



# MODELING A SOLID DATABASE FOR YOUR RUBY FRAMEWORK

**Zoran Majstorović**

[github.com/zmajstor](https://github.com/zmajstor)

[twitter.com/z0maj](https://twitter.com/z0maj)



You want to build an *awesome*, full-stack web app or maybe just a backend (API) server, so you:

1. pick your favourite ruby framework
2. pick an ORM which plays nicely with your ruby framework
3. pick a database which plays nicely with your ORM

Finally, you can start coding and modeling your data ...


# Easy Challenge



Person	Telephone
Joe	555-123-4567, 555-192-1234, 123-4567-123
Jack	555-099-0987, 555-098-7654 - Ext 45
Jill	555-333-4444

Person	Telephone
Joe	555-123-4567
Joe	555-192-1234
Joe	123-4567-123
Jack	555-099-0987
Jack	555-098-7654 - Ext 45
Jill	555-333-4444

# Normalizing a Database Design



Person	Telephone
Joe	555-123-4567, 555-192-1234, 123-4567-123
Jack	555-099-0987, 555-098-7654 - Ext 45
...	...

Person	Telephone
Joe	555-123-4567
Joe	555-192-1234
Joe	123-4567-123
Jack	555-099-0987
Jack	555-098-7654 - Ext 45

split the data into an "atomic" (i.e. *indivisible*) entities: single phone numbers

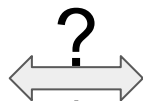
More about 1st Normal Form: [https://en.wikipedia.org/wiki/First\\_normal\\_form](https://en.wikipedia.org/wiki/First_normal_form)

# The Dilemma

Denormalization a.k.a. favor columns over tables ... because we have to avoid “*expensive joins*” (?)

Are joins generally expensive? And what is the price for avoiding them?

Person	Apartment	Floor
Joe	A	1
Jack	B	2
Jill	B	2



Person	Apartment
Joe	A
Jack	B
Jill	B

Apartment	Floor
A	1
B	2
X	3

# Normalizing a Database Design

Person	Apartment	Floor
Joe	A	1
Jack	B	2
Jill	B	2

The **functional dependency**  $\{\text{Person}\} \rightarrow \{\text{Floor}\}$  applies; that is, if we know the person, we know the floor.

Furthermore:

$\{\text{Person}\} \rightarrow \{\text{Apartment}\}$

$\{\text{Apartment}\}$  does not  $\rightarrow \{\text{Person}\}$

$\{\text{Apartment}\} \rightarrow \{\text{floor}\}$

Therefore  $\{\text{Person}\} \rightarrow \{\text{Floor}\}$  is a **transitive dependency**.

Transitive dependency occurred because a *non-key attribute* (*Apartment*) was determining another *non-key attribute* (*Floor*).

# 3rd Normal Form (3NF)

1. Requiring existence of "the key" ensures that the table is in 1NF
2. requiring that *non-key attributes* be dependent on "*the whole key*" ensures 2NF
3. further requiring that *non-key attributes* be dependent on "*nothing but the key*" ensures 3NF

# 3rd Normal Form (3NF)

- improve *database processing* while minimizing storage costs
- ideal for online transaction processing (OLTP)
- most 3NF tables are free of ***update***, ***insertion***, and ***deletion anomalies***\*

---

\* other, few cases, affected by such anomalies usually fall short of the higher normal forms: 4NF or 5NF



# Deletion Anomaly

Person	Apartment	Floor
Joe	A	1
Jack	B	2
Jill	B	2

Joe moves out of the apartment A, so by deleting Joe, we are also losing information of the apartment A (that is on the 1st floor).

# Update Anomaly

Person	Apartment	Floor
Joe	A	1
Jack	B	2
Jill	B	2



Person	Apartment	Floor
Joe	A	1
Jack	B	2
Jill	A	2

Jill moves from the apartment B to the apartment A, but for some reason, only Apartment column got updated.

After some time, we can't tell on which floor is the Apartment A.

???

# Insertion Anomaly

Person	Apartment	Floor
Joe	A	1
Jack	B	2
Jill	B	2
Jim	X	?

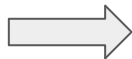
Jim moves into apartment X,  
but where to lookup for the floor?

On which floor is the apartment X?

A database-management system (DBMS) can work only with the information that we put explicitly into its tables.

# Normalized table to meet the 3rd Normal Form

Person	Apartment	Floor
Joe	A	1
Jack	B	2
Jill	B	2



Person	Apartment
Joe	A
Jack	B
Jill	B

Apartment	Floor
A	1
B	2
X	3

# “Partial Denormalization”

Keep the transitive dependent columns, in combination with another “lookup table”

Person	Apartment	Floor
Joe	A	1
Jack	B	2
Jill	B	2

Apartment	Floor
A	1
B	2
X	3

WAT???

# Denormalizing to avoid “*expensive joins*”?

*“It sounds convincing, but it doesn't hold water.”*

Peter Wone's epic post: <http://stackoverflow.com/a/174047/3452582>

From ***Introduction to Database Systems***, which is the definitive textbook on database theory and design, in its **8th edition**:

- **Some** of them hold for **special cases**
- **All** of them **fail** to pay off for **general use**
- **All** of them are **significantly worse** for other special cases

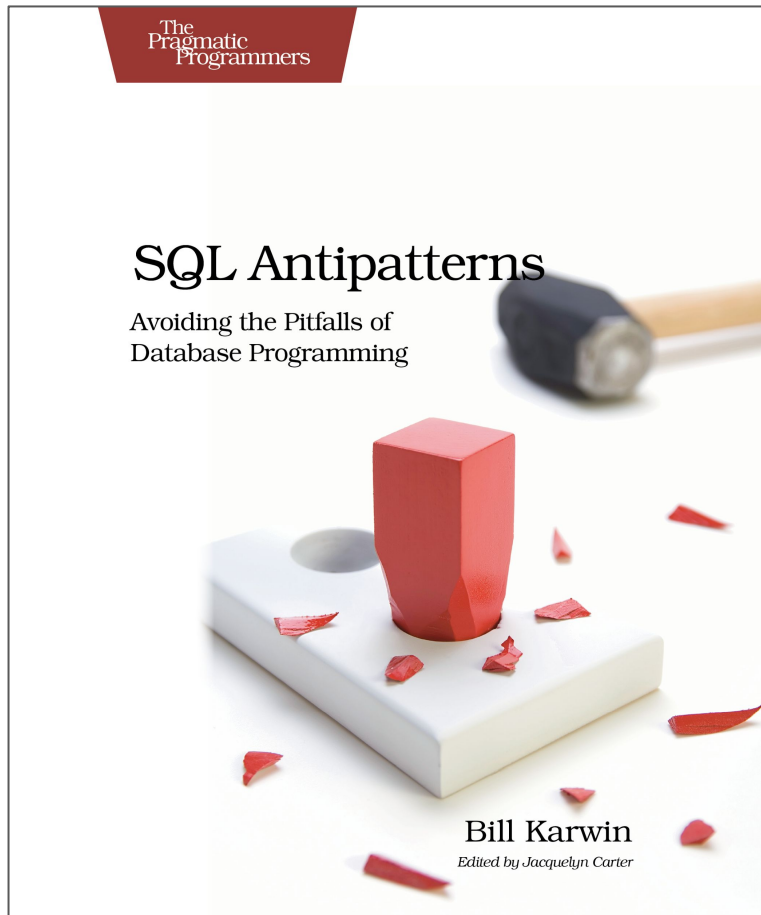
ALWAYS normalise OLTP. Denormalise OLAP if you think it will help.

# Common Sense

*“Rules of normalization aren’t esoteric or complicated. They’re really just a commonsense technique to reduce redundancy and improve consistency of data.”*

***SQL Antipatterns*** by Bill Karwin

Copyright © 2014, The Pragmatic Bookshelf.



# Summary

The application layer can become riddled with bugs if the data layer is too permissive, so here are 5 simple rules for a solid DB design:

1. Normalize tables up to the 3NF (and don't be afraid of SQL Joins)
2. Define foreign keys on DB level (ORM can't be fully trusted on that)
3. Define UNIQUE index where needed (ORM can't guarantee data uniqueness)
4. Define other validations on DB level (e.g. Not Null, Default Value)
5. Define index on every foreign key and any column that will appear in any *where clause* (for better query performance)



# Sources

## Wikipedia

- [https://en.wikipedia.org/wiki/Transitive\\_dependency](https://en.wikipedia.org/wiki/Transitive_dependency)
- [https://en.wikipedia.org/wiki/Third\\_normal\\_form](https://en.wikipedia.org/wiki/Third_normal_form)
- [https://en.wikipedia.org/wiki/Database\\_normalization](https://en.wikipedia.org/wiki/Database_normalization)

## StackOverflow

- <http://stackoverflow.com/a/174047/3452582>
- <http://stackoverflow.com/a/59522>

## Quora

- <https://www.quora.com/What-is-the-difference-between-OLTP-and-OLAP>

THE END