Joseph Annand

DSE 6210

Project 2

## Part I: Background

In Project 1, the flight\_ms schema was designed based on the SRS document for an airline database. The schema was created using a relational model in PostgreSQL. Figure 1 shows the entity-relationship diagram for the schema

A computer screen shot of a computer

AI-generated content may be incorrect.

Figure : Flight\_ms Schema Crow's foot diagram

The objective of this project is to migrate the database into a non-relational document model. A sample document is created that incorporates all information from the tables in the schema illustrated in Figure 1. An interesting additional objective is to create a sample document representing the itinerary view from Project 1. As a refresher, the SQL query for creating the itinerary view is written below:

---CREATE VIEW FOR CUSTOMER ITINERARY

DROP VIEW IF EXISTS itinerary;

CREATE OR REPLACE VIEW itinerary AS

WITH leg\_schedules AS (

SELECT f.flight\_number

,l.origin\_airport

,l.destination\_airport

,l.actual\_departure\_time

,l.actual\_arrival\_time

,i.reservation\_id

,i.leg\_id

FROM flight\_ms.flight\_schedules AS f

JOIN flight\_ms.legs AS l ON f.flight\_number=l.flight\_number

JOIN flight\_ms.itinerary\_legs AS i ON l.leg\_id=i.leg\_id

), ticket\_details AS (

SELECT r.reservation\_id

,r.passenger\_id

,t.ticket\_type

,c.travel\_class

FROM flight\_ms.itinerary\_reservations AS r

JOIN flight\_ms.ticket\_codes AS t ON r.ticket\_type\_code=t.ticket\_type\_code

JOIN flight\_ms.travel\_classes as c ON r.travel\_class\_code=c.travel\_class\_code

)

SELECT leg\_schedules.reservation\_id,

leg\_schedules.flight\_number,

leg\_schedules.origin\_airport,

leg\_schedules.destination\_airport,

leg\_schedules.actual\_departure\_time,

leg\_schedules.actual\_arrival\_time,

leg\_schedules.leg\_id,

ticket\_details.passenger\_id,

ticket\_details.ticket\_type,

ticket\_details.travel\_class

FROM leg\_schedules

JOIN ticket\_details ON leg\_schedules.reservation\_id=ticket\_details.reservation\_id

WHERE ticket\_details.passenger\_id=202

;

## Part 2: Implementation

### 2.1 JSON Files

#### 2.1.1 Sample Document JSON

{

"\_id": "reservation\_301",

"reservation\_id": 301,

"date\_reservation\_made": "2025-02-01",

"number\_in\_party": 1,

"agent": {

"agent\_id": 1,

"agent\_name": "John Doe Travels",

"agent\_details": "Premium Travel Agency"

},

"passenger": {

"passenger\_id": 201,

"first\_name": "Alice",

"last\_name": "Smith",

"phone\_number": "1234567890",

"email\_address": "alice@example.com",

"address\_lines": "123 Elm St",

"state\_province\_county": "NY",

"country": "USA"

},

"reservation\_status": {

"reservation\_status\_code": 1,

"reservation\_status": "Confirmed"

},

"ticket\_type": {

"ticket\_type\_code": 1,

"ticket\_type": "One-Way"

},

"travel\_class": {

"travel\_class\_code": 1,

"travel\_class": "Economy"

},

"flights": [

{

"flight\_number": 1001,

"airline": {

"airline\_code": 1,

"airline\_name": "Sky Airways"

},

"departure": {

"airport": {

"airport\_code": 101,

"airport\_name": "JFK International",

"airport\_location": "New York"

},

"date\_time": "2025-02-01 08:00:00"

},

"arrival": {

"airport": {

"airport\_code": 102,

"airport\_name": "LAX International",

"airport\_location": "Los Angeles"

},

"date\_time": "2025-02-01 11:00:00"

},

"costs": [

{

"aircraft\_type\_code": 1,

"valid\_from\_date": "2025-02-01",

"valid\_to\_date": "2025-02-05",

"flight\_cost": 300

}

],

"leg":

{

"leg\_id": 501,

"origin\_airport": "JFK International",

"destination\_airport": "LAX International",

"actual\_departure\_time": "2025-02-01 08:00:00",

"actual\_arrival\_time": "2025-02-01 11:00:00"

}

}

],

"payments":

{

"payment\_id": 401,

"payment\_status": {

"payment\_status\_code": 1,

"payment\_status": "Paid"

},

"payment\_date": "2025-02-01",

"payment\_amount": 300

}

}

#### 2.2.2 Itinerary View JSON

{"passenger\_id" : 202, "reservation\_id" : 302, "legs" : [{"flight\_number" : 1001, "origin\_airport" : "JFK International", "destination\_airport" : "LAX International", "actual\_departure\_time" : "2025-02-01T08:00:00", "actual\_arrival\_time" : "2025-02-01T11:00:00", "leg\_id" : 501, "ticket\_type" : "Round-Trip", "travel\_class" : "Business"}, {"flight\_number" : 1002, "origin\_airport" : "LAX International", "destination\_airport" : "O'Hare International", "actual\_departure\_time" : "2025-02-02T09:00:00", "actual\_arrival\_time" : "2025-02-02T12:00:00", "leg\_id" : 502, "ticket\_type" : "Round-Trip", "travel\_class" : "Business"}]}

### 2.3 Documents in MongoDB Compass

#### 2.3.1 Screenshot of Sample Document

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Sample Document in MongoDB Compass

#### 2.3.2 Screenshot of Itinerary View Document

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3: Sample Itinerary Document in MongoDB Compass

## Part 3: Discussion

The sample document in Figure 2 represents a reservation in the airline database. The document contains fields for the document id, the reservation id, the date the reservation was made, and the number of people in the reservation’s party. The latter three fields are attributes in the itinerary\_reservations relation of the flight\_ms schema from Project 1. The remaining fields contain sub-documents which represent the information form related tables referenced by the itinerary\_reservations table. The “agent” field of the sample document contains all the information about the agent represented by agent\_id=1 in the booking\_id table. The sub-document is equivalent to a tuple in the booking\_agents relation. Each field in the sub-document is an attribute in the booking\_agents relation. Since only one booking agent can be associated with a reservation, there is only one sub-document, and the field is not an array of sub-documents. One field that does take multiple sub-documents is the “flights” field. A reservation can include multiple flights, so the “flights” field is an array of sub-documents with each sub-document being a different flight. Each flight has several fields that are also sub-documents. Airline, departure, arrival, costs, and legs all contain sub-documents that represent the tuples of the relations that the flight\_schedules and flight\_costs tables reference.

As an additional endeavor, a sample document for the customer itinerary view created in Project 1 was designed and can be seen in Figure 3. This example document was created using the SQL query in the Appendix, which when run in the SQL shell, copies the customer itinerary information into a JSON file. The itinerary document contains fields for passenger and reservations ids. Additionally, its “legs” attribute is an array of sub-documents. Each sub-document has information for each leg of the customer’s itinerary.

The description of these two documents illustrates how the complexity of the flight\_ms schema can be captured in a non-relational document model. The sample document contains all the information about a reservation from passenger email address to flight arrival times. The advantage of this is that all the information relevant to the reservation that is queried is stored within the document. When the reservation is queried, the only action the query will have to perform is a filter using the find() function in MongoDB to search for either the reservation identification number or the passenger identification number. Conversely, with the relational database model from project 1, to get the same information that is featured in the sample document, the user would have to run a query in SQL that employs multiple JOIN operations between nearly every table in the schema. The same benefit is seen in the sample customer itinerary document as it eliminates the need for the complex query found in the itinerary view from Project 1. In Project 1, the advantage of having a materialized view rather than a virtual table was discussed. The materialized view exists physically in the database like the itinerary document in MongoDB. The materialized view, however, still requires the complex querying when it is created. Also, as the database grows and more customer itineraries are created, storage needs are increasingly stressed. The non-relational model is better suited for scaling horizontally, partitioning documents, and expanding storage.

## Appendix

### SQL Query to JSON

SELECT json\_build\_object(

'passenger\_id', MAX(passenger\_id),

'reservation\_id', MAX(reservation\_id),

'legs', json\_agg(

json\_build\_object(

'flight\_number', flight\_number,

'origin\_airport', origin\_airport,

'destination\_airport', destination\_airport,

'actual\_departure\_time', actual\_departure\_time,

'actual\_arrival\_time', actual\_arrival\_time,

'leg\_id', leg\_id,

'ticket\_type', ticket\_type,

'travel\_class', travel\_class

)

)

) AS itinerary\_json

INTO TEMP TABLE temp\_itinerary\_json

FROM itinerary

WHERE passenger\_id = 202

GROUP BY passenger\_id, reservation\_id;

---must use \copy in SQL shell

\copy temp\_itinerary\_json TO 'C:/Users/janna/Documents/Merrimack MSDS/DSE6210/Project 2/itinerary\_json.json' WITH (FORMAT text, HEADER false);