Needs

- Monkeys can handle structure migrations cleanly
- Monkeys can edit mod data *without* touching migrations
- Easy for Monkeys to understand data structure
- Monkeys can grab content from running deck into mod
- Platypuses can maintain seed data in files (without bootstrapping from db)
- Monkeys can add test seed data (will be needed for testing mods)
- Sharks don't have to think about any of this. They just run `decko update`

Data Representation

Current:

- fixtures in card/db/seed
- json lists in card/db/migrate_cards/data
- yml in card/db/migrate_card/data/cards.yml
- content in card/db/migrate_card/data/cards

Proposed

Specify mod to seed from in config. Move fixtures to:

- [modname]/data/fixtures/real # *contains cards.yml, card_actions.yml, etc*
- [modname]/data/fixtures/test

Replace all the other data representations with "pod" data in

[modname]/data/real.yml
[modname]/data/test.yml
[modname]/data/real/OTHER.yml # *(referred to by real.yml)*
[modname]/data/test/OTHER.yml # *(referred to by test.yml)*

Rationale

- only one authoritative copy to maintain
- can avoid writing migrations (usually)
- can review changes easily in github
- easy to edit by hand when needed
- simplify seed updating tasks (easy to regenerate fixtures)
- mechanism for mod-specific cards

Pod Data

Each YAML entry is *either* a card "pod" or a relative reference to another YAML file. A pod looks something like this:

```
name: My Card
codename: :mycard
type: typemark
content: mycontent
```

Pods support any **normal argument** to Card.create (or, more precisely, to **#ensure_card**), including **subcards, fields, skip, trigger,** etc. We also handle some special ones:

- **`user`**: mark of user credited with act
- **`time:`** alter create/update time in timecop. Val is integer (Time.now.to_i). If prefaced by a "+" or "-", we compute a time in the future or past (respectively) from Time.now
- **`conflict:`** what to do if the name and/or codename exist - *NOT YET IMPLEMENTED*

Conflicts

The conflict policies, as spelled out below, assume every card is in one of three states:

1. *new*: there is no card with this name or this codename
2. *pristine*: a card with this name and/or codename exists but card has not been edited by anyone other than cardbot
3. *altered:* card has been edited by someone other than just cardbot

For **(a) entries with no codename** and **(b) entries with a codename and a name that is not already in use**, the conflict handling is straightforward:

	missing	*pristine*	*altered*
defer	create	do nothing	do nothing
default	create	update	do nothing
override	create	update	update

Things get more complicated when the entry has a codename and the name is in use. Here is what we do when **(c) the name is in use but the codename is not**. (Column headings refer to the state of the existing card with the entry's *name*.)

	pristine	*altered*
defer	create with new name	create with new name
default	update	create with new name
override	update	update

Finally, here's what we do when **(d) both the name *and* the codename are already in use**. (Column headings refer to the state of the existing card with the entry's *codename*.)

	pristine	*altered*
defer	do nothing	do nothing
default	update all but *name*	do nothing
override	alter conflicting name & update *codename card*	alter conflicting name & update *codename card*

For altering names, the standard behavior is to increment the name (eg *myname 1*).

Seeding / Migrating

Currently there are 4 types of migrations: structure and "card" migrations for core and deck respectively. We should **reduce to two types**, each with its own mod directory:

- **data/schema/** # structure migrations
- **data/transform/** # card migrations that can't be handled by YAML ingestion, including:
  - renaming cards without codenames
  - deleting cards without codename
  - patterned content changes
  - non-card changes (lookups, etc)

When you run **decko** (or *card*) **setup,** we progress through these stages:

1. **Seed**
    a. from specifiable fixtures
    b. includes migration tables
2. **Update**
    a. **migrate:port** - confirm ported to new migration system)
    b. **migrate:schema** - schema migrations (do not load card)
    c. **migrate:recode** - handle changed codenames
    d. **eat** - ingest pod data
    e. **migrate:transform** - transform migrations
    f. **reset** - reset tmp dirs and cache
    g. **mod:install**
    h. **mod:uninstall**
    i. **mod:symlink** - update symlinks

It would be nice to be able to add scripts to this, too. That way we could support more significant data transformations that require logic *without* locking the database. (Note: eating does not lock the db).

Optimizations

- only eat in mods in which yaml has been updated since the mod card's latest update
-

CLI

There are three main groups of commands:

- *card* commands, eg `card eat`. Note these should *not* require the decko gem.
- *decko* commands, which include all the card commands (eg `decko eat`) AND some others that require the decko gem.
- *rake* commands, which include (nearly) all the decko commands (eg `rake decko:update`) AND some additional rarely used commands for platypuses. Some special cases may not be executable as rake commands (eg `decko new`(?))

Card commands

For Sharks

```
new                     create a new deck
setup                   populate a database
update        (or u)    run data updates
version       (or v)    card gem version
help          (or h)    show this text
```

Eventually `card version` and `decko version` should be the same. But for now they're different and they output the version of their respective gems.

For Monkeys

```
card console
card dbconsole
card runner

card eat                # import from yaml
 -n  --name              import only card with name
                         (handles : for codenames)
 -m, --mod MOD           only eat cards in given mod
 -u, --user USER         user to credit unless specified
                         (otherwise uses Decko Bot)
 -p, --podtype TYPE      pod type: real, test, or all
 -v, --verbose           output progress info and error backtraces
 -e  --env               environment (test, production, etc)
 -h  --help
```

card sow *# download card yml*

```
    -n, --name NAME              export card with name/mark
                                 (handles : and ~ prefixes)
    -i, --items                  also export card items (with -n)
    -o, --only-items             only export card items (with -n)
    -c, --cql CQL                export cards found by CQL (in JSON format)
    -m, --mod MOD                output yaml to mod
    -p, --podtype PODTYPE        podtype to dump
                                (real or test. default based on current env)
    -t, --field-tags FIELD_TAGS  comma-separated list of field tag marks
    -e, --env ENV                environment to dump from (when local)
    -u  --url                    source card details from url
    -h  --help
```

```
card generate # auto-generate code
  mod name
  set mod pattern anchor1 [, anchor2, anchor3..]
  migration name
    -m  --mod
    --schema
```

We should make the most important mod-developer operations really easy to remember and use. (`decko generate card:set` is much harder to remember than `card generate set`). **`set`** and **`mod`** are definitely the most important of these two.

```
card reset     # by default clears both cache and tmpfiles
 -c --cache    # cache only
 -t --tmpfiles # tmpfiles only
```

```
card rspec DECKO/CARD ARGS -- RSPEC ARGS
 -f, --file FILENAME             Run specs with filename
                                 (with or without _rspec.rb)
 -m, --mod MODNAME               Run all specs for a mod
 -s --[no-]simplecov             Run with simplecov
    --pry-rescue                 Run with pry-rescue
    --[no]-spring                Run with spring
```

Would be nice if rspec and cucumber didn't have to have the -- separating the arguments we pass on from those we don't. (low priority)

decko commands

Decko adds a couple new commands:

```
decko server (for sharks)
decko cucumber (for monkeys
...
```

...and some commands have additional meaning in decko:

`decko rspec`

... also calls decko specs, whereas card rspec only calls card specs

`decko update`

Call `card update` and `rake card:mod:symlink`

Note: it would be nice if commands with mod-specific options (eat, sow, generate, etc) could be smart about the current mod they're in if called from within a mod.

rake commands

Rake is clearly a powerful tool for organizing a quick api for tasks that are sometimes performed independently and other times as part of a larger process. We wouldn't want to write separate scripts for all those tasks.

```
rake card:eat                          # Ingests card data from mod yaml

rake card:mod:install                  # install all mods
rake card:mod:leftover                 # list mods still installed but not configured for use
rake card:mod:list                     # list current mods in load order
rake card:mod:symlink                  # symlink from deck public/{modname} to mod's public
                                         directory
rake card:mod:uninstall                # uninstall leftover mods

rake card:reset_cache                  # Resets cache
rake card:reset_tmp                    # reset with an empty tmp directory

rake card:seed                         # Loads seed data
rake card:seed:build                   # completely regenerate seed fixtures starting with
                                         dependee seed fixtures
rake card:seed:replant                 # Truncates tables of each database for current
                                         environment and loads the seeds(alias for
                                         db:seed:replant)
rake card:seed:update                  # regenerate seed fixtures quickly from current
fixtures
rake card:setup                        # Creates the database, loads the schema, initializes
                                         seed data, and adds symlinks to public directories


rake card:sow                          # Exports card data to mod yaml

rake card:update                       # Runs migrations, installs mods, and updates symlinks
```

```
rake card:migrate                       # migrate structure and cards
rake card:migrate:redo                  # Redo the transform migration given by VERSION
rake card:migrate:schema                # run structure migrations
rake card:migrate:stamp[type]           # write the version to a file (not usually called
directly)
rake card:migrate:transform             # run transform migrations
```

Additional Notes

- Every `rake card:*task*` call should also be callable as `rake decko:*task*`. (but not necessarily vice versa)
- `rake -T` shows all rake commands with a `desc` call).
- We should make sure all the rails commands we want/need work as card/decko. Any other commands should not show up from `rake -T`.
- Generally speaking, rake commands should pass on basic rake args (eg --trace) and maintain environments
- All tasks should have tests