Homework 7 | NoSQL Modeling

Updates made to the assignment after its release are highlighted in red.

Objectives: To gain experience with designing schemas in NoSQL systems, and to contrast them with relational schema design.

Due date: Fri, Dec 5 @ 10pm; **only one late day token permitted**

Median completion time (25sp): **5.5 hours**

Introduction

Congratulations! You've been hired at Flightapp, Silicon Valley's hottest unicorn, which sells tickets to *flights that have already departed*. Your new employers were deeply impressed with your experience with the September 2024 flights dataset and are excited to have their newhire design a schema that will ✨*WOW!*✨ investors and secure their next billion dollars.

To help them choose a schema, they've asked you to write three "one-pagers" describing how to design Flightapp in the following:

- a key-value database (eg, AWS DynamoDB)
- a document store (eg, Asterix)
- *a wide column database (eg, Google Cloud Bigtable) - extra credit!*

Please note that you are only designing the Flightapp schema; you are not implementing anything (unless you want to). In fact, this homework is equivalent to doing M0 3 times! This homework relies heavily on familiarity with the FlightApp spec, so we recommend having it open in a different window as you read through this spec.

Homework Requirements

Data Modeling

For each NoSQL family, we have provided prompts/questions on Gradescope to get you pointed in the right direction; **do them first**. These prompts are not required, *not worth points*, and will not yield a fully-designed schema; they are, however, enough to guide you approximately halfway through the design. For the other half, you will provide **a few paragraphs of high-level description** and then answer a few questions (which *are* worth points) that assess your design's functionality.

Some students benefit from being able to create specific key stores/datasets/tables and test queries on them. For those students, we include a "recommended sandbox"; for students who

do not find sandboxes helpful, you can safely ignore these recommendations. (note: the sample solution is written without the aid of a sandbox).

Authoring (Your First?) Design Document

The phrase "high level description" means that it is not necessary to give table creation statements or even to select a specific database management system. **Do not overthink your design!** While it is tempting to get into the details of transaction handling or high performance writes, the goal of this assignment is to give a brief taste of NoSQL design. The sample solution is only 4 pages long (including rather large screenshots!) and only describes the rows and columns plus how some of the less-intuitive commands are implemented (eg, `book`). That's it. This spec and its associated documentation is longer than the sample solution!

Most students find it helpful to see an example problem; to assist, we have released the full Payroll/Registry/ParkingTickets example here. Please note that this example is able to cover the main functionality in 5 pages (pages 2-7).

Additionally, if you would like to see an example of how to organize a "design document", we have prepared a designdoc for a book catalogue application and a second for Flightapp in a graph database (a topic which we no longer cover).

Functional Specification

The functionality of the flights application remains mostly the same; the greyed-out commands are completely unchanged, whereas **book** and **search** have minor tweaks.

- ***create*** *is unchanged; it creates a new user account with the initial balance. You can assume your Java code for error checking (eg, negative dollar amounts) and password management (eg, salting+hashing) can be reused.*

- ***login*** *is unchanged; it verifies that the user exists in the database and that the password matches. As with* ***create****, you can assume your FlightApp Java code is reusable and focus only on the data which is necessary to implement this function.*

- **search** continues to require that itineraries have numerical IDs which are not stored in the database.

  Unlike FlightApp, however, we **do not limit the number of flights** that can be in an itinerary; your application must be able to support 2-, 3-, or an **arbitrary n-hop itinerary**. You do not need to write Java code for this, but you should be able to describe it to another developer (eg, "do a BFS on the graph starting from X … make sure you track duplicates in a hash table using Y as the key … it's an error when Z …")

- **book** continues to convert a user's (in-memory) itinerary into a (database-backed) reservation without exceeding a flight's maximum capacity. It should continue to verify that the user does not have another reservation on the same day.

However, we no longer concern ourselves with how the reservation ID is assigned (eg, whether it's a hash vs a sequence; whether it starts at 0 or 1; whether it handles rollbacks, etc). You can assume that we will provide a Java library that generates valid-and-unique reservation IDs.

- ***pay*** *is unchanged; it decrements the user's balance and marks a reservation as paid. You can assume that your Java code for error checking and payment can be reused.*

- ***reservations*** *is unchanged; it lists the current user's reservations.*

Note how we have not yet discussed transactions; this is deliberate. Recall that NoSQL databases explicitly trade away consistency for availability. Your solutions **do not need to solve concurrency issues**, but we will ask you to identify what (if any) issues may arise from your design choices by comparing some of the FlightApp m2 "transaction" test cases in our problem set.

Problem Set

Expected Query Patterns

Designing a NoSQL schema requires understanding the pattern of reads/writes to its data. You may make the following assumptions:

- Carrier, Aircraft_Type, and N_Number data will never change.
- Flight information will be updated very rarely (eg, once a week). These updates will happen in bulk; there are no incremental changes to the flight data.
    - For example, we might replace the entire September 2024 dataset with October 2024's data.
- You can assume a moderate number of users (eg, N million, where N is a double-digit number) registered in your system.
- We expect users to be created rarely (eg, a few times a day), and we expect users to login at a moderate frequency (eg, a few times an hour).
- We expect users to book, pay, and view reservations frequently (eg, a few times per minute)
- We expect users to search for itineraries very frequently (eg, hundreds of times a second)

1. Key-Value Data Store

Describe how you would implement the flights application using a key-value data store. Your description should contain enough detail that we understand the key (or keys if you have multiple "types" of keys), any structure that you introduce into the key, and the structure of a key's value.

Recommended Sandbox: If you prefer to instantiate a specific database to test out your ideas or generate screenshots, we recommend AWS's DynamoDB; you can download their NoSQL Workbench for this purpose.

2. Document Store

Describe how you would implement the flights application using a document store database. Your description should contain enough detail that we understand all the types in your document. If your database requires more than one "top-level type", please describe its structure, why it was necessary, and any performance or consistency implications.

Recommended Sandbox: If you prefer to instantiate a specific database to test out your ideas or generate screenshots, we recommend AsterixDB; download their software and start a single-node instance on your machine.

3. [Extra Credit] Wide Column Database

Describe how you would implement the flights application using a wide column database. Your description should contain enough detail that we understand the keyspace (eg, do you have different "types" of keys?), the column families and their contained columns, as well as whether you use explicit timestamps.

Recommended Sandbox: If you prefer to instantiate a specific database to test out your ideas or generate screenshots, we recommend Google Cloud Bigtable. Sign up for a free trial for *all* the Google Cloud products here, then create a "Cloud Bigtable" instance (depending on where you are in the UI, Cloud Bigtable is either a "Cloud and Storage" or a "Storage" product); once you have a Cloud Bigtable instance, create tables/colfamilies/rows using the cbt tool.

Please note that this portion of the assignment is designed to provide feedback to students who attempt wide-column modeling. Do not do this portion "for the points"; they're minimal and are unlikely to change your grade meaningfully for this assignment (much less the course).

4. Reflection

Your reflection should review your writeups for the entire quarter (HW1-HW6 and the two FlightApp writeups) and answer:

1. What~~, if anything,~~ comes to mind as you review your progress throughout the quarter?
    a. Eg, "I'm surprised I ever thought joins were hard" or "I anticipated the NoSQL trend"
2. ~~*Are you now able to answer any of*~~ *Consider one of* the "one question you still have" from previous homeworks? *Are you able to answer it now?* If so: how did you do so? If not, how might you answer it now?

3. Think back to the beginning of the quarter. What is one thing that you would tell yourself to do differently? What is one thing you would do the same?
    a. Eg, "spend more time reviewing SQL aggregates" or "don't stress about RA"; "keep on attending section" or "definitely keep taking notes on my tablet".

As before, your reflection should contain answers to the standard questions. Specifically:

- What is one thing that you *learned* while doing this assignment?
- What is one thing that *surprised you* while doing this assignment?
- What is one *question that you still have* after doing this assignment?

5 and 6. Analytics and Feedback

Lastly, please note that this homework's feedback is **required for this assignment**; it's relatively new to CSE 344, and we appreciate your assistance in assessing whether it should stay and in what form.

There is **optional** feedback for the course's homework series as a whole. This helps us calibrate the required HW7 feedback (described above), though you may choose to skip it if you feel that your HW7 feedback can stand on its own.

Please see the Gradescope assignment for the analytics and feedback questions.

Submission

Please submit your .pdfs containing the text and all supporting documentation (eg, screenshots, code snippets, etc) to Gradescope. Your feedback and reflection answers will also be submitted to Gradescope.

✨🥂✨ *Congratulations!* ✨🥂✨

You've finished the last homework for CSE 344 :)