Hello, I am Jasminder or Jas as I let my group members call me. My individual work for the group project is committed to GitHub under the username jaga41. I will also review my contributions here.

The first to-do was to review and critique everyone else’s models. I posted this on GitHub Issue #10. My comments to Amber were, “The verb phrases between business objects/entities that you added is good for understanding the relationships between everything. I like the segregation between the payment types, but I’m not sure that that relationship between Customer and Check, Credit, EPay should all be mandatory/not null; a Customer might not necessarily have all three types. Like Payment type, the rest of your model is normalized well, although there might be a few more dependencies that can be removed or even added back. For example, maybe Condition and Format can be combined under a Detail, but I respect your decision to have them as separate tables. The PascalCase makes everything easy to read and your data types are selected well. I think the traditional movie model is strong here, so it can merge well with something that has more of an emphasis on streaming.” I took a similar tone when writing my comments to Adrian, in short: “I like the originality of merchandise in a store, your licensing is like my subscription model so we should collaborate, but maybe some aspects go out of our scope.” So, I constructively criticized group members by highlighting their model’s strengths while also pointing out what I thought could be improved upon.

I also made constructive suggestions to group members’ models in later stages, like on Wade’s PDM under GitHub Issue #38 on October 7th. I wrote, “Thanks for your contribution, Wade. In my model, I also added Gift Card Balance as an attribute to Gift Card. Another small thing is the naming conventions: I use "Id" instead of "ID", but we can see what everyone else also does; I also used Wallet (binary data type) for Bitcoin instead of AccountNumber, but it's the same concept so we can decide as a group. Thanks.”

A few minutes after putting these two posts up, my GitHub account, user gaja9141, was flagged. As a result, my posts were not visible to my group members. I submitted a ticket to GitHub because I believed this was a mistake. The email verifying my ticket said that it may take as long as a week to review; so, in the meantime, I made jaga41 and did all my project work there, ensuring that it would be visible to other group members for collaboration. A GitHub moderator named Sean unflagged my other account on October 7th, after about 6 days, saying it was a false positive for spam. But by then, I had already done most work for the group project on jaga41 and continued to do. This was a new and unique experience to me; therefore, I thought it would be prudent to share. Learning from this, I would advise future group members to not post comments consecutively on GitHub issues, especially if their account is relatively new. Instead, post one long comment and format sections within it, which is I how I submitted all my to-dos without issue.

Likewise, I responded to critiques other group members had about my model. One member, Adrian, had opinions on my work so I had a voice meeting with him on October 2nd. I explained how I thought my model was valid, but he thought it was more representative of a hub as I accounted for people having subscriptions to multiple streaming services. So, I tried to reorient my model to match the perspective from within a single streaming service. For example, I dissolved the many-to-many relationship between Customer and Subscription. So instead, a Customer has a single Streaming Service Subscription. I also normalized SubscriptionTier into its own separate entity because I saw that those characteristics, cost, devicelimit, and resolution were really describing it. And having it separate makes it simpler to add more features in the future. So, that Streaming Service Subscription has a single Tier, which is made by the Streaming Service; the Streaming Service itself can provide multiple Tiers. I made these logical modifications myself then sent it back to the group for peer review.

I was also assigned to write the definitions and attributes for the Person and Address entities, which are posted on GitHub Issue #29. Subsequently, I created the conceptual, logical, and physical data models for these on GitHub Issue #35.

The definition of the business object Person is a single human, and an Address is where a Person lives. The business rule between these two objects is that A Person has an Address, and an Address belongs to a Person.

The attributes of the Address Entity are Address Id a surrogate key, State, City, Zip Code, and Street all prefixed with Address. Unlike in eMovies, where these attributes were in multiple tables, I normalized them into a separate entity. The advantage of this is that when an Address is updated, only this single relevant table is touched. The attributes of the Person entity are Person Id (a surrogate key), Date of Birth, Gender Pronouns, First Name, Last Name, Phone Number, Email, and a foreign key referencing the primary key of the Address entity. Person and Address share a non-identifying relationship. These are the identifiable information of Person, necessary to keep record of a subscription and its payment.

The Address table: Column AddressId is a surrogate key of type integer; when I merge this with the group model, it will belong to the SurrogateKeyInt domain. State is char(2) because I want input to be state abbreviations which are two fixed characters, such as NY for New York. Therefore, I created a domain named StateAbbreviation and set it as the domain parent of the AddressState column.

The Address table: Column AddressId is a surrogate key of type integer; when I merge this with the group model, it will belong to the SurrogateKeyInt domain. State is char(2) because I want input to be state abbreviations which are two fixed characters, such as NY for New York. To achieve this, I created a domain named StateAbbreviation.

The StateAbbreviation domain is a child of the String domain. I tied a validation rule to it, a check constraint with the valid values being only the two letter abbreviations of all 50 US states. I consulted an official government website, the FAA, to ensure the accuracy of these abbreviations and even added the full state name as their display value.

Similarly, AddressZipCode is a char(5) because a zip code is 5 fixed characters, like 11367. So, I made a domain named ZipCode and set it as the Domain Parent of this column.

The ZipCode domain is a child of the String domain. In America, the lowest zip code is 00501 and the highest is 99950. I referenced these business rules once again from a government authority, the US Postal Service. Because of this, I added a check constraint, a validation rule of type MinMax to check if a zipcode is between those two string values.

City and Street are varchars because city and street names vary. All the columns in Address are not null because I deem them non-optional. Since I require them in my model, for billing, they form an alternate key together because their combination uniquely identifies an Address.

The Person table: Column PersonId is a surrogate key of type integer. First Name, Last Name, Email, and Phone Number are varchars with different max lengths because these fields vary. But, a phone number, international or not, can only have a maximum of 15 digits. All columns are not null because I need this information to record payments. For the same reason, AddressId is not null. Address and Person have a non-identifying relationship, meaning technically a Person can exist without an address, but for our purposes, we require an address to bill a subscription. And although Date of Birth is a datetime, we do not need the minutes and seconds because usually that precision is not shared except on the birth certificate. So, I created a Domain named DateOfBirth and made it this column’s parent domain.

The domain DateOfBirth is a child domain of Datetime. I add an expression to its which formats the datetime to month slash day slash year, which is the convention in America.

To be more inclusive of the diversity of genders in the LGBTQ community, I renamed the traditional Gender column in Person to GenderPronoun. A “gender pronoun" is the pronoun that a person chooses to use for themselves; there are many depending on how that person identifies. So, I create a domain GenderPronoun and made it this column’s parent domain.

The GenderPronoun domain is a child of the String domain. Its valid values include “He slash Him slash His”, “She slash Her slash Hers”, and more for the transgender and Gender Non-Conforming community. I even included “Other” in case the list missed a gender pronoun and manual input is desired. When compiling these values, I consulted an article from New York City Department of Social services, which is a reputable government agency. Lastly, in my Person table, further tightens I could have included are regular expressions to validate the format of email and phone number strings.

Those were my contributions to the project. I believe that I did my job as a group member by attending all the meetings, completing all the to-do’s on time (as set by our great project leader Amber), and contributing constructive comments when critiquing other group member’s work. I also worked on ideas for SubscriptionTier and the Subscription model, which were a springboard, and individually on the Person and Address tables. Similarly, I introduced the idea of Bitcoin (which later become Cryptocurrency) to Payments. I wrote #1-3 in the Project Plan Specifications and finally made tweaks such as fixing naming conventions in the final group model. Thank you.