Tarek El-Hajjaoui, Erin Lee, Connor Lydon

Prof. Ali

CPSC-408-01

December 18, 2021

Final Project: Magazine Subscription App

Introduction

In real-world e-commerce applications, users need to be able to create, login into, or edit their account. The IT department, such as admin personnel, need to be able to view and modify database records. This includes performing a lookup on the customers in the database or deleting a record to reflect an account that is no longer active. Data professionals (data engineers, data scientists, and analysts) need to be able to gather data on customers for their analyses. Being able to query from a database allows them to access the latest data that flows in and out of the database as a result of interactions with a user interface, for example. Other applications might seek to actually store the magazines themselves as documents, requiring a document-based NoSQL database. We, however, set out to create a relational database that acts as a payment console application to store data on magazine subscriptions and its customer-related data. Our solution was to create a magazine subscription console application with a command-line interface and a Python/MySQL implementation on the back-end, where the data is pseudo-generated from Python modules to initially populate our database. The goal of the project is to create an efficient, easy-to-use program that is maintainable, scalable, and holds to real-world standards. The two types of users for the application would be: customers and administrators. A customer would interact with our application by initially setting up their account. Every interaction after would include logging into their existing account to manage their magazine subscription. Admin users would be looking to login to manage database records by performing various queries and produce database objects like tables and views to surface certain data to data scientists and analysts.

Schema *(see diagram in Appendix)*

In designing our entity-relationship model, we determined that we would need five distinct entities. That is: magazines, customers, profile, subscription, and payments which contain the information for us to store and organize data of the application. An example of how these entities relate to each other is that a customer would start interacting with our database by setting up a profile to purchase a subscription for a magazine.

To determine cardinality between two tables, we needed to examine whether relationships were one-to-one, many-to-one, one-to-many, or many-to-many. This relationship is decided by evaluating if one record in a table is associated with many records in the other. If one record is associated with only one other occurrence in another entity, then it is one-to-one. If in one direction there is one record associated with multiple records of another entity, then we can say this is one-to-many or many-to-one. If there is a many-to-one and a one-to-many relationship, or a one-to-many relationship going both ways, then it is many-to-many. The tables are laid out in the following order. One customer has one profile and one profile has one customer. One customer has many subscriptions, but one subscription only has one customer. One magazine has many subscriptions, but one subscription only has one magazine. Lastly, a subscription only has one payment and a payment only has one subscription.

Customer relation

After considering what magazine they want from the app, they can create an account or sign in. It will prompt the user to either sign up or login by entering their name ("firstName" and "lastName") as well as login credentials ("username" and "password"). Name and login attributes are all information we would want to store together to form a customer entity. The moment this information is entered we will also want an attribute that describes the date the customer record was created ("recCreateDate").

Profile relation

Additionally, the user would need to enter more details related to location, phone number, and other attributes that would need to be updated or edited. A reason why this is separate is that this information is useful for an analysis of the users; these attributes build a profile about who they are as a customer. All data values in the profile table are auto-generated except for the phone number, where the user is asked to enter this number.

Magazine relation

The magazine name ("magazineName"), the category ("category"), and the cost ("cost") are attributes of a magazine the user would be considering. When looking up magazines, if something was out of print, we would want to display only the ones that are active or remove them from the catalog ("recSatus").

Payment relation

To sign up for a subscription, a customer would need to submit some payment information. Using Python packages, the payment information is pseudo-generated when a customer wants to pay for a new subscription. This means attributes like the card number ("cardNumber"), the card code ("cardCode"), the payment type ("paymentType"), and payment amount. Depending on what magazine they selected, they would have the amount ("paymentAmount") prompted in order to complete the payment.

After processing the transaction, the customer and subscription tables are updated. However, the user is prompted whether they want to undo (rollback) the payment. If they choose to do so, the database will be rolled back to the previous commit state and undo the mutations that occurred in the customer and subscription tables.

Subscription relation

Once payment information has been entered, we will have information built up around a subscription. We have an entire entity describing the subscription in that we have information on the start date ("startDate"), the end date ("endDate"), the number of magazines mailed ("numMagsMailed"), and whether or not the payment was completed ("paymentCompleted").

Solution

As part of the front-end design, the user interacts with the application via the command-line to perform queries by selecting various menu options. After a customer has logged in, they might request a list of all the categories of magazines available from which they could purchase a subscription. We implemented a few statements that select the information a customer might want to see whilst abstracting away the details that should not be exposed, including any id attributes in the underlying base tables. In the process, they also gain a high-level overview of their personal data on record, which they may then need to update, such as their preferred contact method ("contactType"). In the back-end, we used DML statements to perform such updates.

We produced two views to aid the admin user. The view that returns all records from the customer and profile table provides an overview of all the customer-related information in the database. Specifically, this means their credentials, which we pulled from the customer table, and any modifiable account information from the profile table. There is also a view that combines data from the magazine and subscription tables. Doing so allows for the discovery of the number of magazines mailed or the categories of magazines with the most or least subscriptions.

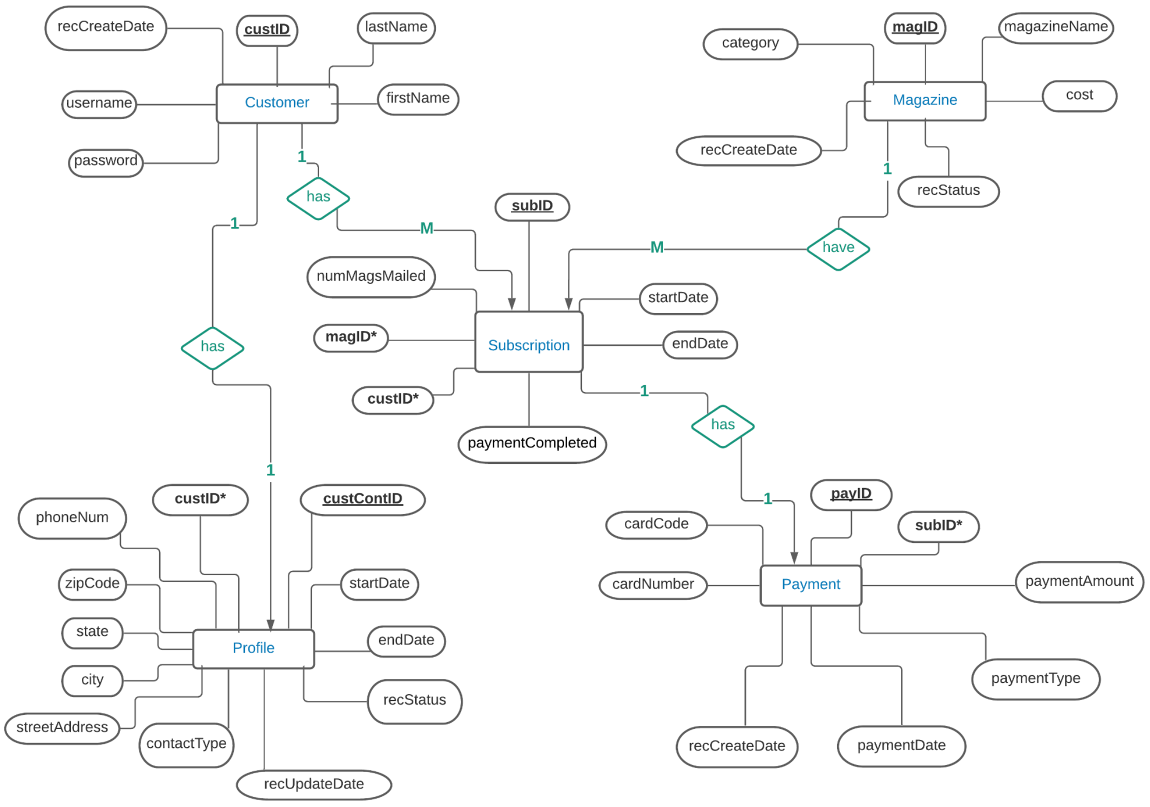
To learn about the customers in our database based on (grouped by) a city was a request we thought we would call upon more than once, giving rise to the creation of a stored procedure. We developed mechanisms that protect the data by means of referential actions and transactions. Reports can also be generated for the admin, who could export this to a CSV.

Results & Conclusion

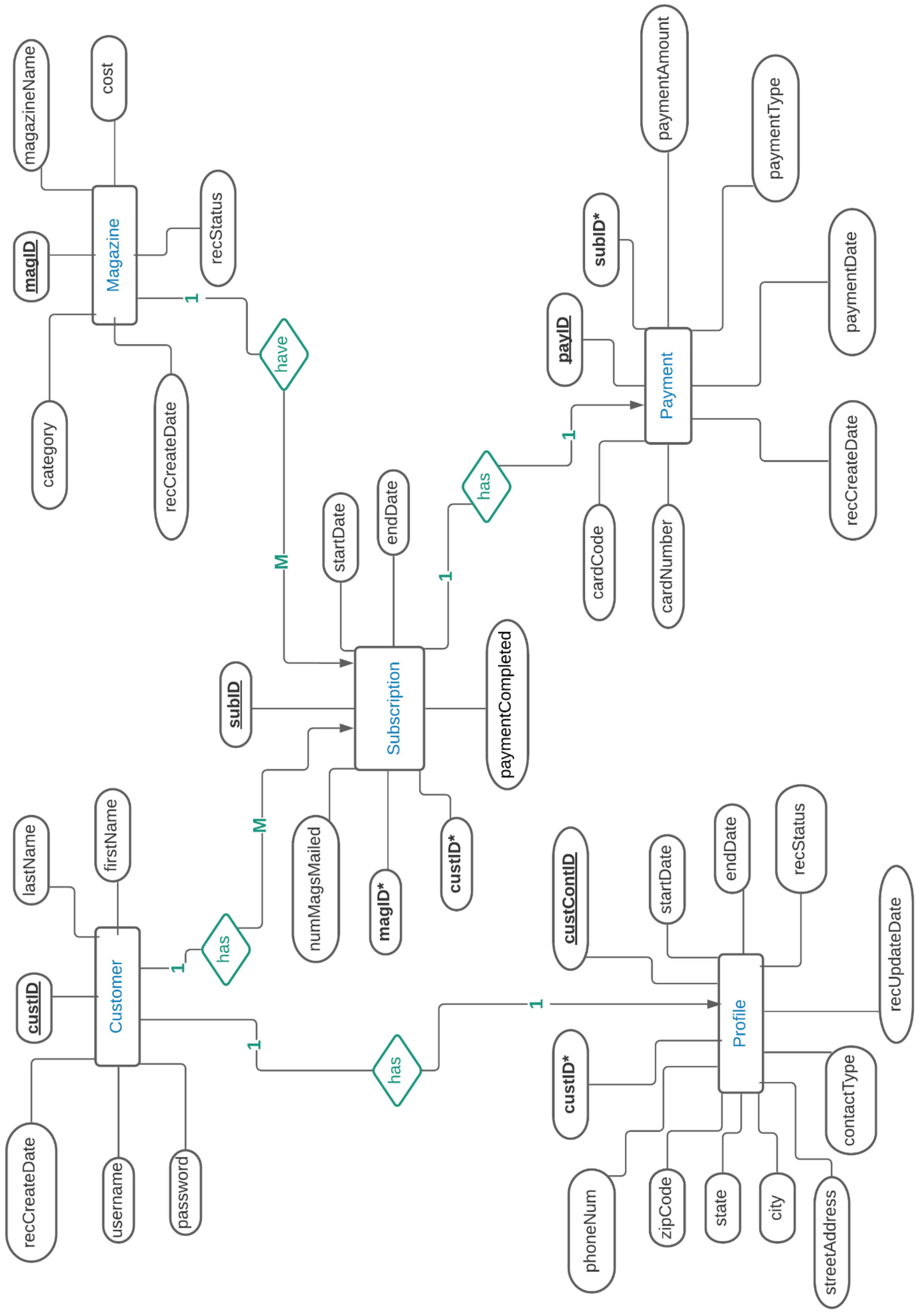
When a user interacts with our application, they will be able to, at a minimum, retrieve information about magazines stored in this database, enter login credentials, explore their profile and update it, and review subscription level details. In addition to these basic actions, we put in place functionality that will run a basic exploratory analysis on the data when logged in as an admin user. This includes a selection query based on the average costs of magazines per category, a selection of the number of subscriptions per city in descending order, most popular category of magazine within a given zip code, and summary statistics on the number of subscription days. The result of this application would allow for the secure storage and retrieval of data for a variety of end users.

Appendix

Landscape view (smaller image):



Portrait view on next page (bigger image)…


Magazine
magID
magazineName
cost
recStatus
category
recCreateDate
have
1
M
Subscription
subID
startDate
endDate
paymentCompleted
numMagsMailed
magID*
custID*
has
1
1
Payment
payID
subID*
paymentAmount
paymentType
paymentDate
recCreateDate
cardCode
cardNumber
Customer
custID
lastName
firstName
recCreateDate
username
password
has
1
M
has
1
1
Profile
custContID
startDate
endDate
recStatus
recUpdateDate
custID*
contactType
phoneNum
zipCode
state
city
streetAddress