☕DATA 514 Section 8/9 Worksheet

This doc outlines the running **Employees/Regist example**, as well as how it should be implemented in the various **NoSQL** options we'll be discussing. We will add a new table to the running example, ParkingTickets.

As a reminder, these are the schema and example values. Note that Frances doesn't have a car; Magda has two, and nobody employed at UW owns the Aston Martin.

Employees

UserID	**Name**	**Job**	**ParkingPermit**
123	Leslie	TA	C15
345	Frances	TA	NULL
567	Magda	Prof	E18
789	Quinn	Prof	NULL

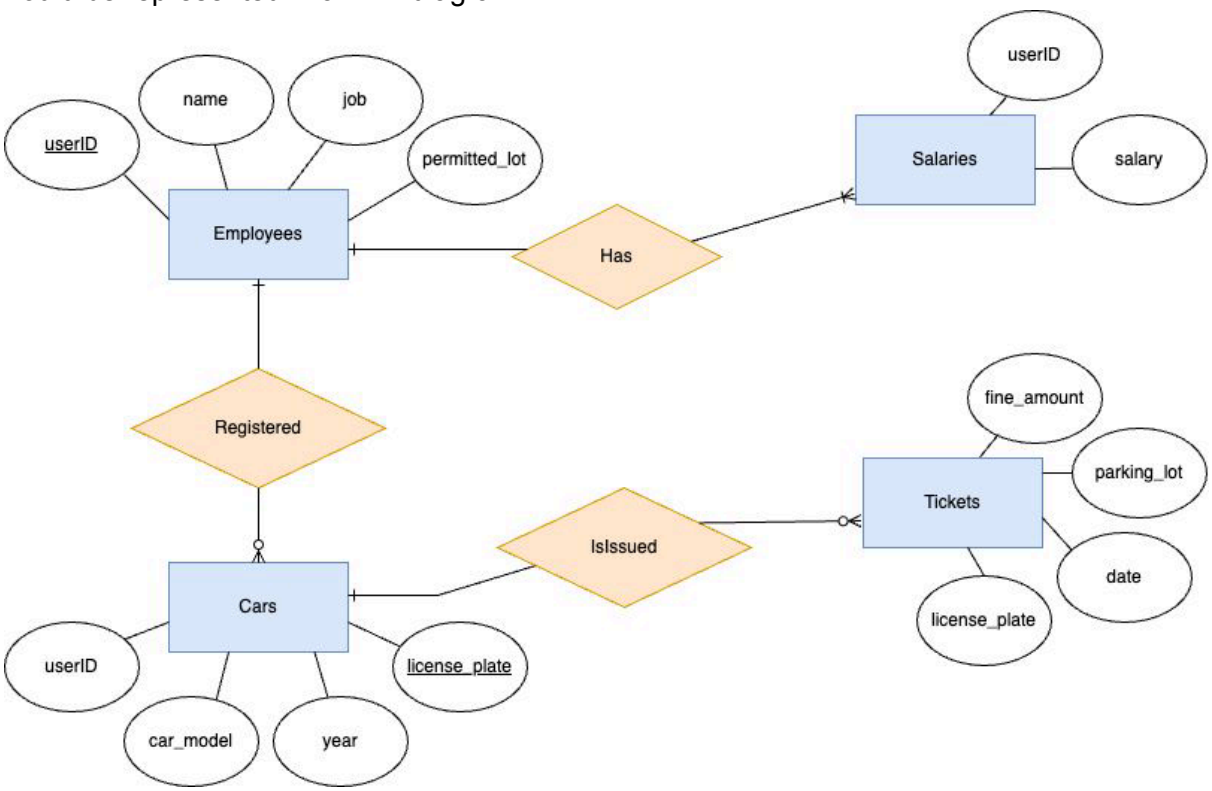
Regist

UserID	**Car**	**LicensePlate**
123	Charger	123 AAA
567	Civic	MMM 1234
567	Ferrari	MMM 5678
789	Picklemobile	PIK 1024
007	Aston Martin	XYZ 0007

ParkingTickets

LicensePlate	**ParkingLot**	**Date**	**Amount**
MMM 1234	C15	2022-11-20	$10
MMM 1234	E01	2022-11-21	$15
PIK 1024	E18	2022-11-22	$10
XYZ 0007	C19	2022-11-01	$10
MMM 1234	E01	2022-11-22	$20

You may also refer to the following ER diagram, which was presented in section. While this diagram includes an additional entity (Salaries) it also accurately represents how the tables would be represented in an ER diagram.



Application / Use-Cases

This is a parking enforcement app which supports the following methods:

1. [infrequent] Listing the permitted parking lot and per-car tickets incurred by each user
    - method sig is `uid -> [ {car1, [{tix1}, {tix2}]}, {car2, []} ]`
2. [frequent] Counting how many tickets a license plate has ever had
    - method sig is `plate -> int` or it can be `plate -> [ {tix1}, {tix2} ]`
    - We use the output of this method to determine the citation amount.
3. [multiple times / sec] Determining whether a plate is allowed to be a specific lot
    - method sig is `(plate, lot) -> true/false` or it can be `plate -> lot`

Because NoSQL design is so intimately tied to its use-cases, there are three decisions which will need to be made for each system:

- **Method 2**: should we store a count of the tickets, or the actual tickets themselves? Note that the latter introduces the possibility of data anomalies (which may be deemed acceptable)
- **Method 3**: What key should we use to track registration and permitting, and how do we represent an existing permit?
- How should we handle **cars without owners** (the Aston Martin) or **employees without cars** (Frances)? Is data loss acceptable or not?

Question 1: Relational Databases

Write a SQL query to complete each of the proposed methods, assuming the provided tables.

1. For a given userid, select the per-car permitted parking lot and tickets issued. (Sort by the license plate.)
2. For a given license plate, return the number of tickets issued.
3. For a given license plate, return the permitted lot

What indexes would you consider for this set up?

Question 2: Key-Value Data Store

Describe how you would implement the Employees/Regist 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.

Question 3: Document Store

Describe how you would implement the Employees/Regist application using a document store database. Your description should contain enough detail that we understand all the types in your document.

Question 4: Wide Column Database

Describe how you would implement the Payroll/Regist 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.

Question 5: Graph Database

Describe how you would implement the Payroll/Regist application using a graph database. Your description should include enough detail that we understand the relationships between the data, and you should provide a sample of how you might execute the suggested queries.