Homework 5 | Physical Design

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

Objectives: To demonstrate how requirements become conceptual, then schematic, then finally physical designs. To practice taking physical concerns into design consideration

Due date: ~~Thu, May 14~~ *Sat, May 16* @ 9:00pm - *no late day tokens allowed!*

Median completion time: Unknown (this homework is new!). But we aimed for ~4-6h

Introduction

Delta Airlines wants to add a new "last minute deals!" feature. Customers will save their favorite vacation destinations alongside a maximum price they're willing to pay for that destination; then, every time they log in, the system will retrieve and display all flights departing within the next 30 days to any of their saved destinations whose price is under the price cap.

This will be a read-heavy workload that touches multiple tables, so use-case analysis, schema design, and index selection matter significantly. You will design the relations needed to support this feature, reason about normalization, propose and justify indices, and analyze query execution strategies; however, **you are not implementing anything**. Instead, please focus on the correctness of your design and the quality of your reasoning.

Part 0: Existing Design

You may assume that Delta Airlines already has the following base tables and primary keys:

- **Locations**(lid VARCHAR(3) PRIMARY KEY, **-- airport code. Eg, -- "SEA", "YVR"**
  address VARCHAR(64) NOT NULL,
  num_gates INTEGER NOT NULL)
- **Customers**(cid INTEGER PRIMARY KEY,
  full_name VARCHAR(64) NOT NULL,
  email VARCHAR(32) NOT NULL,
  location VARCHAR(3) NOT NULL REFERENCES Locations(lid))
- **Flights**(fid INTEGER PRIMARY KEY,
  origin VARCHAR(3) NOT NULL REFERENCES Locations(lid),
  dest VARCHAR(3) NOT NULL REFERENCES Locations(lid),
  departure_date DATA514DATE(10) NOT NULL, **-- "YYYY-MM-DD", -- eg "2026-05-07". See add'l note.**
  price INTEGER NOT NULL) **-- in dollars**

This assignment builds on top of these tables. Do not redefine or modify them.

We have invented a new fake SQL type for this assignment: **DATA514DATE**. It can be seamlessly converted to and from a string in YYYY-MM-DD format, but it also supports most of the operations you expect from an integer: you can compare them using operators like '=', '<', etc and perform operations such as adding or removing days (eg, `"2026-05-07" + 2 = "2026-05-09"`).

You may assume that Delta Airlines' RDMS will create a **clustered** index on every table's primary key, including any tables you add. If a table's primary key has multiple columns, you may assume that the accompanying primary key index lists its columns in the same order (ie, `CREATE TABLE … PRIMARY KEY (x, y)` means that the accompanying index would save `x` before `y`)

Part 1: Schema Design

Extend Delta Airlines' existing design to support the last-minute-deals feature. Your schema should be normalized and it shouldn't store information already in the given tables.

1.1-1.2: Give an ER Diagram

Present an ER diagram containing the entity sets and relationships that you introduced in order to represent the last-minute-deals feature **and also** the original base tables. Note that a

customer cannot save a destination more than once; if they've already saved `LAX` as a destination, a second attempt to save `LAX` won't change the database's state.

Hint: recall that foreign keys are one of the techniques used to *implement* a relationship; they are not inherent to an entity set. As such, you should not have any FKs in your diagram.

1.3: Define New Table(s)

Convert your **introduced** entity sets and relationships to a set of tables; you may omit `CREATE TABLE` statements for the base tables. For each new table, give its CREATE TABLE statement and a very brief (1-2 sentences per table) of its purpose. If your key is composite (ie, more than one column), please add a comment indicating why the columns are ordered the way they are.

Part 2: Query Formulation

Consider the following query specification:

> *"Find all flights departing in the next 30 days from customer 123's location to every destination they've saved, and whose price is less than customer 123's max price for that location"*

2.1: Write a SQL Query

Using your schema from Part 1, write a single SQL query that retrieves all flights satisfying the condition above. Specifically, your query should:

- filter by this specific customer ID (ie, 123)
- filter flights by departure date (ie, within 30 days of today).
    - You may use the constant `TODAY` to represent, uhhh, today's date
- filter flights by customer 123's location and arrive any of the destinations which the customer has saved
- filter flights by customer 123's max price for that saved location

Your query should return all the columns from `Flights` rows which match; you **do not need** to return any columns from `Customers`, `Locations`, or any of the tables you introduced.

Hint: A subquery and a join are both acceptable; choose the approach which is clearer to you.

2.2-2.3: Identify Which Joins Are Required and Why

Identify every join in your query from 2a. For each join, state the two tables being joined, the join condition (ie, which columns are being equated), and explain in plain english why this join is necessary. Note: it is insufficient to say “to connect the tables”; you must explain what information is being linked.

2.4: Explain the Naive Implementation's Behavior

Again, we assume the database has no indices beyond a **clustered index on each table's primary key**. Next, we introduce 4 constants:

- `L`, the number of unique locations, approximately hundreds (eg, ~300)
- `C`, the number of total customers, approximately middle-millions (eg, ~40 million)
- `F`, the number of Delta-operated flights, approximately low-millions (eg, ~6 million)
- `S`, the number of destinations saved in aggregate by all customers, approximately hundreds-of-thousands (eg, ~500,000)

Which of the existing indices, if any, are used in the execution of the query from 2a? Said in another way: give an expression in `L`, `C`, `F`, `S`, the **constants representing your introduced tables**, and any temporary variables you think you might need – for example, `s123` for the number of destinations saved by customer `123` – that quantifies the number of rows which are read to fulfill your query. Please note that we will accept mild overestimates if you don't want to introduce additional variables; in the previous example, we would accept `S` (instead of `s123`) alongside an explanation that it is an overestimate.

Part 3: Index Design and Use-Case Analysis

You are now tasked with optimizing your query from problem 2 and justifying your optimization design.

3.1-3.3: Discuss Two Indices That Improve Performance

Discuss **at most two indices** which can prevent their underlying tables from being fully scanned during query processing; one or both of them might already exist (ie, they are on the table's primary key). If you introduce a new index to improve performance, please give:

- name of the indexed table
- column(s) being indexed and, if composite, the order in which they are indexed
- its `CREATE INDEX` statement

For each index, trace through how the query engine would use it during execution. For example, does the index speed up a filter predicate vs a join predicate?

Finally, give a **new expression** in `L`, `C`, `F`, `S`, and **your introduced constants** which quantifies the new number of rows which are read when **both indices** are available. As before, you can introduce temporary variables such as `fod` (the number of flights matching on origin and destination) but we will also accept overestimates if documented as such.

3.4: Explain the Read-Write Tradeoff of Adding Indices

Adding indices **can speed up** reads but **always slows down** writes. Why does this tradeoff exist? What additional work must the database perform on every `INSERT`, `UPDATE`, or `DELETE` statement when an index is present?

To answer this question, give a realistic use-case or scenario in which one of **Delta Airlines' existing applications** performs **more slowly** due to your optimization from 3a. Your description should include a qualitative (not necessarily quantitative) description of this use-case's **frequency or the number of affected rows**, as well as what operation(s) cause the slowdown.

***Example**: "if we assume that Delta Airlines does a single "spring cleaning" of old flight data at the end of the year, then* **`12*F`** `DELETE` *statements will perform more slowly because …"*

Hint: what other operations might Delta Airlines do with Flight or Customer data?

3.5: Describe One Use-Case Where an Index Would Not Be Used

Even when an index exists, the query optimizer may choose not to use it. Returning to your **query from 2a**, describe one use-case or scenario where the optimizer would **prefer full table scans** to satisfy that query instead of using either of the indices you discussed in 3a. **Why it would prefer** the full scan?

You **may not change** the values of our constants (ie, you may not change **`L`**, **`C`**, **`F`**, **`S`**, and **your introduced constants**) but you **may change** the assumptions which provided values to your temporary variables (eg, your scenario can assume **`s123`=0** if customer `123` hasn't saved any destinations).

Hint: what happens when a very large or very small fraction of rows match the predicate?

3.6-3.7: Synthesize a Recommendation

At this point, we can (1) quantify the available **speed-up for this specific query** if we implement your optimization and we have (2) at least two use-cases where **your optimization is neutral to negative**.

Give a yes/no recommendation on whether **and how** to implement your optimization immediately upon feature launch **on day 0**, when we have one customer, customer `123`, who has saved <10 destinations; justify your reasoning.

Lastly, assume that this feature is widely popular and that, 6 months after launch, **every customer**, not just customer `123`, checks for last minute deals; furthermore, every customer has saved approximately **`k`** destinations (in other words, **on day 180** we can assume **`S≅C*k`**),

where `k` is a single-digit number. Does your recommendation change? Justify your new recommendation, or justify why your recommendation didn't change.

Analytics and Feedback

This homework is brand new, so the standard analytics and non-standard feedback questions are required; they help us evaluate whether to keep it for next year's DATA 514 offering.

a. Analytics:
    i. How many hours did it take you to finish this assignment?
    ii. How many of those hours did you feel were valuable and/or contributed to your learning?
    iii. Did you collaborate with others on this assignment? If so, please write how many students you collaborated with, and their UW NetIDs. If you did not collaborate with anyone, please enter "N/A".
b. Feedback
    i. Give one specific thing that you liked about this assignment (ie, which you would not change if this assignment is offered next year). You may give "N/A" if you hate every part of the assignment.
    ii. Give one specific thing that you disliked (ie, which you would definitely change). You may give "N/A" in the highly-unlikely scenario where you love every part.
    iii. For our **autograded coding** homeworks (that is: HW2 and HW3) and on a scale of 1-5 (1=hate, 5=love):
        1. How much do you **enjoy** this style of homework?
        2. How much do you **learn from** this style of homework?
    iv. For our **technical homeworks with qualitative components** (HW1 and HW4):
        1. How much do you **enjoy** this style of homework?
        2. How much do you **learn from** this style of homework?
    v. For **this particular homework** (HW5):
        1. How much did you **enjoy** this homework?
        2. How much did you **learn from** this homework?