CS 327E Class 5

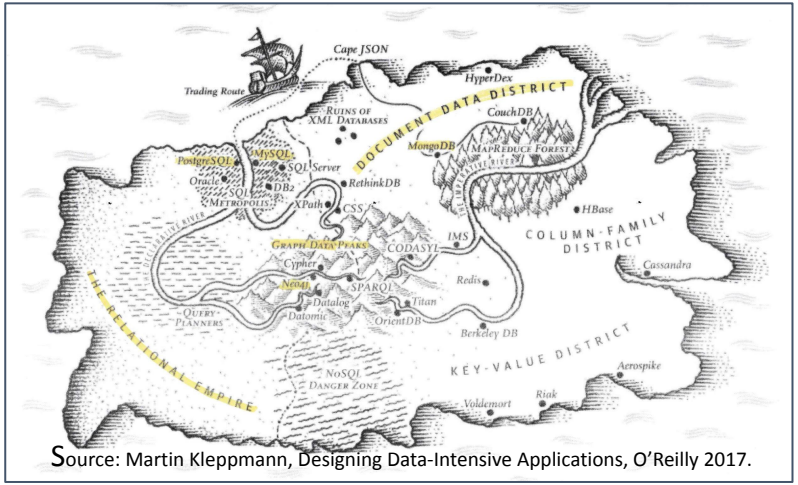
March 5, 2021

Instapolls

- Collect feedback from Test 1
- Verify Firestore set up

Why NoSQL systems?

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



Source: Martin Kleppmann, Designing Data-Intensive Applications, O'Reilly 2017.

The CAP Theorem:
Consistency versus Availability

Why Firestore?

+ Distributed database 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
+ Massive scale (10+M requests/sec, PBs of storage)
+ Cost efficient
- Only available on Google Cloud
- Write throughput limits in native mode (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 are grouped into *collections*
- Documents of the same type can have different schemas
- Documents have unique identifiers (id)
- Documents can store hierarchical data with *subcollections*

Writing to Firestore

- Every document has unique identifier
- Set method converts Python dictionary into Firestore document
- A write must also update indexes on the collection

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

author = {
    'id': 'atuma',
    'name': 'Mary Tuma',
    'title': 'Reporter',
    'section': 'Metro',
    'primary_specialty': 'Business',
    'secondary_specialties': ['State Government', 'City Government'],
    'avg_pub_week': 3.5,
    'tenure_years': 7,
    'full_time': True,
    'emp_start_date': '2014-01-01'
}

db.collection('authors').document('atuma').set(author)
```

Writing to Firestore

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

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

article = {
    'id': '20210301:1030:news',
    'authors': ['atuma', 'skumar'],
    'title': 'How many people have been vaccinated in Travis',
    'section': 'News',
    'pub_date': '2021-03-01',
    'pub_time': '10:30 CST',
    'last_update_time': '12:55 CST',
    'word_length': 2000,
    'clicks': 9000,
    'video_clip': True,
    'video_clip_length': '40s'
}

db.collection('authors').document('atuma').collection('articles').document('20210301:1030:news').set(article)

```

Reading from Firestore

- Get(id) method fetches single document
- Stream method fetches all documents in collection
- Stream + where methods filter documents in collection
- Order by and limit methods available
- All reads require indexes!

```
1  from google.cloud import firestore
2
3  db = firestore.Client()
4  doc = db.collection('authors').document('atuma').get()
5
6  if doc.exists:
7      print(f'Document data: {doc.to_dict()}')
8  else:
9      print(u'No such document!')
10
```

Reading from Firestore

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

authors_ref = db.collection('authors')
query = authors_ref.where('primary_specialty', '==', 'Business').order_by('name').limit(3)
results = query.stream()

for doc in results:
    print(f'{doc.id} => {doc.to_dict()}')

query1 = authors_ref.where('primary_specialty', '==', 'Business')
query2 = query1.where('secondary_specialties', 'array_contains', 'City Government')
query3 = query2.where('tenure_years', '>', 1)

results = query3.stream()

for doc in results:
    print(f'{doc.id} => {doc.to_dict()}')
```

Design Guidelines for Document Databases

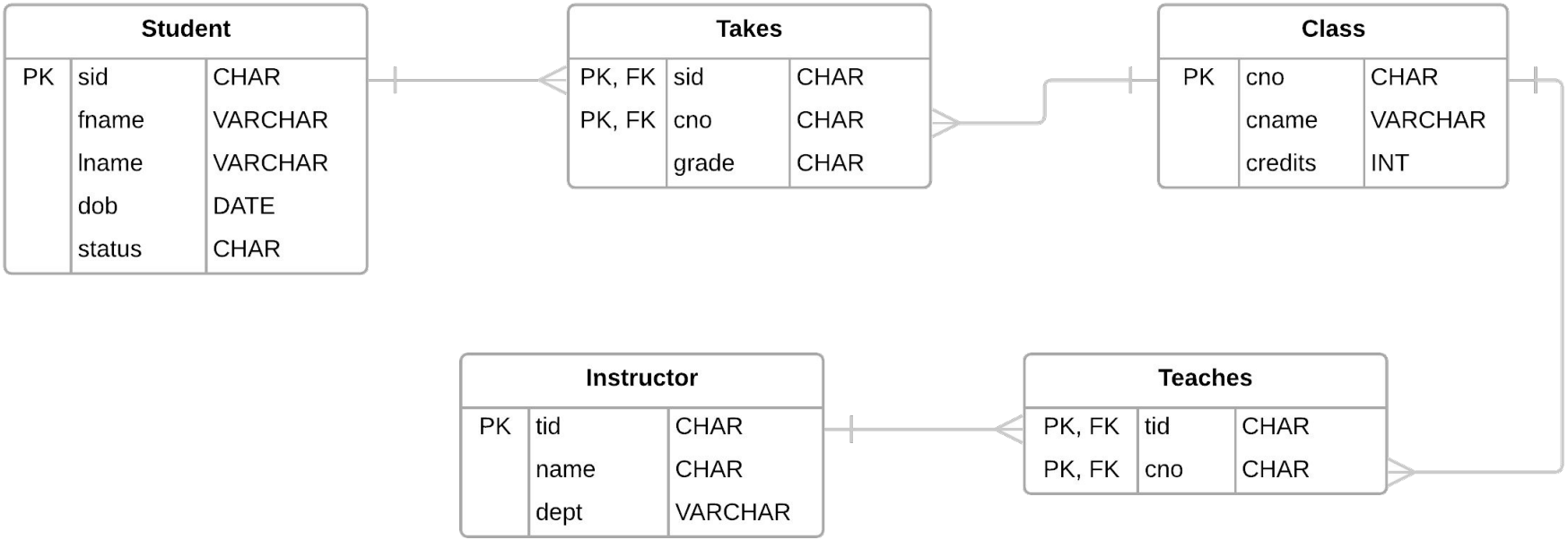
- Analyze your access patterns.
- Group entities into *top-level* and *low-level* types based on your access patterns.
- Convert each top-level entity into a collection of documents.
- Convert each low-level entity into a subcollection of documents (Firestore) / array of subdocuments (Mongo) nested in its parent collection.
- Create a single unique identifier per document. Concatenate composite primary key fields into single identifier if needed.

Schema Conversion Example

Access patterns:

- Get classes by cname
- Get students and their classes by sid
- Get instructor and their classes by tid

Normalized college schema for relational systems.

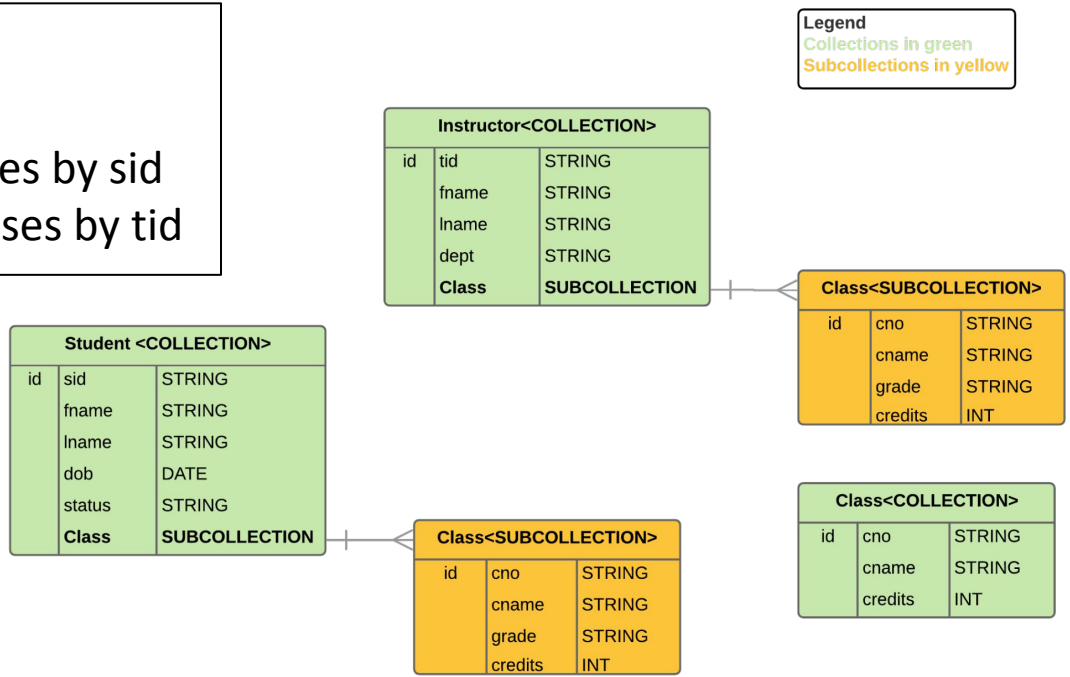


Schema Conversion Example

Converted college schema for Firestore based on access patterns.

Access patterns:

- Get classes by cname
- Get students and their classes by sid
- Get instructor and their classes by tid

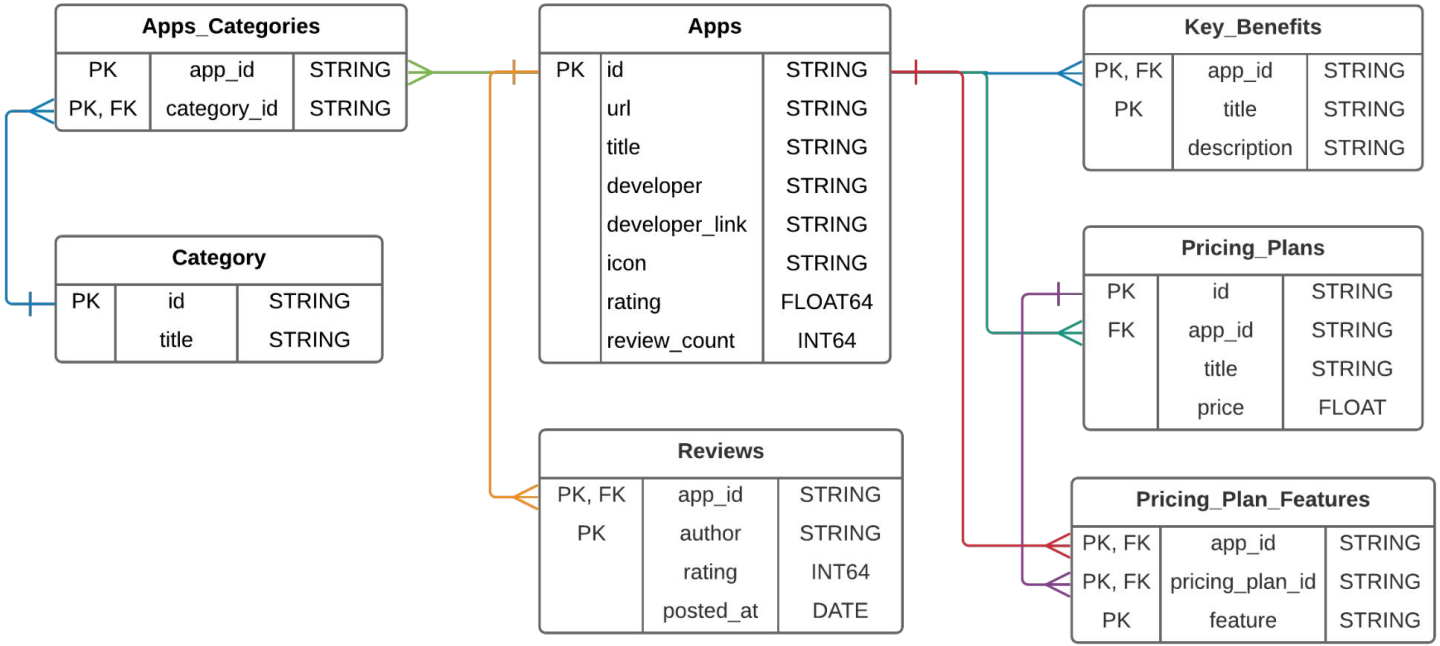


Practice Problem 1

Convert Shopify schema to Firestore.

Access patterns:

- Get apps by category (Category.title)
- Get apps with highest review_count
- Get pricing plan details by app (Apps.id)
- Get key benefits by app (Apps.id)



Firestore Lab

https://github.com/cs327e-spring2021/snippets/wiki/Firestore-Setup-Guide

https://github.com/cs327e-spring2021/snippets/blob/master/firestore.ipynb

Practice Problem 2

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

Project 4

http://www.cs.utexas.edu/~scohen/projects/Project4.pdf