

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

```
{
  "@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 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 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

```
1 from google.cloud import firestore
2 db = firestore.Client()
3
4
5 ▼ author = {
6     'id': 'sarah.asch',
7     'first_name': 'Sarah',
8     'last_name': 'Asch',
9     'job_title': 'Reporter',
10    'seniority': 'L3',
11    'hire_date': '2018-01-01',
12    'employed_full_time': True,
13    'primary_specialty': 'Entertainment',
14    'secondary_specialties': ['Business', 'State Government'],
15    'articles_to_date': 351,
16 ▲ }
17
18 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)

```
1 import datetime
2
3 from google.cloud import firestore
4 db = firestore.Client()
5
6 ts = datetime.datetime.now().strftime('%Y-%m-%d-%H-%M-%S')
7
8 ▼ article = {
9     'id': ts,
10    'author_names': ['sarah.asch', 'atuma'],
11 ▼    'author_details': {
12        'lead_author': 'sarah.asch',
13        'supporting_author': 'atuma'
14 ▲    },
15    'title': 'Why stores say Austin book lovers should shop early for holidays',
16    'source': 'Austin 360',
17    'section': 'Life',
18    'release-date': ts,
19    'last-updated': None,
20    'num_clicks': 821,
21    'contains_photos': True,
22    'contains_videos': False
23 ▲ }
24
25 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

```
1 from google.cloud import firestore
2 db = firestore.Client()
3 batch = db.batch()
4
5 author = {
6     'id': 'sarah.asch',
7     'first_name': 'Sarah',
8     'last_name': 'Asch',
9     'job_title': 'Reporter',
10    'seniority': 'L3',
11    'hire_date': '2018-01-01',
12    'employed_full_time': True,
13    'primary_specialty': 'Entertainment',
14    'secondary_specialties': ['Business', 'State Government'],
15    'articles_to_date': 351,
16 }
17
18 for i in range(399):
19
20     author_ref = db.collection('authors').document('sarah.asch' + str(i))
21     batch.set(author_ref, author)
22
23 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

```
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

```
1 from google.cloud import firestore
2
3 db = firestore.Client()
4 authors_ref = db.collection('authors')
5 query = authors_ref.where('seniority', '==', 'L3').order_by('last_name').limit(5)
6 results = query.stream()
7
8 ▼ for doc in results:
9     print('Document: ' + str(doc.to_dict()))
```

```
1 from google.cloud import firestore
2
3 db = firestore.Client()
4 authors_ref = db.collection('authors')
5 query = authors_ref.where('secondary_specialties', 'array_contains', 'Business') \
6     .where('articles_to_date', '>', 100)
7 results = query.stream()
8
9 ▼ for doc in results:
10     print('Document: ' + str(doc.to_dict()))
```

Updates and Deletes

```
1 from google.cloud import firestore
2
3 db = firestore.Client()
4 author_ref = db.collection('authors').document('sarah.asch')
5 author_ref.update({'articles_to_date': firestore.Increment(1), 'seniority': "L4"})
```

```
1 from google.cloud import firestore
2
3 db = firestore.Client()
4 author_ref = db.collection('authors').document('sarah.asch')
5 author_ref.update({'articles_to_date': firestore.DELETE_FIELD})
```

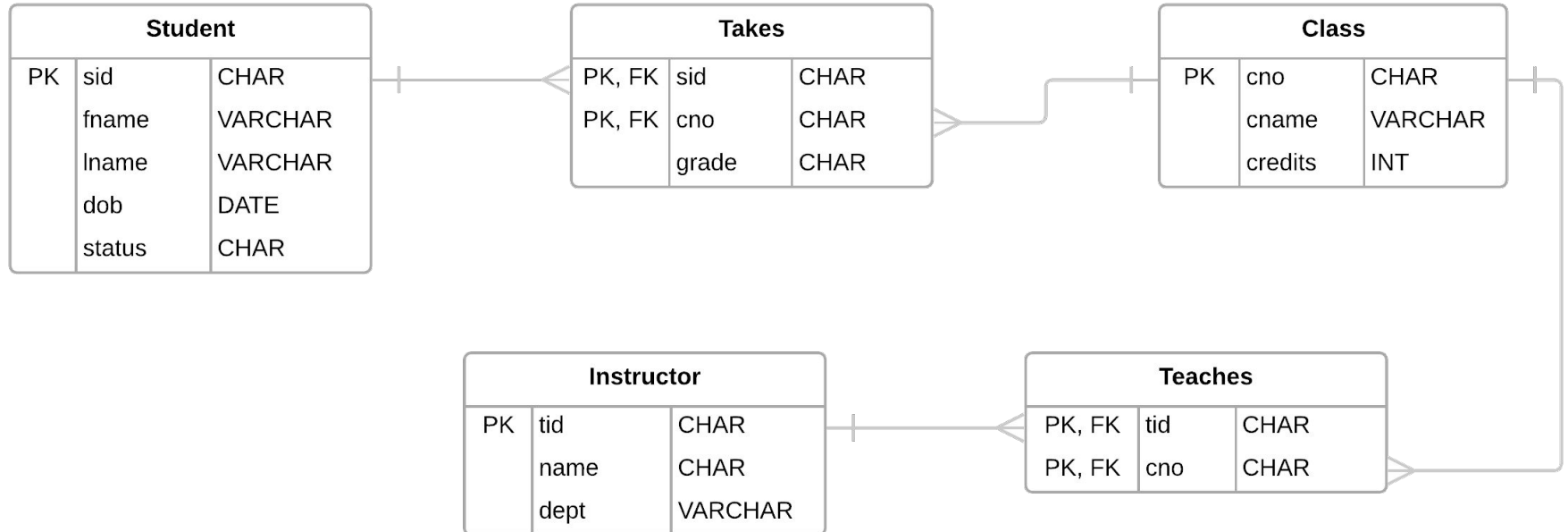
```
1 from google.cloud import firestore
2
3 db = firestore.Client()
4 db.collection('authors').document('sarah.asch').delete()
```

College Schema

Read access patterns:

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

Convert this relational model to Firestore.



College Schema

Converted relational model to Firestore.

Access patterns:

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

Legend
Collections in green
Subcollections in yellow

Student <COLLECTION>		
id	sid	STRING
	fname	STRING
	lname	STRING
	dob	DATE
	status	STRING
	Class	SUBCOLLECTION

Instructor<COLLECTION>		
id	tid	STRING
	fname	STRING
	lname	STRING
	dept	STRING
	Class	SUBCOLLECTION

Class<SUBCOLLECTION>		
id	cno	STRING
	cname	STRING
	grade	STRING
	credits	INT

Class<SUBCOLLECTION>		
id	cno	STRING
	cname	STRING
	grade	STRING
	credits	INT

Class<COLLECTION>		
id	cno	STRING
	cname	STRING
	credits	INT

Design Guidelines for Document Databases

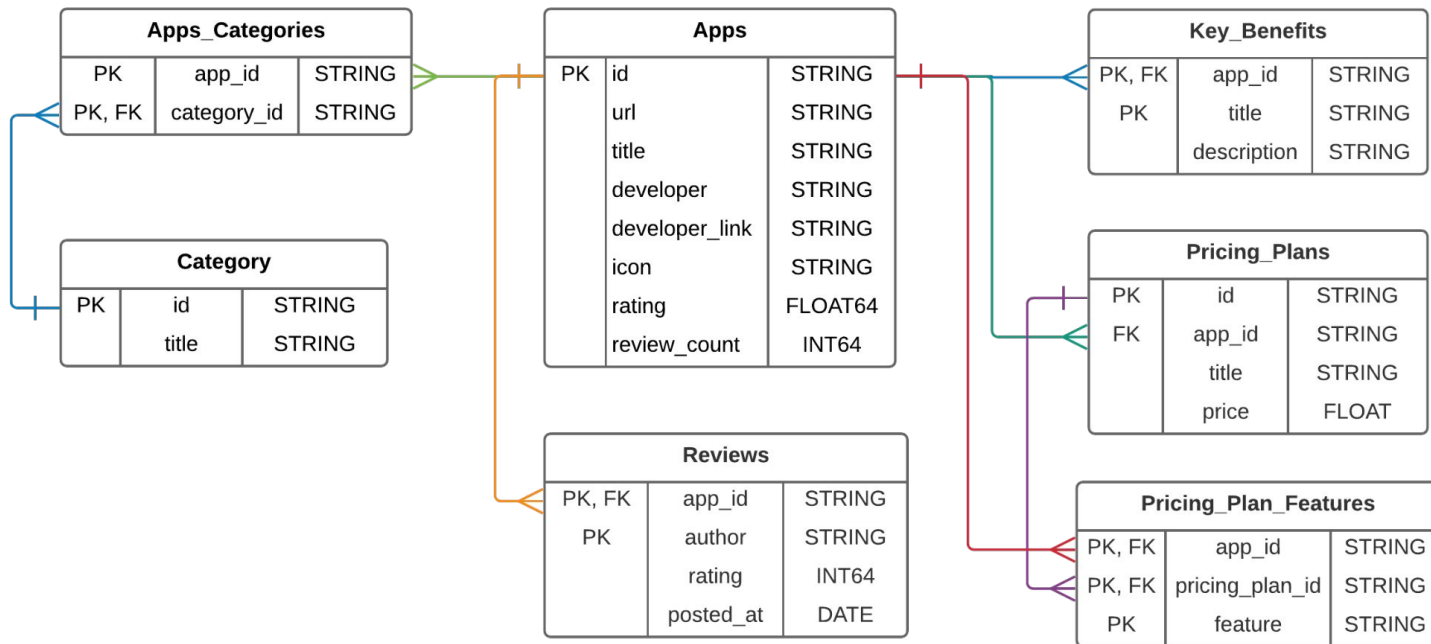
- Identify the expected access patterns against the database.
- For each access pattern, group entities into a hierarchy: *top-level* and *lower-level* types.
- Convert each top-level entity into a Firestore collection.
- Convert each lower-level entity into a Firestore subcollection nested in its parent 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

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