication EF).

Patient_ID	Outcome	AB	CD	EF
10010	0	0	0	0
10011	1	1	1	0
10020	1	1	1	0
10013	0	0	1	0
10014	1	1	1	0
10015	1	0	0	0
10016	0	0	1	0
10041	1	1	1	0

Improvement

Better still might be to represent the relation between patient and medication by its own table:

Patient_ID	Medication
10010	AB
10011	AB
10011	CD
10020	CD
10013	AB
10013	CD
10014	AB
10014	CD
10016	CD
10041	AB
10041	CD

First normal form

- We eliminated many-to-one relationships in the table, removing the need for complicated cell values.
- Two simple tables replace one cumbersome table.

Variation on this theme

- Do not make columns do the work of rows.
- Complications and obscurities in columns make for complicated programs.
- Changes to columns can change your analysis script.

Bad

Patient	Time	Acuity	Time	Acuity	Time	Acuity
1	0	0.8	18	0.9	21	0.9
2	18	0.25				
3	0	CF	7	1.3	21	1.3

Why

- Column names not unique; crucial information encoded in the ordering of columns.
- Need to add extra columns if you have extra observations
- Lots of missing data fields for patents that have fewer observations
- Many columns represent the same thing, such as time

Better

Patient	Time	Acuity
1	0	0.8
1	18	0.9
1	21	0.9
2	18	0.25
3	0	CF
3	7	1.3
3	21	1.3

Another example

Exper1-time	Exper1-temp	Exper2-time	Exper2-temp
0	24	0	10
1	26	3	14
2	28	5	12
3	27	NA	NA

Another example

- Here, the rows don't represent anything coherent. Each row is just the second observation of unrelated experiments.
- As before, similar things (time) are located in different columns.
- Missing values for some experiments due simply to the differing numbers of observations

Fix

Exper	Time	Temp
1	0	24
1	1	26
1	2	28
1	3	27
2	0	10
2	3	14
2	5	12

A different problem

The key for this table is *Patient* and *Time*.

Patient	Time	VA	Gender
1	21	0.1	m
1	93	0	m
2	20	0.8	f
3	21	0	m
3	90	0	m
4	90	0	f

Why is this bad?

- The gender information for each patient is needlessly duplicated—if you learn that patient 1 was gender `f`, you will have to change it in two places.
- A simple query, such has *how many women are there?* is complicated by the fact that some people are counted more than once.
- Here, the attribute (variable) *Gender* depends on *Patient* only, if we assume that *Gender* does not change over time.

Better

Remove the *partial key dependence* using a second table for gender, keeping VA in a table keyed by *Patient* and *Time*. Here is the new table:

Patient	Gender
2	f
3	m
4	f
1	m

A still different problem

The key for this table is *Patient* and *Time*.

Patient	Time	Logmar VA	Snellen
1	21	0.1	20/30
1	93	0	20/20
2	20	0.8	20/130
3	21	0	20/20
3	90	0	20/20
4	90	0	20/20

(Logmar and Snellen are two different ways to measure the same thing; they are essentially interconvertible.)

Why is this bad?

- Now, Snellen acuity is derived from Logmar acuity.
- If Logmar were the primary measurement, it would depend on the key; there is an acuity measurement for each patient at each time.
- But once you know the Logmar value, you know the Snellen value—no matter what the key is.
- This is a *transitive dependency* and is a different type of redundancy.
- Now, if you determine that a logMAR acuity is wrong, you must remember to change the Snellen also.
- Better to have a separate lookup table or formula.

Another transitive dependency

If you *know* that each medication is made by only one manufacturer, then this table has a transitive dependency:

Patient	Medication	Manufacturer
1	Azithromycin	Pfizer
1	Erythromycin	Lilly
2	Azithromycin	Pfizer
3	Moxifloxacin	Bayer
4	Erythromycin	Lilly
4	Moxifloxacin	Bayer

Fix:

Have a table keyed by *Patient* and *Medication*. Notice this table is "all key" now.

Patient	Medication
1	Azithromycin
1	Erythromycin
2	Azithromycin
3	Moxifloxacin
4	Erythromycin
4	Moxifloxacin

Fix

...and a second table keyed by *Medication*, containing the manufacturer information:

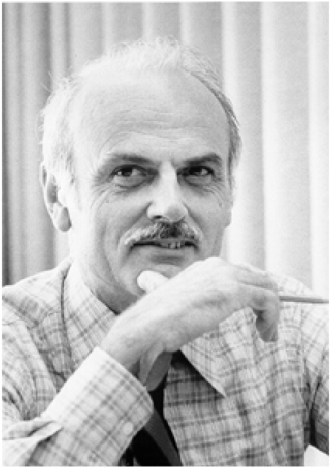
Medication	Manufacturer
Azithromycin	Pfizer
Erythromycin	Lilly
Moxifloxacin	Bayer

Third normal form

- No repeating groups
- No partial key dependencies
- No transitive dependencies

Columns that are not part of any possible key should depend on the key, the whole key, and nothing but the key...

So help you Codd



Pointers to future learning

- `[]` Brackets serve for both subsets of rows, and subsets of columns
- `merge` Selecting elements from one table to put into another (`table join`)
- `ddply` From the `plyr` package
- `cast` From `reshape2` package
- `melt` From `reshape2` package

Quick word

- Be pragmatic
- Know your tools—goal is get the job done correctly, on time, within budget
- Not trying to win a programming contest
- Clear, simple, documented, maintainable
- A simple clear program you can write correctly and maintain is better than a clever trick.
- "Do not be clever. Clever kills." — Steve Oualline, *Practical C Programming*
- PERL motto: There's More Than One Way To Do It (TMTOWTDI).
- Sometimes the latest new thing is great and sometimes it isn't.

Select

- Use the bracket operator to select rows from a table, based on a boolean variable for each row.
- Normally you would just have a data frame imported from somewhere. Just as an example, suppose we had a toy data frame:

```
ds1 <- data.frame(patid=1:5,gender=c("m","f","f","f","m"))
```

- Select rows corresponding to female gender:

```
ds1[ds$gender=="f",]
```

Select

`%in%`

Example

```
> xmp <- lep[lep$district %in% c("Chittor","Cuddapah","Adilabad"),]
> xmp
              state subject district population newcases prevalence year
1    Andhra Pradesh       1 Adilabad    2713960      542        337 2008
3    Andhra Pradesh       3  Chittor    4088653      404        254 2008
4    Andhra Pradesh       4 Cuddapah    2817001      364        233 2008
615  ANDHRA PRADESH       1 Adilabad    2749241      492        255 2009
617  ANDHRA PRADESH       3  Chittor    4141805      495        271 2009
618  ANDHRA PRADESH       4 Cuddapah    2853622      310        172 2009
1241 ANDHRA PRADESH       1 Adilabad    2784982      402        208 2010
1243 ANDHRA PRADESH       3  Chittor    4195649      457        250 2010
1244 ANDHRA PRADESH       4 Cuddapah    2890719      311        174 2010
1857 ANDHRA PRADESH       1 Adilabad    2737738      346        177 2011
1859 ANDHRA PRADESH       3  Chittor    4170468      342        192 2011
1860 ANDHRA PRADESH       4 Cuddapah    2884524      251        138 2011
2446 ANDHRA PRADESH       1 Adilabad    2766758      281        188 2012
2448 ANDHRA PRADESH       3  Chittor    4214675      356        229 2012
2449 ANDHRA PRADESH       4 Cuddapah    2915100      177        108 2012
3036 ANDHRA PRADESH       1 Adilabad    2796086      371        247 2013
3038 ANDHRA PRADESH       3  Chittor    4259351      336        206 2013
3039 ANDHRA PRADESH       4 Cuddapah    2946000      204        127 2013
```

Example

Note the previous example had the state information in a noncanonicalized form: the same information is represented in two different ways (capitalized versus not).

Merge

- `ds2 <- data.frame(patid=c(1,3,2,5),va.3m=c(0,0.6,0.2,0.8))`
- `merge(ds1,ds2,by="patid",all.x=TRUE)`

Notes

- If you work with multiple smaller normalized tables, this is a common operation prior to analysis.
- A common error is to omit the `all.x=TRUE` clause, and wonder where some of your records went.
- Another common error is to merge without making sure that the values of the common variable are actually the same—an identifier might be a string in one table, but a factor in another.
- Duplicate names are resolved by suffixes, which you can choose.

Notes

- `ddply` is used to break up a table into small data frames based on unique combinations of values of a set of variables (columns), perform an operation on them, and reassemble the results.
- Examples could include picking out baseline measurements, plotting different subsets of data in different colors, taking the difference between cases and controls, and many others.

Notes

- `melt` and `cast` are used in reshaping tables (long to wide form), and similar common tasks.
- From the `reshape2` package

Learn more

- Oppel, A. Databases DeMYSTiFieD, 2d ed. 2011, McGraw-Hill.
- Date CJ, Introduction to Database Systems, 8th Ed, 2004, Addison Wesley.