Class 7 Firestore

Elements of Databases

Oct 15, 2021

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

```
{
  "@context": "https://schema.org",
  "@type": "Restaurant",
  "address": {
    "@type": "PostalAddress",
    "addressLocality": "Sunnyvale",
    "addressRegion": "CA",
    "postalCode": "94086",
    "streetAddress": "1901 Lemur Ave"
  },
  "aggregateRating": {
    "@type": "AggregateRating",
    "ratingValue": "4",
    "reviewCount": "250"
  },
  "name": "GreatFood",
  "openingHours": [
    "Mo-Sa 11:00-14:30",
    "Mo-Th 17:00-21:30",
    "Fr-Sa 17:00-22:00"
  ],
  "priceRange": "$$",
  "servesCuisine": [
    "Middle Eastern",
    "Mediterranean"
  ],
  "telephone": "(408) 714-1489",
  "url": "http://www.greatfood.com"
}
```

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 on Google Cloud
- Write throughput limits (10K writes/sec)

Firestore’s Document Model

- Firestore *document* == collection of typed <key, value> pairs
- Primitive types: String, Int, Float, Bool, Datetime
- Complex types: Array, Map, Geo points

- 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 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

```
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 in batches up to 400 documents
- Batches can contain documents for multiple collections

```
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
- `stream` + `where` methods filter documents in collection
- `order_by` and `limit` methods available
- All document reads require indexes!

```
1  from google.cloud import firestore
2
3  db = firestore.Client()
4  doc = db.collection('authors').document('sarah.asch').get()
5
6  if doc.exists:
7      print('Document: ' + str(doc.to_dict()))
8  else:
9      print('No such document')
```

Reading Multiple Documents

```
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()))
```

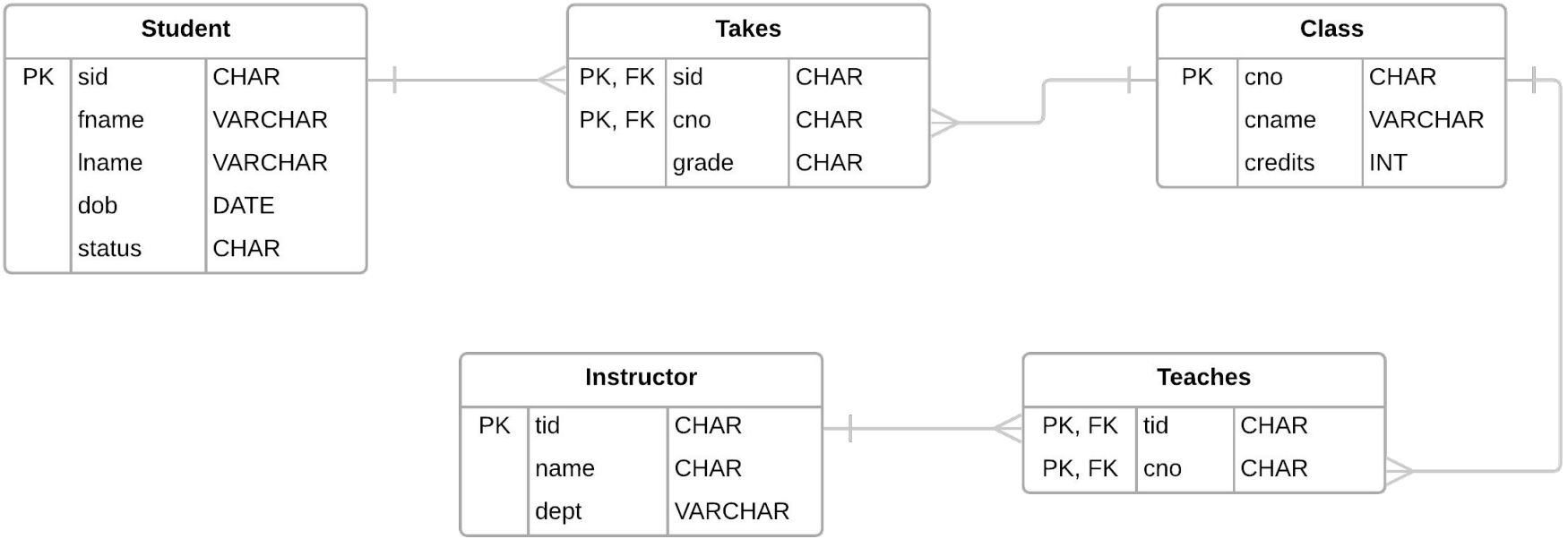
```
from google.cloud import firestore

db = firestore.Client()
authors_ref = db.collection('authors')
query = authors_ref.where('secondary_specialties', 'array_contains', 'Business') \
        .where('articles_to_date', '>', 100)
results = query.stream()

for doc in results:
    print('Document: ' + str(doc.to_dict()))
```

Schema Conversion Example

Access patterns:

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

Normalized college schema for relational systems.

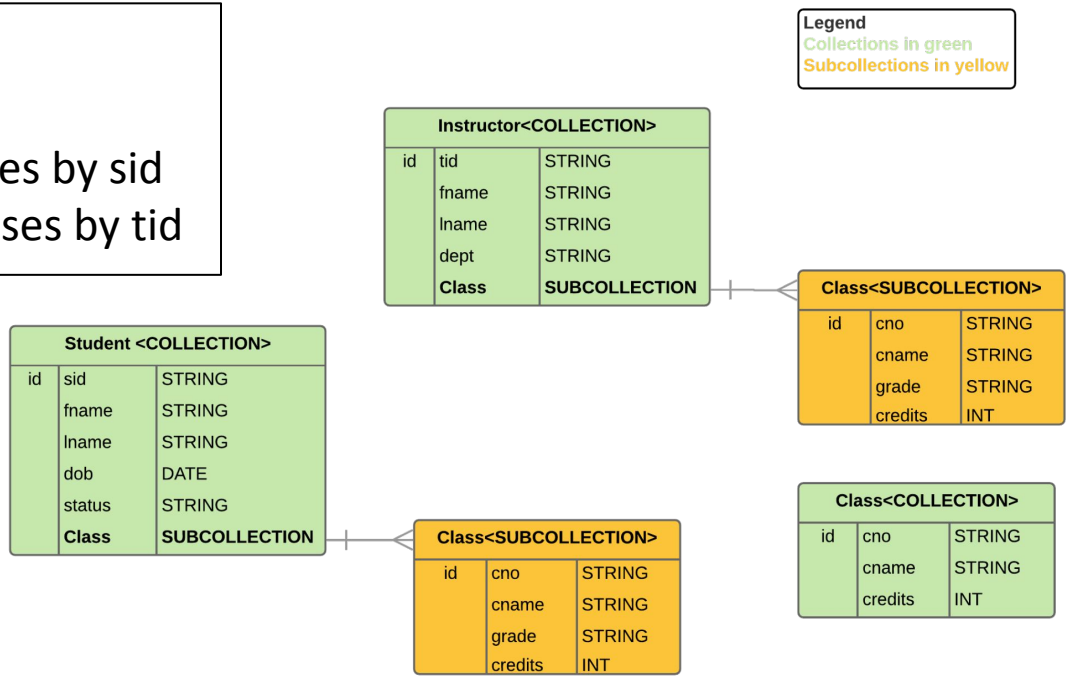


Schema Conversion Example

Converted college schema for Firestore based on access patterns.

Access patterns:

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



Design Guidelines for Document Databases

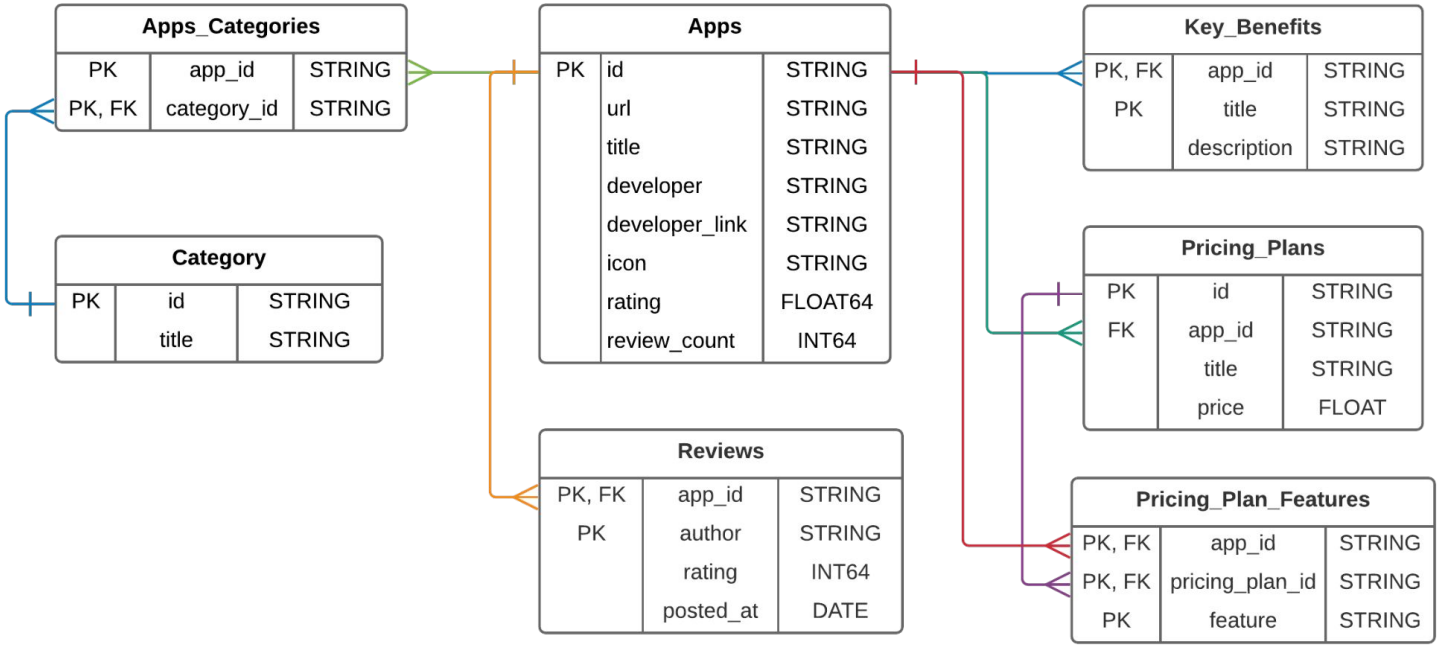
- Identify and analyze access patterns.
- For each access pattern, group entities into a hierarchy: *top-level* and *low-level* types.
- Convert each top-level entity into a Firestore collection.
- Convert each low-level entity into a Firestore subcollection nested in its parent collection.
- Construct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

Practice Problem 1

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)

Convert Shopify schema to Firestore.



Firestore code lab

- Clone snippets repo
- Open firestore notebook
- Create College collections and subcollections
- Explore the data in Firestore

Practice Problem 2

Find all classes taught by Prof. Mitra. Return the cno of those classes.

Project 5

http://www.cs.utexas.edu/~scohen/projects/Project5.pdf