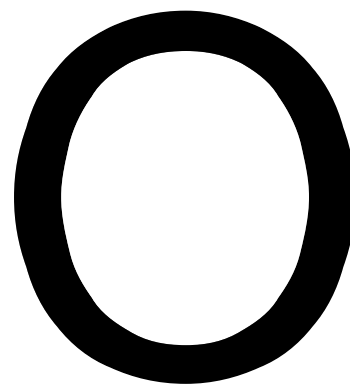
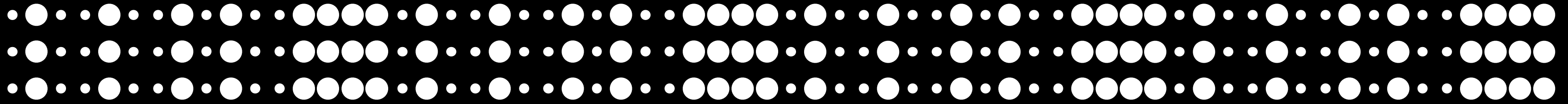
GOOGLE





Working With Google App Engine Models

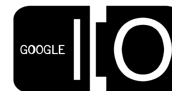
Rafe Kaplan
May 28, 2008

Google App Engine Models

- Introduction
- Relationships
  - One-to-many
  - Many-to-many
- Aggregated properties

Design Goals

- Declaratively describe entities and their properties
- Single location for model description
- Object Oriented
  - Built in encapsulation
  - Extensible entity types and properties



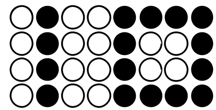
Not A Completely New Concept

Object-relational mapping

- ActiveRecord (Ruby on Rails)
- Django Models
- Hibernate (to a lesser degree)

Not An Object-Relational Mapping

- Does not map to underlying SQL database
- Does not require a pre-existing schema
- Naturally describe things in scalable Python environment
- Information in datastore can be heterogeneous
- Some aspects that seem like drawbacks
  - Does not allow joins
  - Absence of aggregation functions (avg, sum, etc.)
  - No function calls or stored procedures
  - Limited inequality operators



Modeling Relationships

Defining A Real-World Model

Contact database

- What to capture:
  - Personal information (name, date of birth)
  - Phone number
  - Address
- Easy to define a simple Model class to capture this
- Provided GData types help you get started quickly

GOOGLE I/O

First Pass

A naïve model

```
class Contact(db.Model):

  # Basic info.
  name = db.StringProperty()
  birth_day = db.DateProperty()

  # Address info.
  address = db.PostalAddressProperty()

  # Phone info.
  phone_number = db.PhoneNumberProperty()
```

GOOGLE I/O

Problems With This Model

- What if a contact has multiple phone numbers
- What if a contact has multiple addresses
- What if you would like to arbitrarily categorize contacts

GOOGLE I/O

Single Model Solution

Just add another property

```
# Phone info.
phone_number_home = db.PhoneNumberProperty()
phone_number_work = db.PhoneNumberProperty()
phone_number_mobile = db.PhoneNumberProperty()
```

- Can't predict how many phone numbers are required – what if someone invents a phone for a place we never thought of?
- What if someone has more than one mobile phone number?
- Can't easily perform a search across all phone properties

Need Model Relationships

One-to-many

```
class Contact(db.Model):

  # Basic info.
  name = db.StringProperty()
  birth_day = db.DateProperty()

  # Address info.
  address = db.PostalAddressProperty()

  # The original phone_number property has been replaced by
  # an implicitly created property called 'phone_numbers'.

  ...

class PhoneNumber(db.Model):
  contact = db.ReferenceProperty(Contact,
                                 collection_name='phone_numbers')
  phone_type = db.StringProperty(
    choices=('home', 'work', 'fax', 'mobile', 'other'))
  number = db.PhoneNumberProperty()
```

GOOGLE I/O

Working With One-To-Many Relationships

Create

```
scott = Contact(name='Scott')
scott.put()

PhoneNumber(contact=scott,
            phone_type='home',
            number='(650) 555 - 2200').put()

PhoneNumber(contact=scott,
            phone_type='mobile',
            number='(650) 555 - 2201').put()
```

Working With One-To-One Relationships

Read

```
# Accessing the contact from a phone number
print 'The phone number %s belongs to %s' %
      (phone.number, phone.contact.name)

# Getting scotts phone numbers using GQL
all_numbers = PhoneNumber.gql(‘WHERE contact=:1 ’
                              ‘ORDER BY phone_type, number’,
                              scott).fetch(10)


# Traversing from the scott instance
for phone in scott.phone_numbers.order('phone_type')
                                .order('number'):
  print '%s: %s' % (phone.phone_type, phone.number)
```

GOOGLE IO

Working With One-To-One Relationships

Delete

```
# Delete a phone number directly
scott.phone_numbers.filter('phone_type =',
                           'mobile').get().delete()

# One-to-many relationships are not cleaned up automatically.
# Add a method to Contact make it so.
class Contact(db.Model):
  ...

  def delete(self):
    for phone in scott.phone_numbers:
      phone.delete()
    super(Contact, self).delete()
```


GOOGLE I/O

Contact Categories

Many-to-many

- Want to be able to text message a whole category of contacts
- Keep additional information about a category (long description)
- Various types of user defined contacts
  - friends
  - family
  - co-workers
- Users can belong to more than one category

List Of Keys

The preferred way to build two way relationships

- Choose one side to contain list
- This should be the side expected to have fewer members
- The other side defines a virtual property

```
class Contact(db.Model):
  ...

  # Category membership
  categories_contact_is_in = db.ListProperty(db.Key)

class Category(db.Model):

  name = db.StringProperty()
  description = db.TextProperty()

  @property
  def members(self):
    return Contact.gql('WHERE categories_contact_is_in = :1 ',
                       self.key())
```


GOOGLE I/O

Working With Many-To-Many Relationships

Create

```
friends = Category(name='friends')
friends.put()
co_workers = Category(name='co-workers')
co_workers.put()

scott.categories_contact_is_in.append(friends.key())
scott.categories_contact_is_in.append(co_workers.key())
scott.put()
```

Working With Many-To-Many Relationships

Read

```
# Get all categories that Scott is in
categories = db.get(scott.catagories_contact_is_in)

# Use virtual property to get all the members of a group
for member in friends.members.order('name'):
  print member.name, 'is a friend of mine.'
```

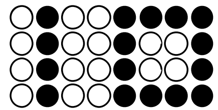
GOOGLE I/O

Working With Many-To-Many Relationships

Delete

```
# Just delete the key from the list
scott.categories_contact_is_in.remove(friends.key())
scott.put()

# Be careful about dangling queries
co_workers.delete()
db.get(scott.categories_contact_is_in) == [None]

# To avoid over-ride Category.delete()
class Category(db.Model):
  ...

  def delete(self):
    for member in self.members:
      member.groups_contact_is_in.remove(self.key())
      member.put()
    super(ContactGroup, self).delete()

# But be wary of large category collections!
```

GOOGLE I/O



Aggregation Properties

Do I Call My Grandmother Enough?

- Grandmother does not remember that I call (or anything else)
- Need to keep track of phone calls
- Count how many phone calls made and their duration
- Get average duration for phone calls
- Remind Grandmother on every call how often I call and for how long

Defining A Call

Add Call class

- Call records date/time and duration of call
- Has a basic one-to-many relationship with Contact

```
class Call(db.Model):
  start = db.DateTimeProperty()
  duration = db.IntegerProperty()
  contact = db.ReferenceProperty(Contact,
                                 collection_name='calls')
```

GOOGLE I/O

How To Do It In SQL

Use aggregation functions

```
SELECT count(*) FROM Calls
 WHERE contact = 19110606

SELECT avg(duration) FROM Calls
 WHERE contact = 19110606

# NOTE: 19110606 is the id for my Grandmother contact
```

How Not To Do It On Google App Engine

Iteration over object elements

- Iteration over all elements in a collection produces dynamic results
- Quickly leads to too much processor use
- Can drain your datastore quota
- Using count() is not efficient either

```
class Contact(db.Model):
  ...

  def average_call_duration(self):
    durations = [call.duration for call in self.calls]
    return sum(durations) / len(durations)

  def call_count(self):
    return self.calls.count()
```

GOOGLE I/O

A Better Way

Maintain aggregate calculations

- Store counts and other computed values on container object
- Encapsulate call insertion within a Contact method

```
class Contact(db.Model):
  ...

  call_count = db.IntegerProperty(default=0)
  average_call_duration = db.FloatProperty(default=0)

  def record_call(self, start, duration):
    call = Call(contact=self, start=start, duration=duration)
    call.put()
    a, c = self.average_call_duration, self.call_count
    a = ((a * c) + duration) / (c + 1)
    self.call_count = c + 1
    self.average_call_duration = a
    self.put()
```

GOOGLE I/O

Thank You!

- Docs: http://code.google.com/appengine/datastore
- Articles: http://code.google.com/appengine/articles
- Group: http://groups.google.com/group/google-appengine

GOOGLE