

Google™





# The Softer Side Of Schemas

Max Ross  
May 28, 2009



# Overview

- The App Engine Datastore
- Soft Schemas
- Migrating to App Engine
- Migrating from App Engine
- Questions





# The App Engine Datastore



# The Datastore Is...

- Transactional
- Natively Partitioned
- Hierarchical
- Schema-less
- Based on Bigtable
- Not a relational database
- Not a SQL engine

“I don’t want an RDBMS for my application, I just want persistence.”



Luiz-Otavio Zorzella, Software Engineer and fellow GBus patron



# Simplifying Storage

- Simplify development of apps
- Simplify management of apps
- App Engine services build on Google's strengths
- Scale *always* matters
  - Request volume
  - Data volume

# Datastore Storage Model

- Basic unit of storage is an Entity consisting of
  - Kind (table)
  - Key (primary key)
  - Entity Group (partition)
  - 0..N typed Properties (columns)

Kind	Person
Entity Group	/Person:Ethel
Key	/Person:Ethel
Age	Int64: 30
Best Friend	Key:/Person:Sally

# Noteworthy Datastore Features

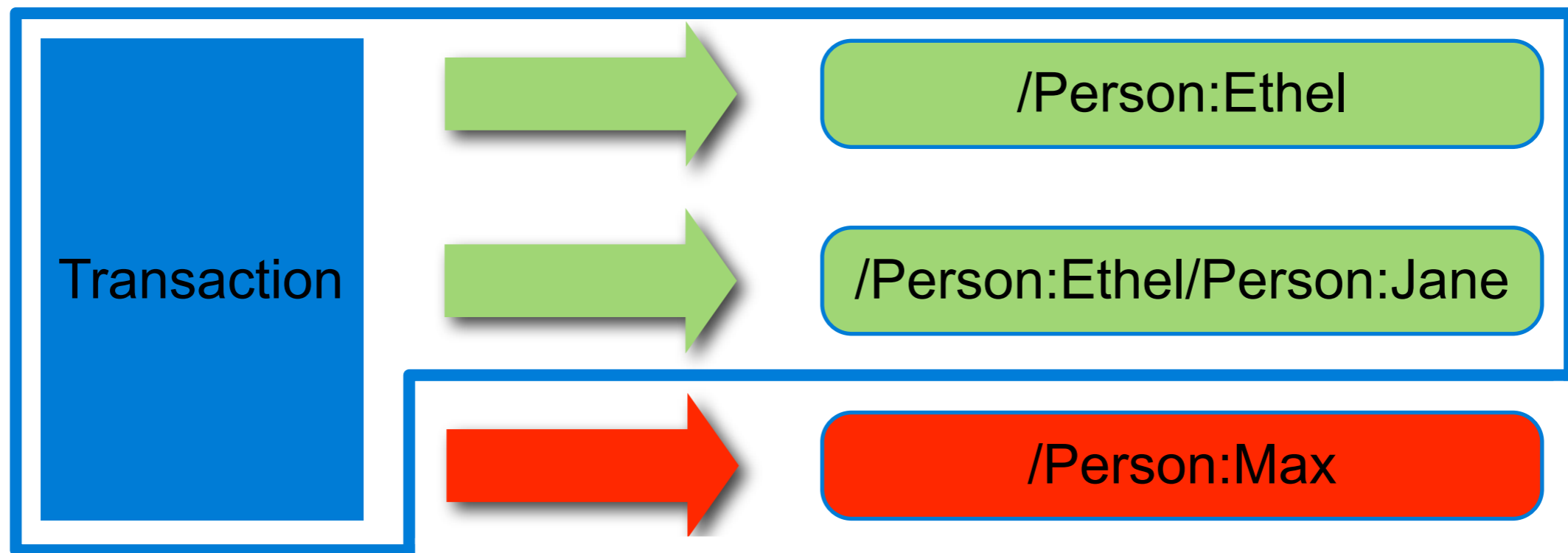
- Ancestor
- Heterogenous property types
- Multi-value properties
- Variable properties

Kind	Person
Entity Group	/Person:Ethel
Key	/Person:Ethel
Age	Int64: 30
Best Friend	Key:/Person:Sally

Kind	Person	
Entity Group	/Person:Ethel	
Key	/Person:Ethel/Person:Jane	
Age	Double: 8.5	
Best Friend	Key:/Person:Eloise	Key:/Person:Patty
Grade	Int64: 3	

# Datastore Transactions

- Transactions apply to a single Entity Group
  - Watch out for contention!
  - Global transactions are feasible
- get(), put(), delete() are transactional
- Queries cannot participate in transactions (yet)







# Soft Schemas



“A soft schema is a schema whose constraints are enforced purely in the application layer.”



# Soft Schema Pluses

- Simpler development process
  - Rapid *typesafe* prototyping
- One less language in your SDLC
  - Online evolution works!



# Implementing A Soft Schema On The Datastore

- JDO or JPA meta-data defines the soft schema
- Established apis
- Existing tooling
- Easier porting
- Specs are (mostly) mappable to datastore features
- Datastore features are (mostly) mappable to specs

# Filtering By Ancestor

- Expose 'parent' on your model object
- Filter on it (equality only)
- Decent substitute for a composite pk

```
@Entity
public class Address {
    // ...
    @Extension(vendorName = "datanucleus", key = "gae.parent-pk")
    private Key personKey;
}

select from com.example.Address where personKey = :personKey
```

# Filtering By Multi-value Properties

```
@PersistenceCapable
public class Person {
    // ...
    @Persistent
    private List<String> hobbies;
}
```

```
select from com.example.Person where hobbies.contains("yoga")
```

# Transactions

- API is a good fit
- Implementation is tougher
- Global vs Entity Group transactions
  - Similar to sharding
- Two phase commit

# Relationship Management

- JDO and JPA are not just about object relationships
  - Transparent persistence
  - Object view of your data
  - Centralized mapping
  - Big maintainability win
- Letting a framework manage relationships can simplify code
  - True for RDBMS
  - Especially true for App Engine Datastore

# Transparent Entity Group Management

- Entity Group layout is important
  - Write throughput
  - Atomicity of updates
- Object relationships can be described as “owned” or “unowned”
- We let ownership imply co-location within an Entity Group

# Owned One To Many (Today)

```
@Entity
class Person {
    // ...
    @OneToMany
    List<Pet> petList;
}
```



Kind	Person
Entity Group	/Person:13
Key	/Person:13

Kind	Pet
Entity Group	/Person:13
Key	/Person:13/Pet:18

# Owned One To Many (Future)

```
@Entity
class Person {
  // ...
  @OneToMany
  List<Pet> petList;
}
```



Kind	Person
Entity Group	/Person:13
Key	/Person:13
Pets	/Person:13/Pet:18


Kind	Pet
Entity Group	/Person:13
Key	/Person:13/Pet:18




# Future JDO/JPA Work

- Provide more control over physical layout
  - Requires getNextId() to avoid multiple updates to same entity
    - Create parent to get parent key
    - Create child with parent key to get child key
    - Update parent with child key
- Support unowned relationships
  - Tricky transaction issues here
- Loosen our query restrictions
  - Parity with Python





# Migrating To App Engine



# Bringing Existing Code To The App Engine Party

- The Datastore is not a drop-in replacement for an RDBMS
- Analyze your use of
  - Primary keys
  - Transactions
  - Queries
  - Views
  - Triggers
- Don't forget about data migration!

# Porting On: Primary Keys

- Single-column numeric and string primary keys fit nicely
- Composite keys can map to an ancestor chain

PET	
PET_ID (pk)	PERSON_ID (pk)(fk)
8	44



Key	/Person:44/Pet:8
-----	------------------

- Mapping tables can be represented using multi-value properties

FRIENDSHIP	
P_ID1 (pk)(fk)	P_ID2 (pk)(fk)
8	32
8	34



Key	/Person:8	
Friends	/Person:32	/Person:34

# Porting On: Transactions

- Identify “roots” in your data model
  - User is often a good choice for online services
- Identify operations that transact on multiple roots
- Analyze the impact of partial success and then either
  - refactor
  - disable the transaction
  - disable the transaction and write compensating logic

# Porting On: Queries

- Shift processing from reads to writes
- Identify joins
  - Denormalize or rewrite as multiple queries

```
select * from PERSON p, ADDRESS a
  where a.person_id = p.id and p.age > 25 and a.country = "US"
```

```
select from com.example.Person where age > 25 and country = "US"
```

- Identify unsupported filter operations (distinct, toUpper)
  - Rewrite as multiple queries
  - Filter in-memory







# Migrating From App Engine



# Taking Your Code To Someone Else's Party

- App Engine persistence code is generally more restrictive
  - Queries
  - Transactions
  - Multiple updates
- Decide what portability means and how important it is
  - To Key or not to Key?
  - Multi-value properties
- Congratulations, you've already sharded your data model!

# Portable Root Object

```
@Entity
class Book {
  @Id
  String id;
  String title;
  // ...
}
```

Kind	Book
Entity Group	/Book:2
Key	/Book:2
Title	Vineland

BOOK	
ID (pk)	TITLE
2	Vineland

# Portable Child Object

```
@Entity
class Chapter {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Extension(vendorName = "datanucleus", key = "gae.encoded-pk")
    String id;

    @Extension(vendorName = "datanucleus", key = "gae.parent-pk")
    Long bookId;

    String pages;
    // ...
}
```

Kind	Chapter
Entity Group	/Book:2
Key	/Book:2/Chapter:8
Pages	23

CHAPTER		
ID (pk)	BOOK_ID (pk)(fk)	PAGES
8	2	23

# Key Takeaways

- The App Engine Datastore simplifies persistence
- You can use JDO/JPA to implement a soft schema
- Denormalization is not a dirty word
- Plan for portability





# Questions



# For More Information

- <http://code.google.com/appengine>
- <http://code.google.com/p/datanucleus-appengine>
- <http://groups.google.com/group/google-appengine-java>
  
- App Engine Chat Time
  - [irc.freenode.net#appengine](irc://freenode.net/#appengine)
  - First and third Wednesday of each month
  
- [maxr@google.com](mailto:maxr@google.com)
  
- To give feedback on this talk: <http://haveasec.com/io>



Google™

