

ORMs: Strengths, Weaknesses, and Building a PostgreSQL-specific ORM

Jonathan S. Katz

November 7, 2009 - PGDay.eu

Outline

- Overview & Dirty Work
- The “ORM Patterns”
- Seriously, Why?
- Dissecting & Discussing ORM Features
- Peeking at some code
- Scalability & the PostgreSQL ORM

Nota Bene

- I like working with SQL
- ORMs are another tool for solving problems, not necessarily **the** tool

Overview: We're Jumping In

- Let's build an address book!

Requirements

- Multi-user
- Store first name, last name, location
- Searchable / Sortable by location
- (Pretend this is a web app)

Example 1:
Naïvely We Commence

Notes:

- Had to handle “SQL semantics” every-time
- Reusable? (or too contrived)
- Ruby “Hash” objects: no errors if no value for a key
- Also, isn't Ruby object-oriented?

Try #2: Make it “Developer Friendly”

- Make it geared towards my development language (in this case Ruby)
- Use Ruby to encapsulate SQL

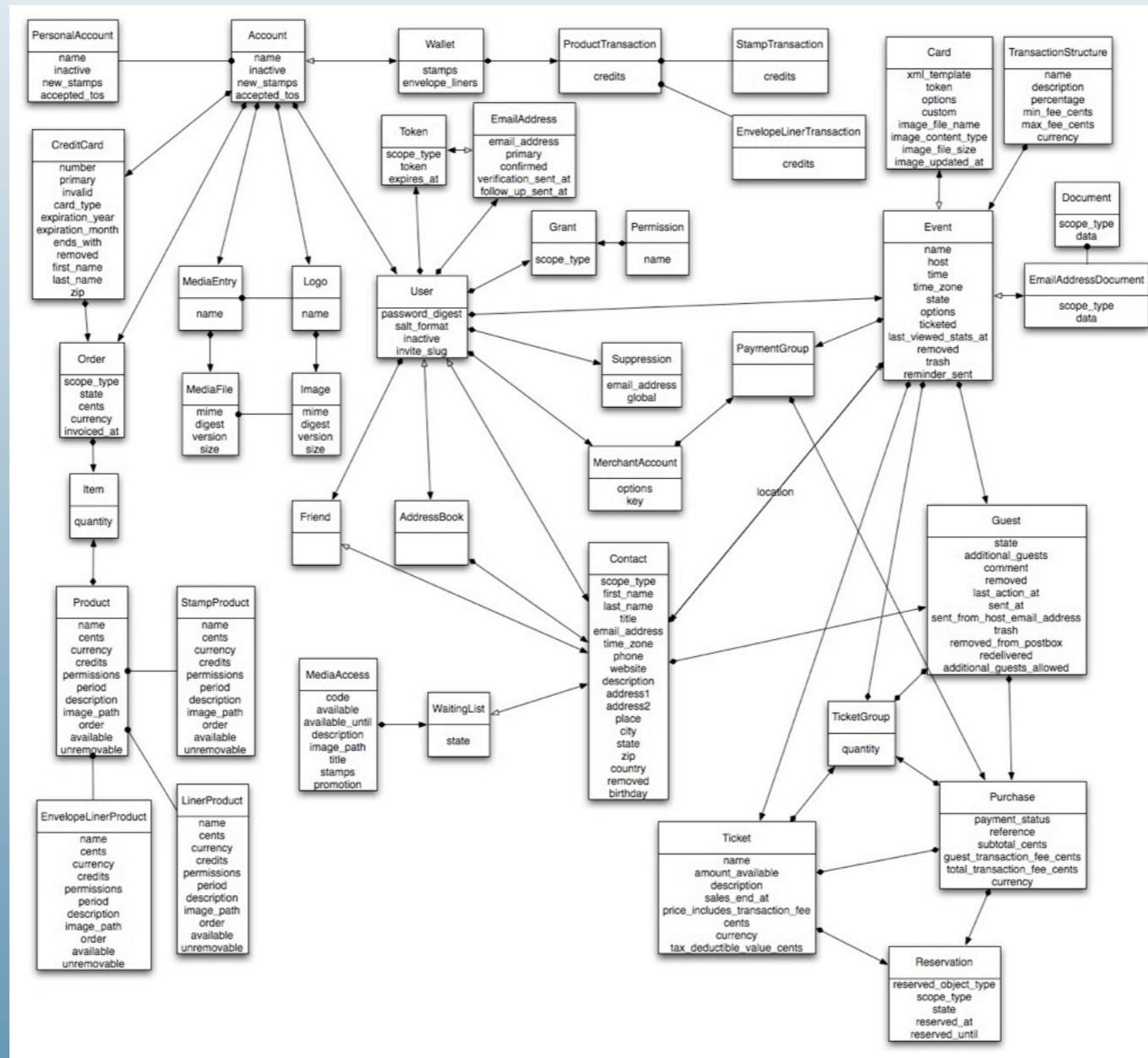
We Try Again

Notes

- More code to setup, but less to accomplish our tasks
- Reading / Writing data much more familiar to developer

Eureka!

- We made a simple object-relational mapper!
- Made our DB interactions more programmatic
- We could continue using this, but imagine if our domain were something like...



Formality Sake

- Object-relational mapping (ORM) - a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages.

(Source: http://en.wikipedia.org/wiki/Object-relational_mapping)

Examples:

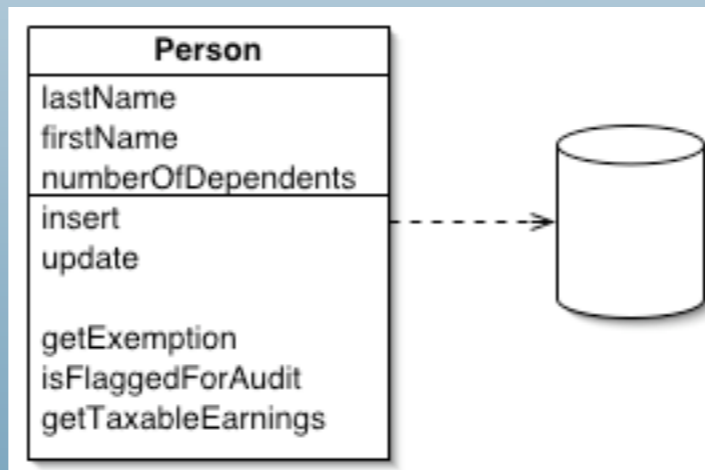
- Ruby:
 - ActiveRecord (<http://www.rubyonrails.org>)
 - DataMapper (<http://datamapper.org/>)
 - Sequel (<http://sequel.rubyforge.org/>)
- Python
 - SQLAlchemy (<http://www.sqlalchemy.org>)
- PHP
 - Propel (<http://propel.phpdb.org>)
 - Doctrine (<http://www.doctrine-project.org/>)

Two Notable Patterns

- Courtesy of and many thanks to Martin Fowler for the contents on the next two slides
- Source: “*Patterns of Enterprise Application Architecture*” (<http://martinfowler.com/books.html#eaa>)

Active Record

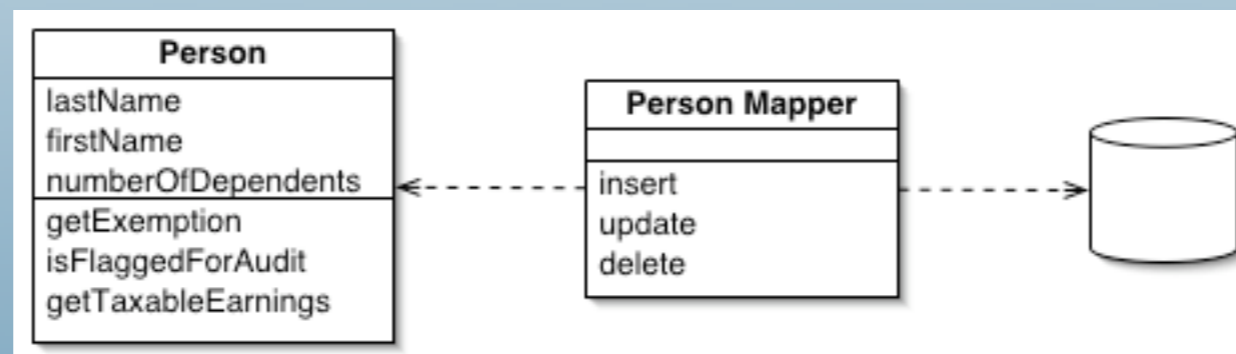
- “An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data.”



Source: <http://www.martinfowler.com/eaCatalog/activeRecord.html>

Data Mapper

- *“A layer of mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.”*



Source: <http://martinfowler.com/eaCatalog/dataMapper.html>

A Quick Interjection

- Why not combine parts of both?
- We will look at:
 - ActiveRecord (eponymous)
 - SQLAlchemy (data mapper)

So, Why?

- Abstraction: Take SQL out of daily development (gasp!)
- Portability
- Development Speed
- Relationship management, or managing relations
- Expressibility via API
- Features!

SQL: like ASM for a developer talking to databases - nice to have it abstracted out

portability: can use with postgresql, mysql, mssql, oracle -- but still keep the same application code

development speed: abstracted a lot, less code to write; familiarity with an OSS ORM allows for quick turnaround on new projects

Immediate Drawbacks

- Yet another layer of code
- Surrender some control
- Can be learning another language

Design, Design, Design

- ORMs are no panacea: still need good software design
- API level
- Developer level

ActiveRecord

- Core part of “Ruby on Rails”
- (Being overhauled for Rails 3)
- MVC = Model-View-Controller
- Let’s look at some code, then “Rails



Example 3!

Interface Notes

- create, save vs. create!, save!
- true/false vs. Ruby exception -- consistency?

Validations

- can test constraints before committing record to database
- can add custom validations if ActiveRecord does not provide one that fits
- returns a special “Error” object that can be parsed if there are failures (fun)

Callbacks

- Trigger methods to run before, after initialize / create / save / update
- Useful for forcing data mutation or running special command

Named Scopes

- Way of writing “programmatic SQL”
- In ActiveRecord, does not load data immediately
- `Contact.begins_with('s').visible.scoped(:conditions => { :created_at => Date.today })`
- Note: if I used “ILIKE” suddenly code is not portable(!) - not “developer-proof”

Which Reminds Me...

- Notice how ActiveRecord was smart enough to format the data types correctly?
- ActiveRecord also takes care of quoting and avoiding SQL injections *if used properly*
- ~~User.all(:conditions => "name = #{name}")~~
- User.all(:conditions => { :name => name })

Associations

- `has_many :blahs`
- `has_one :blah`
- `belongs_to :blah`
- `has_and_belongs_to_many :blahs`
- (and many options for configuring these, e.g. `:dependent => :destroy`)

Single-Table Inheritance

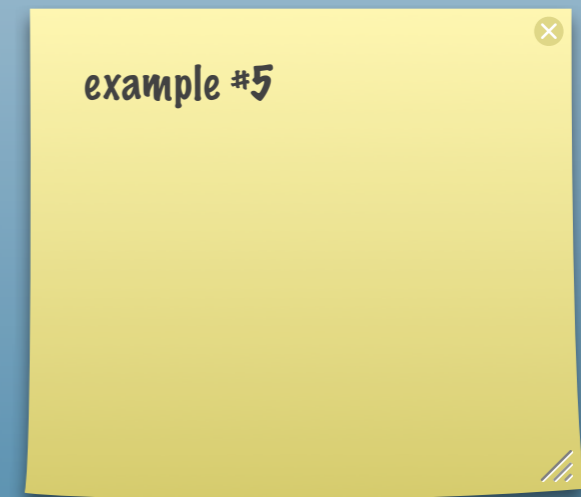
- `class Friend < Contact`
- `belongs_to :user`
- `end`
- ActiveRecord smart enough to reify appropriate class when `User#contacts` is called
- Note: Not using PostgreSQL's built-in inheritance mechanism!

Serialization

- Take a Ruby data type, store it, load it when record is reloaded
- `serialize :data, MyOwnClass`

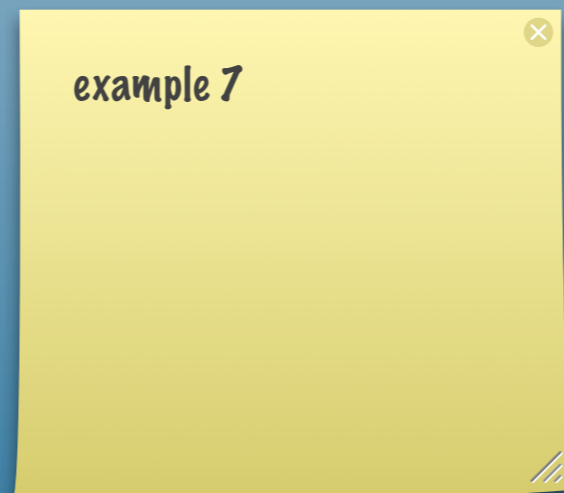
Transactions / Locking

- Supports PostgreSQL transactions (fakes it for some other databases)
- Can use locking for business logic purposes
- Occurs at application level
- Optimistic vs. Pessimistic



Managing the Schema: Migrations

- Not part of the “ORM” per se, but worth a mention
- Helps keep track of “state” of the database and reproduce / tear down as needed



This sound too good...

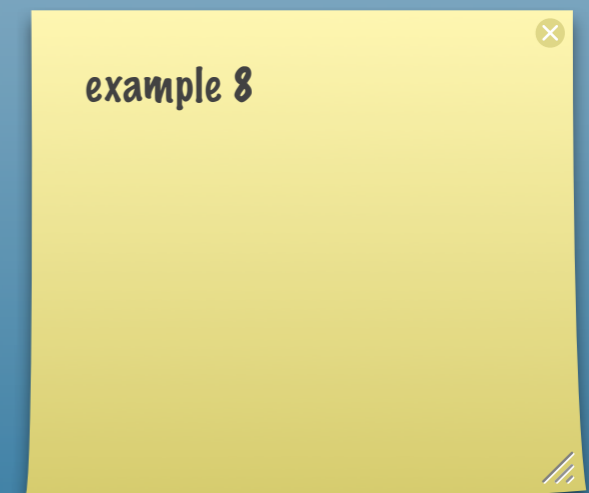
- You're right!
- Well, you're also wrong. But you're partially right, there are some issues.

How is the SQL generated?

- In ActiveRecord: all over the place
- Since can use multiple database adapters, does include database-specific SQL
- Examples for PostgreSQL

More on ActiveRecord SQL

- Sometimes, ORM cannot do it all, e.g. a complex sort
- `Model.find_by_sql`
- Lose out on “Scope” features when using this method, e.g. fast pagination



Other “Missing Features”

- Cursors: ActiveRecord either loads everything, nothing, or in X batches
- Datatypes and Modules: e.g. XML, hstore
- Uniqueness / Constraint Errors: Treated as “SQL Error
- Functional indexes
- Inheritance / Partitions
- Functions (a lot of them)
- Prepare / Execute ?

Shifting Gears: SQLAlchemy

- uses “data mapper” pattern
- separates SQL generation from actual objects
- I will give more of an overview due to less-familiarity with Python / SQLAlchemy
- *Source:* <http://www.sqlalchemy.org/docs/05/ormtutorial.html>

Noteworthy Points

- Separates SQL generation from mapper itself (sounds familiar?)
- SQL generation: “not necessarily clean, but programmatic”

Introducing “Postgresina”


- So pre-alpha it's Ω
- Main idea: build on ORM towards specific PostgreSQL features

Ideas

- Prepare / Execute - implicitly (hard) and explicitly (easy[-ier])
- Inheritance - table-wise vs. string-column / index and compare performance differences
- Easier to access different data-types / methods
- Loading into memory: when it's time.

Starting-Points

- SQL-generator:
 - Must have solid API
 - Must be accessible to developer

A yellow sticky note with a close button in the top right corner and a corner icon in the bottom right corner. The text on the note is "Idea: want to be programmatic".

Idea: want to be programmatic

Scalability

- Want to remain programmatic and scale
- Be able to maintain roles for both developer + DBA

Conclusions

- ORMs can help developers start-off quickly
- Issues with scalability + taking advantage of PostgreSQL features
- Should be possible to extend ORM functionality, but will not completely remove the need for SQL

Questions?

- Now
- jonathan.katz@excoventures.com
- jkatz05@gmail.com
- Twitter: jkatz05
- github.com/jkatz/postgresina