Jackson Keyser

CMSC 461

Professor Kalpakis

May 9, 2024

Final Report

Introduction

Pluto Reality Inc. is interested in developing a database application to help it in managing the rental of real estate properties across the country. As this is a cumbersome task, they are looking for a company to build a product that will be accessible to the employees of Pluto, assisting them in managing their vast quantities of data. The software is expected to be usable for people that have little to no technical background and to retrieve results fast. As both ease of use and speed are crucial to their success, it is important to build a database that fulfills these requests.

Executive Summary

1.1 Phase B

Pluto Reality Inc. is a company interested in developing a database application to assist them in managing their rental real estate properties. The goal of this first phase it to create a set of entities and their relations which can store and retrieve data efficiently and accurately. To achieve this goal, I first created an entity set for employees, with 2 child classes, partner and associate. I also created an entity set for rental properties, property viewings, property owners, and leases. Next, I would create foreign keys to prevent orphaned records from being stored, slowing down

the database. Following, I would create the cardinality constraints, such as an employee not being allowed to be both a partner and associate. Lastly, I included some extra entity constraints such as all associates may have a max of twelve properties and a lease duration must be between three and thirty-six months. Overall, the entity-relationship model tries to accomplish all goals while being as simple and easy to follow as possible.

1.2 Phase C

Phase C of the project involved mapping the conceptual design onto the relational data model. The client, rental property, and property owner tables were left untouched. Here, the employee table is split into associate and partner tables. Next, I combined the lease table and its connecting relationships into a single table with client_id, owner_id, partner_id, and rental_id foreign keys. I did the same for the property viewing table, creating foreign keys for rental_id, associate_id, and client_id. I would go on to create tables for owned by and managed by, to connect properties to their owners and managers. Finally, I gave emails their own table, with client_id, owner_id, associate_id, and partner_id foreign keys. The foreign key that the email is associated with is set, the other three foreign keys are set to NULL. Overall, both models were very similar, and small changes were made when moving from conceptual to relational.

1.3 Phase D

This phase consisted of building the database in MySQL and filling it with dummy data for the testing process. Most of the variables were stored as either varchars, numerics, or dates to keep the logic as simple as possible. A check was put into place to prevent an associate from having more than 12 properties. Next, I used a tinyint for the advertised Boolean in the rental property table. Following, to increase the speed of querying, I also created several indexes based

on commonly searched attributes within the tables. The populated test data not only keys in on the basics of the goals of the database, but also includes many of the edge cases.

1.4 Phase E

The goal of this phase was to create a client that would connect to the MySQL database through the use of a Jupyter Notebook. It was split into two parts, activities and queries. The activities focused on inserting, editing, and deleting data points in the database. It simulates common database manipulations that would be done from either the Pluto Reality website or by employees that work at their company. For the fifth and final activity, I was tasked with creating a transaction to ensure ACID properties. I went with serializable as my isolation level for the highest level of security. Pluto will not have to worry about dirty reads, non-repeatable reads, and phantom reads if employees are working with the database concurrently. Following, was the queries that would simulate common user requests. It tested the ability of my database to handle queries ranging from straight forward to complex. Queries six and seven were very specific (and probably wouldn't apply in a small test data set), but showed that the database can handle very specific questions that an employee may have. Query nine includes the use of a function. This is helpful in assisting employees who have little to no experience with coding design. Instead of trying to build the query on their own, they can take a function somebody else built and reuse it as much as they would like. Finally, query ten demonstrates the use of a trigger. When a certain event occurs, a database may need to be transformed in real time. This is where triggers come in. When one variables changes, an event occurs, and the trigger changes another variable automatically. Triggers account for certain events while maintaining low overhead. Overall, the activities and queries represent common use cases of the database for when it is shipped over to Pluto Reality Inc.

1.5 Limitations, Room for Improvement

One glaring limitation is related to advertising rental properties. All that is stored is whether or not the property is being advertised. There is nothing related to the amount of money being spent, how many advertisements there are currently, or even where the advertisements are. This could be fixed by creating an entity set that stores all advertisements. As advertisements are crucial to the success of getting properties rented, this should be prioritized.

Phase B

In this first phase, we are introduced to the customer who we will be helping in the development of their product. Their name is Pluto Reality, and they are interested in developing a database application to assist them in managing the rental of real estate properties across the nation. Thus far, the task has been to develop an ER diagram that will represent the layout of the database. The goal is to create a set of entities and their relations which can store and retrieve data efficiently and accurately. This is crucial since tracking properties across the nation will most likely create millions of data points that have to be warehoused.

Now, onto the logic that went behind designing the ER diagram. Each of the entity sets have an attribute called id which will be their primary key to be able to separate entity instances. The first entity set is for employees. It includes their employee id, first name, last name, hire date, address, telephone number, and emails. All of these attributes are crucial in being able to contact the employees. The employee instance also has 2 child classes, partner and associate. This was done as partners and associates have different roles in the company. The partners write leases and the associates both manage the rental properties and host the property viewings.

The next entity set is rental properties. The attributes here are address, property type, number of bathrooms, number of bedrooms, area square footage, monthly rent, monthly management fee, and adv. It is important to know where the property is so clients can find it. All of the other features are crucial characteristics that must be considered when buying, and is why they are included. The adv attribute also lets the associates know if advertisements have already been posted for the property. Knowing the current amount of advertisements of a property is important when deciding if a property needs more advertisements.

Following, there is the property viewing entity set. The property viewing entity set includes the date and time which is helpful in letting the associates, clients, and property owners know when a viewing will take place. It is also especially important as it serves as the connection point between all of the parties involved in the home scouting process. While the structure of how employees are stored has already been established, next is how property owners and clients are stored.

The property owner entity set is similar to the employee entity set in that it includes their name, address, and contact information. The client entity set is also similar, but includes a few extra details such as their property preferences and max rent. Having data about what they want and how much money they have is immensely valuable in narrowing down their search. At the end of the day, what matters to Pluto is setting up as many leases as possible to maximize their profitability.

Lastly, the lease entity set includes the date, monthly rent, deposit, and lease duration. Seeing what is being paid for a property and for how long is helpful in projecting the prices of future properties, which can give Pluto a leg up on their competition when selling future

properties for customers. Next, I will move onto the logic that went behind setting up the foreign key constraints.

Foreign keys are important because they prevent orphaned records from being stored and slowing down the database. Orphaned records are when one row in a table references a nonexistent row in another table. Obviously, this is bad and needs to be avoided. All rental properties must have an associate and an owner. Without an associate there is nobody to sell the property and without an owner there is no one to communicate to in selling the property. Next, all property viewings must have a property, associate, and at least one client. There is no viewing without a property, there is no one to pitch the house without an associate, and there is no point in the viewing without at least one client. A property owner must have at least one property or else there is no point in storing their data. Lastly, the lease must have a partner, a client, the property, and the property owner. The partner is needed to write the lease, the data from the property is needed for the lease, and the other two are needed to sign the lease.

Next is the cardinality constraints. The employee must be a partner or associate, but not both. Someone working at Pluto cannot have both of these jobs. There can also be one to many of both partners and associates working at the company. Having zero partners mean no leases can be written and having zero associates means there is no one to stage property viewings. Thus, a minimum cardinality of one is necessary for a partner and associate. Next, all rental properties are assigned exactly one associate and an associate can have zero to many rental properties. Having more than one associate on a rental property can create redundancy and confusion, but associates are encouraged to work on many rental properties to boost sales. A rental property has exactly one owner and an owner can have one to many rental properties. Having to deal with

many rental property owners can create confusion so it is best to only have one. An owner must have at least one property or storing their data isn't helpful.

Furthermore, a property viewing has exactly one rental property and a rental property can have zero to many viewings. Considering the distance and differences between properties, an associate hosting more than one in a viewing isn't very practical. A property can be brand new and have zero viewings thus far, or it can be around for a while and have several total viewings. A viewing has exactly one associate and an associate can have zero to many viewings. No more than one associate is necessary to try and sell the house, and an associate's workload can range from no viewings at the moment to many. Succeeding, a property viewing has one to many clients and a client can have zero to many viewings. Often times, more than one client shows up to a viewing, and clients can range from having zero viewings scheduled because they currently aren't in the market or several scheduled as they are narrowing down their options. Finally, a lease has exactly one client, rental property, property owner, and partner. All of the following can have zero to many leases.

There are also some extra entity constraints that are difficult to represent on my ER diagram. Firstly, every attribute for each entity is required. The attributes are the bare minimum information necessary for promoting efficient sales of rental properties. Next, all associates may have a max of twelve properties. Any more, and they are risking spreading themselves too thin to do a good job. Lastly, a lease must have a minimum duration of three months and maximum duration of thirty-six months. As trying to sell a property can be challenging and time consuming, the minimum lease duration minimizes short-term clients who jump from property to property. Minimizing these clients means the property will spend less of its duration being pitched to buyers and instead being used, making the property owner and company more money.

The maximum duration protects the property owner and company from outside factors such as unforeseen, rapid inflation that hurts their financial gains.

Now that what can be solved has been addressed, a few things that this E-R diagram might have some trouble trying to solve. Firstly, what happens if an associate leaves. At the moment, there is nothing in place to choose which rental properties go to which associates and how the database will change to reflect that. Another potential issue is how to sift through the database to find which properties should be prioritized for advertising. Currently, there is just a Boolean for whether a property is currently being advertised, not by how much. The rental may have a lot of potential clients and is being advertised very little, but no one would know because the Boolean is still set to true. Quantifying how much the property has been advertised is out of the scope of this assignment and is something that can't be solved.

Overall, the design of my ER diagram is built to store and retrieve relevant Pluto information quickly and in the most cost affordable way possible. Minimizing partial data and getting rid of data redundancies allows for Pluto to spend less money on databases, which are only getting more expensive year by year. My design also allows for obtaining the data employees at Pluto need, and getting it to them fast. Being able to connect multiple entity sets to retrieve the necessary data to set up events such as property viewings and leases is the most challenging design question, and it is believed that this ER diagram addresses that well.

Phase C

Phase C of the project involved mapping the conceptual design onto the relational data model. When performing the mapping, the Client, Rental Property, and Property Owner tables were left untouched. Next, instead of giving Employee its own table, I gave both of its children

their own tables. These children tables are Associate and Partner. The only difference is Associate stores the number of properties they are overseeing into num_properties and Partner stores the amount of contracts they have into num_contracts. I could have used a single Employee table and made either num_properties or num_contracts NULL depending on what job they had, but this felt less confusing.

Next, I combined the Lease table and its connecting relationships into a single table. The relationships it had with the other tables are represented with foreign keys. The foreign keys are client_id, owner_id, partner_id, and rental_id. As a client, homeowner, partner, and rental property are all necessary for a lease, these foreign key constraints will make it so a lease can't be stored if one is missing. I then took a very similar approach to the Viewing For Property table, and turned its relationships with the other tables into foreign keys as well. The foreign keys for Viewing For Property are rental_id, associate_id, and client_id. As all 3 foreign keys are necessary for a proper viewing, storing a property viewing is impossible without the data from them.

For storing who owns what property, I created a Owned By table which stores the property and the property owner. This could have also been solved by creating a property owner foreign key constraint in the Rental Property table, but I chose this approach because it is a bit less confusing for someone to interpret what is going on with the large number of tables in the database. Both rental_id and owner_id are primary and foreign keys in the Owner By table. An identical approach was taken in the Managed By table. Associate_id and rental_id are both primary and foreign keys in this table.

Lastly, I think what makes my database design different from most of the other proposed solutions is how I dealt with the multi_valued attribute. Emails is a multi-valued attribute,

making it a bit more challenging to perform queries on. To solve this, I created a Email table that gives every email its own identification number. It also has 4 foreign keys which are the 4 different types of people who may have one. There is client_id, owner_id, associate_id, and partner_id. The id of whoever's email it is will be stored into the correct foreign key, and the rest of the foreign keys will be set to NULL.

Overall, the conceptual design was very similar to the relational model, and only a few changes were made. However, creating both the conceptual and logical designs was important in trying to organize my ideas, and assisted in looking for potential weak points before jumping straight into the physical design. The next step was to begin building the model in MySQL.

Phase D

Phase D consisted of building the database in MySQL and filling it with dummy data for the testing process. Most of the attributes were able to be stored as either varchars or numerics, keeping the logic as simple to understand as possible. Dates were also stored as such (instead of varchars) to make querying by date much less complicated and more intuitive. As an associate can only have a maximum of 12 properties, I put a check on that attribute in the Associate table. This check would prevent the associate from assigning themselves more than their maximum number of properties, allowing the manager to intervene if necessary.

Following, there is a Boolean for whether a rental property has been advertised before. I solved this by using a tinyint in the Rental Property table, which serves very much the same purpose as a Boolean. As has been mentioned before, possibly having an entity set for advertisements may have proved to be more beneficial. This is since many advertisements can be

assigned to 1 property, including location data of where the advertisements are listed. However, I have decided to stick with the requirements of the document.

After, to increase the speed of querying, I also created several indexes in the design. I have four indexes for searching for an associate, partner, property owner, and client by last name. Finding someone by their last name will be very common and this will speed up the process. There is also an index which can be used to look for a property viewing by date and time. As viewings are the avenue to lots of sales, allowing clients to search based off a date and time that works for them is very important. Lastly, there is an index in the Rental Property table that includes the number of bedrooms, number of bathrooms, and area square footage. Clients are going to be using this combination of variables when narrowing down their search often, so creating an index so they spend last time waiting is a good idea.

The database has been populated with test data, and performs well even when presented with edge cases. For example, when presented an associate that is trying to add a 13th property, it rejects the request gracefully and without crashing. The database is currently in great shape for the 3rd and final phase of the project which deals with editing data in the tables and performing complex queries.

Phase E

The goal of this phase was to create a client that would connect to the MySQL database through the use of a Jupyter Notebook. The first half of the phase was dedicated to editing certain aspects of the database with a mix of MySQL and Python code, through user input.

The first activity was editing the details/characteristics of a given property. This ranged from changing the street address, to changing the rental ID primary key, to changing the property

owner foreign key. This is an important feature as details of a property may initially be incorrect or changed over time. The second activity was creating a new client. Being able to send client data from registration on a website (Jupyter Notebook is simulating this) to the database is crucial for employees looking to make connections with possible buyers. Following, is creating new viewings. As with user registration, setting up viewing appointments from the website and storing them in the database is necessary for the success of Pluto. Through the use of IDs that can be found within database, dates can be sent up between associates and clients with ease.

Over time, rental properties will obtain new owners that do not wish to work with Pluto anymore. This is where activity 4 comes in. Removing a rental property from the database interferes with many foreign key constraints, including the owned_by, viewing, managed_by, and lease tables. Removing a property from MySQL is more challenging than one would think, making this activity of immense value. Finally, is an activity that changes a partner assigned to an owner, transferring all non-expired leases to the new partner. To ensure ACID properties, this is to be done through the use of an SQL transaction. For SQL transactions, there are 4 levels of isolation. For absolute safety, I went with serializable. Here, no transactions can access the same set of data, akin to a mutual exclusion lock. Here, there is no worry about dirty reads, non-repeatable reads, and phantom reads. These activities simulate good examples of given sets of data that would be inserted, manipulated, or deleted when connecting a client to the database.

Following, I would perform ten queries that, again, would simulate common user requests. The first query was finding the names of all clients. This would be useful for sending advertisements of rental properties to all of the clients. Next, was finding all property owners and how much total square footage they own. Having knowledge of the big fish could be helpful in knowing which owners should be prioritized by employees. Third was obtaining the properties

shown by each associate in a given month. This knowledge could be of use to managers who are looking at employee performance. Fourth was finding the most popular properties in terms of viewings. Understanding why certain properties have lots of potential clients could be helpful to employees in figuring out which properties should prioritize their time in the future.

Again, finding which property owners are making the most money (as in query 5) would be helpful in deciding who the employees should continuing to try and pitch to for further business. Queries six and seven were very specific, but they show that the database can handle very specific questions that an employee may have. Query nine includes the use of a function. This is helpful in assisting employees who have little to no experience with coding design. Instead of trying to build the query on their own, they can take a function somebody else built and reuse as much as they would like. Finally, query 10 demonstrates the use of a trigger. When a certain event occurs, a database may need to be transformed in real time. The downside is it can cause a lot of overhead to keep track of certain events, which is where triggers come in. In the example, the ADV flag of a property is set to false when it is leased. This makes sense as the data on whether or not a property has been advertised becomes irrelevant when someone leases it. Thus, the data is dumped.

Overall, the activities and queries of this phase represent common use cases of the database, which Pluto employees deploy to advance their goals. The activities assist in transforming data, while the queries help employees look for trends to obtain more knowledge in their field. A database that can handle various forms of input and generate complex queries is crucial to the success of a company like Pluto.

Conclusion

Building a database isn't a straightforward process, and it takes several phases to implement a product that performs well in the face of immense data points and concurrent manipulation. It is supposed to be serviceable to any employee, no matter their level of skill when working with databases. From simple queries to the complex, it should always be functional, fast, and reliable. This database checks all of the boxes, and will be of enormous value to Pluto Reality Inc.