

# Database normalization





# What is normalization?

- Re-organizing datasets to emphasize relationships and modularity
  - There are increasingly strict levels of normalization (“normal forms”)
- “Tidy” data is a form of normalization (similar to 3rd normal form)

```
#>   name quiz1 quiz2 test1
#> 1 Billy <NA>    D     C
#> 2 Suzy   F    <NA> <NA>
#> 3 Lionel B     C     B
#> 4 Jenny  A     A     B
```

“Messy” data



```
#>   name assessment grade
#> 1 Billy quiz1      NA
#> 2 Billy quiz2       D
#> 3 Billy test1       C
#> 4 Jenny quiz1       A
#> 5 Jenny quiz2       A
#> 6 Jenny test1       B
#> 7 Lionel quiz1      B
#> 8 Lionel quiz2      C
#> 9 Lionel test1      B
#> 10 Suzy quiz1       F
#> # ...
```

Normalized/“tidied” data



# Why normalize?

- Protects against errors and makes data entry easier

## Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

← A less-normalized setup can make contradictions easier (e.g. conflicting address info).

- Reduces the need to refactor databases as more data is added. (*What if we wanted to add ZIP codes?*)
- Simplifies queries
  - Less shuffling columns, manual parsing, or wrangling nested data
  - Consistency makes statistical analyses more straightforward



## What does this have to do with INFO 523?

- Essential data wrangling step for more complex datasets (tidyverse needs “tidy” data == normalization)
- Often a goal of reshaping or organizing data relationally (see Kabacoff (2015), Ch. 5; Wickham and Grolemund (2016), Ch. 12-13)
- Normalization’s scalability makes it a very good idea for “big” data (data stores, data warehousing, etc.)



# Real-world applications

- Normalization is a fundamental principle of “relational database management systems” (RDBMS): MySQL, Postgres, Oracle Database, Apache Hive, etc.
- Proper use of databases with these systems requires an understanding of database normalization.



**Let's try it!**





# Fundamental ideas

- Sometimes data should might work better in multiple tables!
  - The more narrowly a table describes a relationship, the more “normalized”
- Rows have “primary keys” that identify them, and then additional columns
  - You get to pick which columns are “primary keys”
  - Primary keys are used to cross-reference different tables
  - The exact terminology is very particular, I’ll stick to “(primary) keys” and “columns”

# Normalization levels (for reference)

	UNF (1970)	1NF (1970)	2NF (1971)	3NF (1971)	EKNF (1982)	BCNF (1974)	4NF (1977)	ETNF (2012)	5NF (1979)	DKNF (1981)	6NF (2003)
Primary key (no duplicate tuples) <sup>[6]</sup>	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Atomic columns (cells cannot have tables as values) <sup>[6]</sup>	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
No partial functional dependencies of non-prime attributes on candidate keys <sup>[6]</sup>	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
No transitive functional dependencies of non-prime attributes on candidate keys <sup>[6]</sup>	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with an elementary prime attribute	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	N/A
Every non-trivial functional dependency begins with a superkey	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	N/A
Every non-trivial multivalued dependency begins with a superkey	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	N/A
Every join dependency has a superkey component <sup>[9]</sup>	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	N/A
Every join dependency has only superkey components	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	N/A
Every constraint is a consequence of domain constraints and key constraints	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Every join dependency is trivial	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓





# Conclusion





# Conclusions

- You may never need to use 6NF, but normalization is handy to think about
  - (And you've probably already used it!)
- Be careful of your resources—if you want to know more, read up on the original papers
  - Older: Edgar F. Codd
  - Newer: Christopher J Date