From MySQL to MongoDB at Sluggy.com

Brendan W. McAdams

Sluggy Freelance, LLC (sluggy.com) / Evil Monkey Labs, LLC.

May 2010 / Mongo NYC

Outline

Outline

Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- Advertising Revenues (e.g. Paying the Bills): `I tend to think of advertising as a finky spastic mentally retard cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh! food!"`
- Not Google. Not Trying to *be* Google. Can't afford to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. No ability to cover operations costs, downtime, bugs etc. (all of it means 3am wakeup).

Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- Advertising Revenues (e.g. Paying the Bills): `I tend to think of advertising as a finky spastic mentally retard cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh! food!"`
- Not Google. Not Trying to *be* Google. Can't afford to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. No ability to cover operations costs, downtime, bugs etc. (all of it means 3am wakeup).

Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- Advertising Revenues (e.g. Paying the Bills): `I tend to think of advertising as a finky spastic mentally retard cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh! food!"`
- Not Google. Not Trying to *be* Google. Can't afford to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. No ability to cover operations costs, downtime, bugs etc. (all of it means 3am wakeup).

Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- Advertising Revenues (e.g. Paying the Bills): `I tend to think of advertising as a finky spastic mentally retard cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh! food!"`
- Not Google. Not Trying to *be* Google. Can't afford to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. No ability to cover operations costs, downtime, bugs etc. (all of it means 3am wakeup).

Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- Advertising Revenues (e.g. Paying the Bills): `I tend to think of advertising as a finky spastic mentally retard cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh! food!"`
- Not Google. Not Trying to *be* Google. Can't afford to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. No ability to cover operations costs, downtime, bugs etc. (all of it means 3am wakeup).

Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- Advertising Revenues (e.g. Paying the Bills): `I tend to think of advertising as a finky spastic mentally retard cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh! food!"`
- Not Google. Not Trying to *be* Google. Can't afford to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. No ability to cover operations costs, downtime, bugs etc. (all of it means 3am wakeup).

Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- Advertising Revenues (e.g. Paying the Bills): `I tend to think of advertising as a finky spastic mentally retard cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh! food!"`
- Not Google. Not Trying to *be* Google. Can't afford to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. No ability to cover operations costs, downtime, bugs etc. (all of it means 3am wakeup).

Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- Advertising Revenues (e.g. Paying the Bills): `I tend to think of advertising as a finky spastic mentally retard cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh! food!"`
- Not Google. Not Trying to *be* Google. Can't afford to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. No ability to cover operations costs, downtime, bugs etc. (all of it means 3am wakeup).

Sluggy.com Rundown
Some Stats

- 50GB/day (1.5TB of traffic/month on a single virtual box)
- 13 years of daily comics = 6500 image files (just for the comics)
- Artist is frequently late in updating. System has to handle random unexpected cache flushes & data updates.
- LUMP Stack (Lighttpd, Ubuntu, MongoDB & Python)

Outline

Sluggy.com Technology History
1997 - 2000

- August 25, 1997: Site Started - Static HTML generated via midnight cron executing Perl. No dynamic content. HTML Editing for news, navigation, etc.
- File format requires globs:
  ```
  000217a.gif
  000217b.gif
  000217c.gif
  ```
- All make up the panels for February 17, 2000. Artist likes & understands this format. Code looks for `yyMMdd*.(gif|jpg)` via glob and organizes them in order.
- *2000* - Original Developers split off and formed KeenSpot.com using same code & navigation concepts.

Sluggy.com Technology History
2002

- Rewrite Began using MySQL & PHP to allow for “No More HTML Editing”
- Some Features: Dynamic headlines, news, predefined templates, dynamic navigation. “Members Only Club” (behind the scenes features, sketches, etc)
- *First Folly* - Reading MySQL and dynamically generating on each page request magnitudes harder than static HTML. I/O DOS.
- Moved to smarty template caching, generating on-disk cache file upon first request (expires at midnight)
- Next 4 years became hellish with Midnight DOS’ as everyone hit site and caches regenerated.

Sluggy.com Technology History
2006

- Server move (For cost reasons) introduced new architecture problems
- Perceived cost savings pushed a move from SCSI/SAS disks to SATA
  - Between template files & comic file globs. Disks couldn't keep up.
  - Implemented **memcached** to cache templates off-disk, in memory. Cached glob results (but not files). Cached anything else not likely to change - expiry set to a week (midnight for "index")
  - Sessions performed poorly in both disk and MySQL - caching in memcached helped.
- Apache began crushing us on memory & disk I/O.
  - PHP isn't thread safe; requires forked Apache workers (children are *EXPENSIVE*)
  - Migrated to Lighttpd + FastCGI - IO & RAM usage of webserver & PHP became negligible (Lots of tweaking of handling of static files esp. FAM)

Sluggy.com Technology History
2009

- Rewrote the system in Pylons (Python + SQLAlchemy[MySQL])
- Integrated **Beaker** caching decorators (templates & code blocks) - simplified adding caching code at need.
- Clean ORM model, light & fast with lots of caching.
- Ran significantly better than on PHP - infinitely more tunable; sensible and sane.
- **memcached** continued to become a big, rickety crutch (cascading failure sucks)

Sluggy.com Technology History
Aug. 3, 2009

- Pylons system (v1.0) Live - with MySQL
- Huge amounts of our code (as much as 80%) was dedicated to converting UI Objects to and from Database objects. **WTF?**
- No more forks - Pylons & Python run threaded. System resources were significantly less taxed by the presentation stack.

Sluggy.com Technology History
Aug. 14, 2009

- v1.10 Went Live - MySQL replaced by MongoDB (and MongoKit)
- Easy Migration - MongoKit was quickly dropped in place and queries adjusted to new model (Stuck to MySQL schema as much as possible)
- Maintained all bug fixes on MySQL branch for a few weeks “just in case” (never needed it)
- Performance *vastly* improved.
- Over next few months, built tools to use MongoDB in place of **memcached** for caching (**mongodb_beaker**)
- *LAMP* replaced by **LUMP** (Lighttpd, Ubuntu, MongoDB & Python)
- A few things left in **memcached** through a combination of “makes sense there” and indolence (open debt task to migrate much of the rest)

Sluggy.com Technology History
Sept. 2009

- MongoDB completely removed our need for Physical hardware - migrated to Virtual hosting (*slicehost*)
- Average system load is *0.05* on a 2G slice.
- MongoDB uses *1%* of CPU on average.
- Switchover took 2 minutes (ran data conversion script, deployed new code tag, bounced webserver / pylons app)
- No downtime in any way attributable to MongoDB since go live (Can't say the same for MySQL)

Outline

memcached is a crutch. . .

meant to make up for RDBMS' shortfalls

- **memcached** can be great for things you can afford to lose.
- It's not just about what you "can't afford to lose". Beware of cascading failures.
- Over reliance can cause self-DOS after a crash, reboot, accidental flush (even of just one keyset), etc)
- See *Coders at Work* (Siebel) for a great runthrough of what led to memcached's creation.

memcached is a crutch. . .

meant to make up for RDBMS' shortfalls

- **memcached** can be great for things you can afford to lose.
- It's not just about what you "can't afford to lose". Beware of cascading failures.
- Over reliance can cause self-DOS after a crash, reboot, accidental flush (even of just one keyset), etc)
- See *Coders at Work* (Siebel) for a great runthrough of what led to memcached's creation.

memcached is a crutch. . .

meant to make up for RDBMS’ shortfalls

- **memcached** can be great for things you can afford to lose.
- It’s not just about what you “can’t afford to lose”. Beware of cascading failures.
- Over reliance can cause self-DOS after a crash, reboot, accidental flush (even of just one keyset), etc)
- See *Coders at Work* (Siebel) for a great runthrough of what led to memcached’s creation.

memcached is a crutch. . .

meant to make up for RDBMS' shortfalls

- **memcached** can be great for things you can afford to lose.
- It's not just about what you "can't afford to lose". Beware of cascading failures.
- Over reliance can cause self-DOS after a crash, reboot, accidental flush (even of just one keyset), etc)
- See *Coders at Work* (Siebel) for a great runthrough of what led to memcached's creation.

as long as we're being hyperbolic. . .

mongodb is a bionic leg replacement. . .

- MongoDB's MMAP system gives you a "free" MRU cache. Done right and simple; caching on MongoDB is durable, light and fast.
- No piles of special code to track between what is and isn't in your cache and stuff it in there.
- If it falls out of memory cache, it is persisted to disk.
- But. . . Don't build your MongoDB system like a MySQL system (it'll work, but you'll lose speed and flexibility)
  - DBRefs should be used sparingly - favor embedded objects (don't be afraid to denormalize and duplicate data); autorefs can be even worse as there's a performance penalty imposed.
  - Flexible schemas are good.
  - Wasting your time mapping data back and forth between your presentation layer & RDBMS is not just tedious - it's error prone.
  - The more you can put in memory, the less you beat on your disks. Especially important on virtual hosting: Be a Good Neighbor
  - MongoDB is very good at automatically memory caching frequently used data, reducing the amount of code you need to write.

as long as we're being hyperbolic. . .

mongodb is a bionic leg replacement. . .

- MongoDB's MMAP system gives you a "free" MRU cache. Done right and simple; caching on MongoDB is durable, light and fast.
- No piles of special code to track between what is and isn't in your cache and stuff it in there.
- If it falls out of memory cache, it is persisted to disk.
- But. . . Don't build your MongoDB system like a MySQL system (it'll work, but you'll lose speed and flexibility)
  - DBRefs should be used sparingly - favor embedded objects (don't be afraid to denormalize and duplicate data); autorefs can be even worse as there's a performance penalty imposed.
  - Flexible schemas are good.
  - Wasting your time mapping data back and forth between your presentation layer & RDBMS is not just tedious - it's error prone.
  - The more you can put in memory, the less you beat on your disks. Especially important on virtual hosting: Be a Good Neighbor
  - MongoDB is very good at automatically memory caching frequently used data, reducing the amount of code you need to write.

as long as we're being hyperbolic...

mongodb is a bionic leg replacement...

- MongoDB's MMAP system gives you a "free" MRU cache. Done right and simple; caching on MongoDB is durable, light and fast.
- No piles of special code to track between what is and isn't in your cache and stuff it in there.
- If it falls out of memory cache, it is persisted to disk.
- But... Don't build your MongoDB system like a MySQL system (it'll work, but you'll lose speed and flexibility)
  - DBRefs should be used sparingly - favor embedded objects (don't be afraid to denormalize and duplicate data); autorefs can be even worse as there's a performance penalty imposed.
  - Flexible schemas are good.
  - Wasting your time mapping data back and forth between your presentation layer & RDBMS is not just tedious - it's error prone.
  - The more you can put in memory, the less you beat on your disks. Especially important on virtual hosting: Be a Good Neighbor
  - MongoDB is very good at automatically memory caching frequently used data, reducing the amount of code you need to write.

as long as we’re being hyperbolic...

mongodb is a bionic leg replacement...

- MongoDB’s MMAP system gives you a “free” MRU cache. Done right and simple; caching on MongoDB is durable, light and fast.
- No piles of special code to track between what is and isn’t in your cache and stuff it in there.
- If it falls out of memory cache, it is persisted to disk.
- But... Don’t build your MongoDB system like a MySQL system (it’ll work, but you’ll lose speed and flexibility)
    - DBRefs should be used sparingly - favor embedded objects (don’t be afraid to denormalize and duplicate data); autorefs can be even worse as there’s a performance penalty imposed.
    - Flexible schemas are good.
    - Wasting your time mapping data back and forth between your presentation layer & RDBMS is not just tedious - it’s error prone.
    - The more you can put in memory, the less you beat on your disks. Especially important on virtual hosting: Be a Good Neighbor
    - MongoDB is very good at automatically memory caching frequently used data, reducing the amount of code you need to write.

as long as we're being hyperbolic. . .

mongodb is a bionic leg replacement. . .

- MongoDB's MMAP system gives you a "free" MRU cache. Done right and simple; caching on MongoDB is durable, light and fast.
- No piles of special code to track between what is and isn't in your cache and stuff it in there.
- If it falls out of memory cache, it is persisted to disk.
- But. . . Don't build your MongoDB system like a MySQL system (it'll work, but you'll lose speed and flexibility)
  - DBRefs should be used sparingly - favor embedded objects (don't be afraid to denormalize and duplicate data); autorefs can be even worse as there's a performance penalty imposed.
  - Flexible schemas are good.
  - Wasting your time mapping data back and forth between your presentation layer & RDBMS is not just tedious - it's error prone.
  - The more you can put in memory, the less you beat on your disks. Especially important on virtual hosting: Be a Good Neighbor
  - MongoDB is very good at automatically memory caching frequently used data, reducing the amount of code you need to write.

as long as we're being hyperbolic. . .

mongodb is a bionic leg replacement. . .

- MongoDB's MMAP system gives you a "free" MRU cache. Done right and simple; caching on MongoDB is durable, light and fast.
- No piles of special code to track between what is and isn't in your cache and stuff it in there.
- If it falls out of memory cache, it is persisted to disk.
- But. . . Don't build your MongoDB system like a MySQL system (it'll work, but you'll lose speed and flexibility)
  - DBRefs should be used sparingly - favor embedded objects (don't be afraid to denormalize and duplicate data); autorefs can be even worse as there's a performance penalty imposed.
  - Flexible schemas are good.
  - Wasting your time mapping data back and forth between your presentation layer & RDBMS is not just tedious - it's error prone.
  - The more you can put in memory, the less you beat on your disks. Especially important on virtual hosting: Be a Good Neighbor
  - MongoDB is very good at automatically memory caching frequently used data, reducing the amount of code you need to write.

as long as we're being hyperbolic. . .

mongodb is a bionic leg replacement. . .

- MongoDB's MMAP system gives you a "free" MRU cache. Done right and simple; caching on MongoDB is durable, light and fast.
- No piles of special code to track between what is and isn't in your cache and stuff it in there.
- If it falls out of memory cache, it is persisted to disk.
- But. . . Don't build your MongoDB system like a MySQL system (it'll work, but you'll lose speed and flexibility)
    - DBRefs should be used sparingly - favor embedded objects (don't be afraid to denormalize and duplicate data); autorefs can be even worse as there's a performance penalty imposed.
    - Flexible schemas are good.
    - Wasting your time mapping data back and forth between your presentation layer & RDBMS is not just tedious - it's error prone.
    - The more you can put in memory, the less you beat on your disks. Especially important on virtual hosting: Be a Good Neighbor
    - MongoDB is very good at automatically memory caching frequently used data, reducing the amount of code you need to write.

as long as we're being hyperbolic. . .

mongodb is a bionic leg replacement. . .

- MongoDB's MMAP system gives you a "free" MRU cache. Done right and simple; caching on MongoDB is durable, light and fast.
- No piles of special code to track between what is and isn't in your cache and stuff it in there.
- If it falls out of memory cache, it is persisted to disk.
- But. . . Don't build your MongoDB system like a MySQL system (it'll work, but you'll lose speed and flexibility)
  - DBRefs should be used sparingly - favor embedded objects (don't be afraid to denormalize and duplicate data); autorefs can be even worse as there's a performance penalty imposed.
  - Flexible schemas are good.
  - Wasting your time mapping data back and forth between your presentation layer & RDBMS is not just tedious - it's error prone.
  - The more you can put in memory, the less you beat on your disks. Especially important on virtual hosting: Be a Good Neighbor
  - MongoDB is very good at automatically memory caching frequently used data, reducing the amount of code you need to write.

Caveats

- While there is a lot “wrong” with our first pass implementation, MongoDB has been consistent and most importantly: *forgiving*.
- *Someone* has to enforce a consistent schema - if it’s not your datastore (like a RDBMS does) then your code or ops people (or both) have to.

Outline

Open Source Code

mongodb_beaker: Beaker Caching for MongoDB

- Open Source caching plugin for the Python Beaker stack.
- Uses distutils plugin entry points.
- Switching from **memcached** to Beaker + MongoDB required a 2 line config file change:

```
- beaker.session.type = libmemcached
- beaker.session.url = 127.0.0.1:11211
+ beaker.session.type = mongodb
+ beaker.session.url = mongodb://localhost:27017/emergencyPants#
    sessions
```

- Lots of useful options in MongoDB Beaker
- A few limitations on the beaker side which need changes in Beaker (manipulable cache data)

Open Source Code

mongodb_beaker: Beaker Caching for MongoDB

- Open Source caching plugin for the Python Beaker stack.
- Uses distutils plugin entry points.
- Switching from **memcached** to Beaker + MongoDB required a 2 line config file change:

```
- beaker.session.type = libmemcached
- beaker.session.url = 127.0.0.1:11211
+ beaker.session.type = mongodb
+ beaker.session.url = mongodb://localhost:27017/emergencyPants#
    sessions
```

- Lots of useful options in MongoDB Beaker
- A few limitations on the beaker side which need changes in Beaker (manipulable cache data)

Open Source Code
mongodb_beaker: Beaker Caching for MongoDB

- Open Source caching plugin for the Python Beaker stack.
- Uses distutils plugin entry points.
- Switching from **memcached** to Beaker + MongoDB required a 2 line config file change:

```
- beaker.session.type = libmemcached
- beaker.session.url = 127.0.0.1:11211
+ beaker.session.type = mongodb
+ beaker.session.url = mongodb://localhost:27017/emergencyPants#
    sessions
```

- Lots of useful options in MongoDB Beaker
- A few limitations on the beaker side which need changes in Beaker (manipulable cache data)

Open Source Code

MongoKit-Pylons: Pylons patches for Python MongORM

- Added support to MongoKit to run within a Pylons environment (threadlocal setup of connection pool)
- Adding a globally available thread safe connection pool to Pylon was simple. Add 2 lines to config/environment.py:

```
from mongokit.ext.pylons_env import MongoPylonsEnv
MongoPylonsEnv.init_mongo()
```

- Added a few other features to simplify SQLAlchemy migration
- *setattr* / *getattr* support to allow **mongoDoc.*field*** instead of the dict interface (**mongoDoc['*field*']**)
- DB Authentication
- A few missing corners such as additional datatypes, enhanced index definitions on-document, group statement shortcuts, etc.
- Integrated support for autoreferences (which was/are a bad idea)
- Changes merged into MongoKit Trunk (GREAT unit test coverage)

Open Source Code

MongoKit-Pylons: Pylons patches for Python MongORM

- Added support to MongoKit to run within a Pylons environment (threadlocal setup of connection pool)
- Adding a globally available thread safe connection pool to Pylon was simple. Add 2 lines to config/environment.py:

```
from mongokit.ext.pylons_env import MongoPylonsEnv
MongoPylonsEnv.init_mongo()
```

- Added a few other features to simplify SQLAlchemy migration
- *setattr* / *getattr* support to allow **mongoDoc.*field*** instead of the dict interface (**mongoDoc['*field*']**)
- DB Authentication
- A few missing corners such as additional datatypes, enhanced index definitions on-document, group statement shortcuts, etc.
- Integrated support for autoreferences (which was/are a bad idea)
- Changes merged into MongoKit Trunk (GREAT unit test coverage)

Open Source Code
MongoKit-Pylons: Pylons patches for Python MongORM

- Added support to MongoKit to run within a Pylons environment (threadlocal setup of connection pool)
- Adding a globally available thread safe connection pool to Pylon was simple. Add 2 lines to config/environment.py:

```
from mongokit.ext.pylons_env import MongoPylonsEnv
MongoPylonsEnv.init_mongo()
```

- Added a few other features to simplify SQLAlchemy migration
- *setattr* / *getattr* support to allow **mongoDoc.*field*** instead of the dict interface (**mongoDoc['*field*']**)
- DB Authentication
- A few missing corners such as additional datatypes, enhanced index definitions on-document, group statement shortcuts, etc.
- Integrated support for autoreferences (which was/are a bad idea)
- Changes merged into MongoKit Trunk (GREAT unit test coverage)

Open Source Code

MongoKit-Pylons: Pylons patches for Python MongORM

- Added support to MongoKit to run within a Pylons environment (threadlocal setup of connection pool)
- Adding a globally available thread safe connection pool to Pylon was simple. Add 2 lines to config/environment.py:

```
from mongokit.ext.pylons_env import MongoPylonsEnv
MongoPylonsEnv.init_mongo()
```

- Added a few other features to simplify SQLAlchemy migration
- *setattr* / *getattr* support to allow **mongoDoc.*field*** instead of the dict interface (**mongoDoc['*field*']**)
- DB Authentication
- A few missing corners such as additional datatypes, enhanced index definitions on-document, group statement shortcuts, etc.
- Integrated support for autoreferences (which was/are a bad idea)
- Changes merged into MongoKit Trunk (GREAT unit test coverage)

Open Source Code

MongoKit-Pylons: Pylons patches for Python MongORM

- Added support to MongoKit to run within a Pylons environment (threadlocal setup of connection pool)
- Adding a globally available thread safe connection pool to Pylon was simple. Add 2 lines to config/environment.py:

```
from mongokit.ext.pylons_env import MongoPylonsEnv
MongoPylonsEnv.init_mongo()
```

- Added a few other features to simplify SQLAlchemy migration
- *setattr* / *getattr* support to allow **mongoDoc.*field*** instead of the dict interface (**mongoDoc['*field*']**)
- DB Authentication
- A few missing corners such as additional datatypes, enhanced index definitions on-document, group statement shortcuts, etc.
- Integrated support for autoreferences (which was/are a bad idea)
- Changes merged into MongoKit Trunk (GREAT unit test coverage)

Open Source Code

MongoKit-Pylons: Pylons patches for Python MongORM

- Added support to MongoKit to run within a Pylons environment (threadlocal setup of connection pool)
- Adding a globally available thread safe connection pool to Pylon was simple. Add 2 lines to config/environment.py:

```
from mongokit.ext.pylons_env import MongoPylonsEnv
MongoPylonsEnv.init_mongo()
```

- Added a few other features to simplify SQLAlchemy migration
- *setattr* / *getattr* support to allow **mongoDoc.*field*** instead of the dict interface (**mongoDoc['*field*']**)
- DB Authentication
- A few missing corners such as additional datatypes, enhanced index definitions on-document, group statement shortcuts, etc.
- Integrated support for autoreferences (which was/are a bad idea)
- Changes merged into MongoKit Trunk (GREAT unit test coverage)

Open Source Code

MongoKit-Pylons: Pylons patches for Python MongORM

- Added support to MongoKit to run within a Pylons environment (threadlocal setup of connection pool)
- Adding a globally available thread safe connection pool to Pylon was simple. Add 2 lines to config/environment.py:

```
from mongokit.ext.pylons_env import MongoPylonsEnv
MongoPylonsEnv.init_mongo()
```

- Added a few other features to simplify SQLAlchemy migration
- *setattr* / *getattr* support to allow **mongoDoc.*field*** instead of the dict interface (**mongoDoc['*field*']**)
- DB Authentication
- A few missing corners such as additional datatypes, enhanced index definitions on-document, group statement shortcuts, etc.
- Integrated support for autoreferences (which was/are a bad idea)
- Changes merged into MongoKit Trunk (GREAT unit test coverage)

Open Source Code

MongoKit-Pylons: Pylons patches for Python MongORM

- Added support to MongoKit to run within a Pylons environment (threadlocal setup of connection pool)
- Adding a globally available thread safe connection pool to Pylon was simple. Add 2 lines to config/environment.py:

```
from mongokit.ext.pylons_env import MongoPylonsEnv
MongoPylonsEnv.init_mongo()
```

- Added a few other features to simplify SQLAlchemy migration
- *setattr* / *getattr* support to allow **mongoDoc.*field*** instead of the dict interface (**mongoDoc['*field*']**)
- DB Authentication
- A few missing corners such as additional datatypes, enhanced index definitions on-document, group statement shortcuts, etc.
- Integrated support for autoreferences (which was/are a bad idea)
- Changes merged into MongoKit Trunk (GREAT unit test coverage)

Outline

Why MongoDB?

- Dynamic Querying
- Flexibility
- CouchDB's approach appeared obtused and rather unPythonic
- Tools like MongoKit allowed for easy replacement of existing MySQL ORM code with something almost identical
- FAST
- Great Support & Community Available
- MongoDB Mailing List
- IRC: freenode.net #mongodb
- Easy access - talking to developers NOT support staff. One official development company behind MongoDB.
- Shiny

Why MongoDB?

- Dynamic Querying
- Flexibility
- CouchDB's approach appeared obtused and rather unPythonic
- Tools like MongoKit allowed for easy replacement of existing MySQL ORM code with something almost identical
- FAST
- Great Support & Community Available
- MongoDB Mailing List
- IRC: freenode.net #mongodb
- Easy access - talking to developers NOT support staff. One official development company behind MongoDB.
- Shiny

Why MongoDB?

- Dynamic Querying
- Flexibility
- CouchDB's approach appeared obtused and rather unPythonic
- Tools like MongoKit allowed for easy replacement of existing MySQL ORM code with something almost identical
- FAST
- Great Support & Community Available
- MongoDB Mailing List
- IRC: freenode.net #mongodb
- Easy access - talking to developers NOT support staff. One official development company behind MongoDB.
- Shiny

Why MongoDB?

- Dynamic Querying
- Flexibility
- CouchDB's approach appeared obtused and rather unPythonic
- Tools like MongoKit allowed for easy replacement of existing MySQL ORM code with something almost identical
- FAST
- Great Support & Community Available
- MongoDB Mailing List
- IRC: freenode.net #mongodb
- Easy access - talking to developers NOT support staff. One official development company behind MongoDB.
- Shiny

Why MongoDB?

- Dynamic Querying
- Flexibility
- CouchDB's approach appeared obtused and rather unPythonic
- Tools like MongoKit allowed for easy replacement of existing MySQL ORM code with something almost identical
- FAST
- Great Support & Community Available
- MongoDB Mailing List
- IRC: freenode.net #mongodb
- Easy access - talking to developers NOT support staff. One official development company behind MongoDB.
- Shiny

Why MongoDB?

- Dynamic Querying
- Flexibility
- CouchDB's approach appeared obtused and rather unPythonic
- Tools like MongoKit allowed for easy replacement of existing MySQL ORM code with something almost identical
- FAST
- Great Support & Community Available
- MongoDB Mailing List
- IRC: freenode.net #mongodb
- Easy access - talking to developers NOT support staff. One official development company behind MongoDB.
- Shiny

Outline

MySQL Schema Snippets

The Admin Table...

```
CREATE TABLE `admin_users` (
  `id` int(11) unsigned NOT NULL auto_increment,
  `username` varchar(45) collate latin1_general_ci NOT NULL default '',
  `password` char(32) collate latin1_general_ci NOT NULL,
  `display_name` varchar(64) collate latin1_general_ci default NULL,
  `email` varchar(255) collate latin1_general_ci NOT NULL default '',
  `avatar` varchar(255) collate latin1_general_ci default NULL,
  `last_ip` int(10) unsigned default NULL,
  `last_login_date` timestamp NOT NULL default '0000-00-00 00:00:00',
  `disabled` tinyint(1) default '0',
  PRIMARY KEY  (`id`),
  UNIQUE KEY `UNIQUE` (`username`),
  UNIQUE KEY `admin_users_uniq` (`email`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1 COLLATE=
    latin1_general_ci PACK_KEYS=1;
```

MySQL Schema Snippets

The News Table...

```
CREATE TABLE `news` (
   `id` int(11) unsigned NOT NULL auto_increment,
   `author_id` int(11) unsigned NOT NULL,
   `start_date` date NOT NULL,
   `end_date` date default NULL,
   `headline` varchar(255) NOT NULL,
   `story` text NOT NULL,
   `archive` tinyint(1) default '0',
  PRIMARY KEY (`id`),
  KEY `news_author` (`author_id`),
  CONSTRAINT `news_author` FOREIGN KEY (`author_id`) REFERENCES `
       admin_users` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=201 DEFAULT CHARSET=latin1;
```

SQLAlchemy Model

The Admin Object

```
class AdminUser(ORMObject):
    pass

t_admin_users = Table('admin_users', meta.metadata,
    Column('id', Integer, primary_key=True),
    Column('username', Unicode(45), nullable=False, unique=True),
    Column('password', Unicode(32), nullable=False),
    Column('display_name', Unicode(64)), # Should be unique?
    Column('email', Unicode(255), nullable=False, unique=True),
    Column('last_ip', IPV4Address, nullable=True),
    Column('last_login_date', MSTimeStamp, nullable=False),
    Column('avatar', Unicode(255), nullable=True),
    Column('disabled', Boolean, default=False),
    mysql_engine='InnoDB'
)

mapper(AdminUser, t_admin_users)
```

SQLAlchemy Model

The News Object

```
class NewsStory(ORMObject):
    pass

t_news = Table('news', meta.metadata,
    Column('id', Integer, primary_key=True),
    Column('author_id', Integer, ForeignKey('admin_users.id'), nullable
        =False),
    Column('start_date', Date, nullable=False),
    Column('end_date', Date),
    Column('headline', Unicode(255), nullable=False),
    Column('story', Unicode, nullable=False),
    Column('archive', Boolean, default=False),
    mysql_engine='InnoDB'
)

mapper(NewsStory, t_news, properties={
    'author': relation(AdminUser, backref='news_stories')
})
```

SQLAlchemy Model

Paypal: OH THE HORROR

```python
class PaypalIPN(ORMObject):
    pass


t_paypal_ipn = Table('paypal_ipn', meta.metadata,
    Column('id', Integer, primary_key=True),
    Column('first_name', Unicode(64)),
    Column('last_name', Unicode(64)),
    Column('payer_business_name', Unicode(127)),
    Column('payer_email', Unicode(127)),
    Column('payer_id', Unicode(13)),
    Column('payer_status', Unicode(10)),
    Column('residence_country', Unicode(2)),
    Column('business', Unicode(127)),
    Column('receiver_email', Unicode(127)),
    Column('receiver_id', Unicode(13)),
    Column('item_name', Unicode(127)),
    Column('item_number', Unicode(127)),
    Column('quantity', Integer),
    Column('payment_date', Unicode(127)),
    Column('payment_status', Unicode(20)),
    Column('payment_type', Unicode(10)),
    Column('pending_reason', Unicode(20)),
```

MongoKit Model

Admin

```python
class AdminUser(MongoPylonsDocument):
    use_autorefs = True
    use_dot_notation = True
    collection_name = 'admin_users'
    structure = {
        'username': unicode, # unique
        'password': unicode,
        'display_name': unicode,
        'email': unicode,
        'last_ip': unicode,
        'last_login_date': datetime.datetime,
        'avatar': unicode,
        'disabled': bool,
    }

    required_fields = ['username', 'password', 'email']
    default_values = {'disabled': False, 'last_login_date': datetime.
        datetime.now()}

    indexes = [
    {
      'fields': ['username', 'password'],
      'ttl': 86400
    }
```

MongoKit Model

Admin

```
class NewsStory(MongoPylonsDocument):
    use_dot_notation = True
    use_autorefs = True
    collection_name = 'news'
    structure = {
        'author': AdminUser,
        'headline': unicode,
        'story': unicode,
        'start_date': datetime.datetime,
        'end_date': datetime.datetime,
        'archived': bool # default false
    }

    required_fields = ['author', 'headline', 'story', 'start_date']
    default_values = {
      'start_date': TODAY,
      'archived': False,
    }

    indexes = [
    {
        'fields': ['start_date', 'end_date'],
        'ttl': 86400,
```

MongoKit Model
PayPal: OH THE. . . that’s not so bad.

MongoKit Model

PayPal: “Instead”

```
ipn_db = None
try:
    dbh = MongoPylonsEnv.mongo_conn() \
        [MongoPylonsEnv.get_default_db()]['defenders_paypal_ipn']
    ipn_id = dbh.insert(dict(request.POST.items()), safe=True)
    ipn_db = dbh.find_one({'_id': ipn_id})
    if not ipn_db:
        raise Exception('could not lookup, post-insert')
except Exception, e:
    log.exception("Paypal IPN Error: Unable to commit IPN data to "
                  " Database: %s " % repr(e))
    raise DefendersIPNException(repr(e))
```

MySQL -> Mongo Migration

Admin Migration

```
db = mongoModel.AdminUser._get_connection()
conn = db.connection()
conn.drop_database('emergencyPants')

admins = {}
for user in meta.Session.query(AdminUser).all():
    _admin = mongoModel.AdminUser(doc={
        'username': user.username,
        'password': user.password,
        'avatar': user.avatar,
        'disabled': user.disabled,
        'display_name': user.display_name,
        'email': user.email,
        'last_ip': unicode(user.last_ip),
        'last_login_date': user.last_login_date,
    }).save()
    admins[user.id] = _admin

mongoModel.AdminUser.get_collection().ensure_index('password',
    direction=ASCENDING, unique=True)
mongoModel.AdminUser.get_collection().ensure_index(
[('username', ASCENDING),
('password', ASCENDING)])
```

MySQL -> Mongo Migration

News Migration

```
print "Importing News Stories."
for story in meta.Session.query(NewsStory).all():
    #pprint("\tFound a story: %s" % story)
    _story = mongoModel.NewsStory(doc={
        'author': admins[story.author_id],
        'headline': story.headline,
        'story': story.story,
        'start_date': convert_date(story.start_date),
        'end_date': convert_date(story.end_date),
        'archived': story.archive
    }).save()
    #pprint("\t Mongo News Story: %s" % _story)

print "Setting up news story indices."
mongoModel.NewsStory.get_collection().\
    ensure_index('archived', direction=ASCENDING)
mongoModel.NewsStory.get_collection().\
    ensure_index([('start_date', ASCENDING),
  ('end_date', ASCENDING)])
```

Looking for news
In Mongo...

```
def _get_news(date):
    news = NewsStory.all({
        'archived': False,
        'start_date': {'$lte': c._today}
    }).where('this.end_date == null || this.end_date >= new Date()').\
        sort('start_date', -1)

    return news
```

Looking for news
The old MySQL/SQLAlchemy way. . .

```
 def _get_news(date):
     news = meta.Session.query(NewsStory).\
         filter(and_(NewsStory.archive==False, NewsStory.start_date<=date
             ,
                 or_(NewsStory.end_date==None, NewsStory.end_date>date)))\
                 .order_by(NewsStory.start_date.desc()).all()
return news
```

MongoKit
Upcoming Features

- update validation. This feature would allow to validate the update query in order to match the structure. I have to think carefully about it to not miss something.
- migrate to the new gridfs implementation (won't break the API).
- locked field support : field that cannot be changed once set. This is usefull for slug fields or _id.
- fixtures support : ability to generate on the fly documents which values match the structure.
- rdf support : like the "to_json" method but generate a valid rdf document.

Questions?
Contact Info

- Contact Me
  - twitter: **@rit**
  - email: **bwmcadams@gmail.com**
  - bitbucket: `http://hg.evilmonkeylabs.net`
- Pressing Questions?
  - IRC - freenode.net **#mongodb**
  - MongoDB Users List - `http://groups.google.com/group/mongodb-user`
- Mongo Python Language Center - `http://api.mongodb.org/python/1.6%2B/index.html` (Links to Java driver docs, and many of the third party libraries. Pythonic!)
- MongoKit - `http://bitbucket.org/namlook/mongokit/wiki/Home`
- Pylons -`http://pylonshq.com`