Class 7 Firestore
Elements of Databases
Mar 25, 2022
Instapolls

- Exam 1 feedback
- Firestore setup
The "NoSQL Movement"

- Need for greater scalability
  - Throughput
  - Response time
- More expressive data models and schema flexibility
- Object-relational mismatch
- Preference for open-source software

Source: schema.org
Firestore Overview

+ Distributed system
+ Fully "serverless"
+ Simple APIs for reading and writing
+ Supports ACID transactions (uses Spanner behind the scenes)
+ Designed for mobile, web and IoT apps
+ Implements document model
+ Change data capture for documents
+ Inexpensive

- Only runs on Google Cloud
- Write throughput limits (10K writes/sec)
Firestore’s Document Model

- Firestore document == collection of typed <key, value> pairs
- Primitive types: String, Number, Bool, Timestamp
- Complex types: Array, Map, Geopoint

- Documents Concepts:
  - grouped into collections
  - same type documents can have different schemas
  - assigned unique identifiers (id)
  - store hierarchical data in subcollections
Writing Single Documents

- Every document has a unique identifier of String type
- The `set` method converts a Python dictionary into Firestore document
- A document write must also update any existing indexes on the collection

```python
from google.cloud import firestore
db = firestore.Client()

author = {
    'id': 'sarah.asch',
    'first_name': 'Sarah',
    'last_name': 'Asch',
    'job_title': 'Reporter',
    'seniority': 'L3',
    'hire_date': '2018-01-01',
    'employed_full_time': True,
    'primary_specialty': 'Entertainment',
    'secondary_specialties': ['Business', 'State Government'],
    'articles_to_date': 351,
}

db.collection('authors').document('sarah.asch').set(author)
```
Writing Nested Documents

- Subcollections are nested under documents
- Subcollections can be nested under other subcollections (max depth = 100)

```python
import datetime
from google.cloud import.firestore
db = firestore.Client()

ts = datetime.datetime.now().strftime('%Y-%m-%d-%H-%M-%S')

article = {
    'id': ts,
    'author_names': ['sarah.asch', 'atuma'],
    'author_details': {
        'lead_author': 'sarah.asch',
        'supporting_author': 'atuma'
    },
    'title': 'Why stores say Austin book lovers should shop early for holidays',
    'source': 'Austin 360',
    'section': 'Life',
    'release-date': ts,
    'last-updated': None,
    'num_clicks': 821,
    'contains_photos': True,
    'contains_videos': False
}

db.collection('authors').document('sarah.asch').collection('articles').document(ts).set(article)
```
Writing Multiple Documents

- Write batches of documents in increments of 500 or less
- Batches can contain documents for multiple collections

```python
from google.cloud import firestore

db = firestore.Client()
batch = db.batch()

author = {
    'id': 'sarah.asch',
    'first_name': 'Sarah',
    'last_name': 'Asch',
    'job_title': 'Reporter',
    'seniority': 'L3',
    'hire_date': '2018-01-01',
    'employed_full_time': True,
    'primary_specialty': 'Entertainment',
    'secondary_specialties': ['Business', 'State Government'],
    'articles_to_date': 351,
}

for i in range(399):
    author_ref = db.collection('authors').document('sarah.asch' + str(i))
    batch.set(author_ref, author)

batch.commit()
```
Reading Single Documents

- The `get` method fetches a single document
- The `stream` method fetches all documents in collection or subcollection
- `stream + where` methods filter documents in collection or subcollection
- `order_by` method for sorting results
- `limit` method for limiting number retrieved
- All reads require an index, query will fail if an index does not exist

```python
from google.cloud import firestore

db = firestore.Client()
doc = db.collection('authors').document('sarah.asch').get()

if doc.exists:
    print('Document: ' + str(doc.to_dict()))
else:
    print('No such document')
```
from google.cloud import firestore

db = firestore.Client()
authors_ref = db.collection('authors')
query = authors_ref.where('seniority', '==', 'L3').order_by('last_name').limit(5)
results = query.stream()

for doc in results:
    print('Document: ' + str(doc.to_dict()))
Updates and Deletes

```python
from google.cloud import firestore

db = firestore.Client()
author_ref = db.collection('authors').document('sarah.asch')
author_ref.update({'articles_to_date': firestore.Increment(1), 'seniority': "L4"})
```

```python
from google.cloud import firestore

db = firestore.Client()
author_ref = db.collection('authors').document('sarah.asch')
author_ref.update({'articles_to_date': firestore.DELETE_FIELD})
```

```python
from google.cloud import firestore

db = firestore.Client()
db.collection('authors').document('sarah.asch').delete()
```
College Schema

Convert this relational model to Firestore.

Read access patterns:
1. Get classes by cname
2. Get students and their classes by sid
3. Get instructor and their classes by tid
College Schema

Access patterns:
1. Get classes by cname
2. Get students and their classes by sid
3. Get instructor and their classes by tid

Converted relational model to Firestore.
Design Guidelines for Document Databases

- Identify the expected access patterns against the database.
- For each access pattern, group entities into a hierarchy: top-level and lower-level types.
- Convert each top-level entity into a Firestore collection.
- Convert each lower-level entity into a Firestore subcollection nested in its parent collection.
- Construct a single unique identifier for each document by using the Primary Key column as is or concatenating multiple Primary Key columns.
Exercise: Data Modeling

Convert Shopify schema to Firestore.

**Access patterns:**
1. Get apps by category (Category.title)
2. Get apps with highest review_count
3. Get pricing plan details by app (Apps.id)
4. Get key benefits by app (Apps.id)
Firestore Code Lab

- Clone **snippets** repo
- Open **firestore notebook**
- Create college database in Firestore
- Practice reading and writing CRUD operations
- Answer prompts
Project 5