Project 1 Report: Pluto Reality

Benjamin Thai

University of Maryland Baltimore County

CMSC 461 Database Management

Dr. K. Kalpakis

Summary

The objective is to design and implement a functional SQL database for Pluto Realty Inc. The database will store various data on Pluto Realty's Employees, Rental properties, Property Owners, Clients, Property Viewings, and Leases that enable the company to manage its business assets and data. The Database will be designed to facilitate various queries that allow us to manage and filter the data in a way that complements Pluto Realty's business functions.

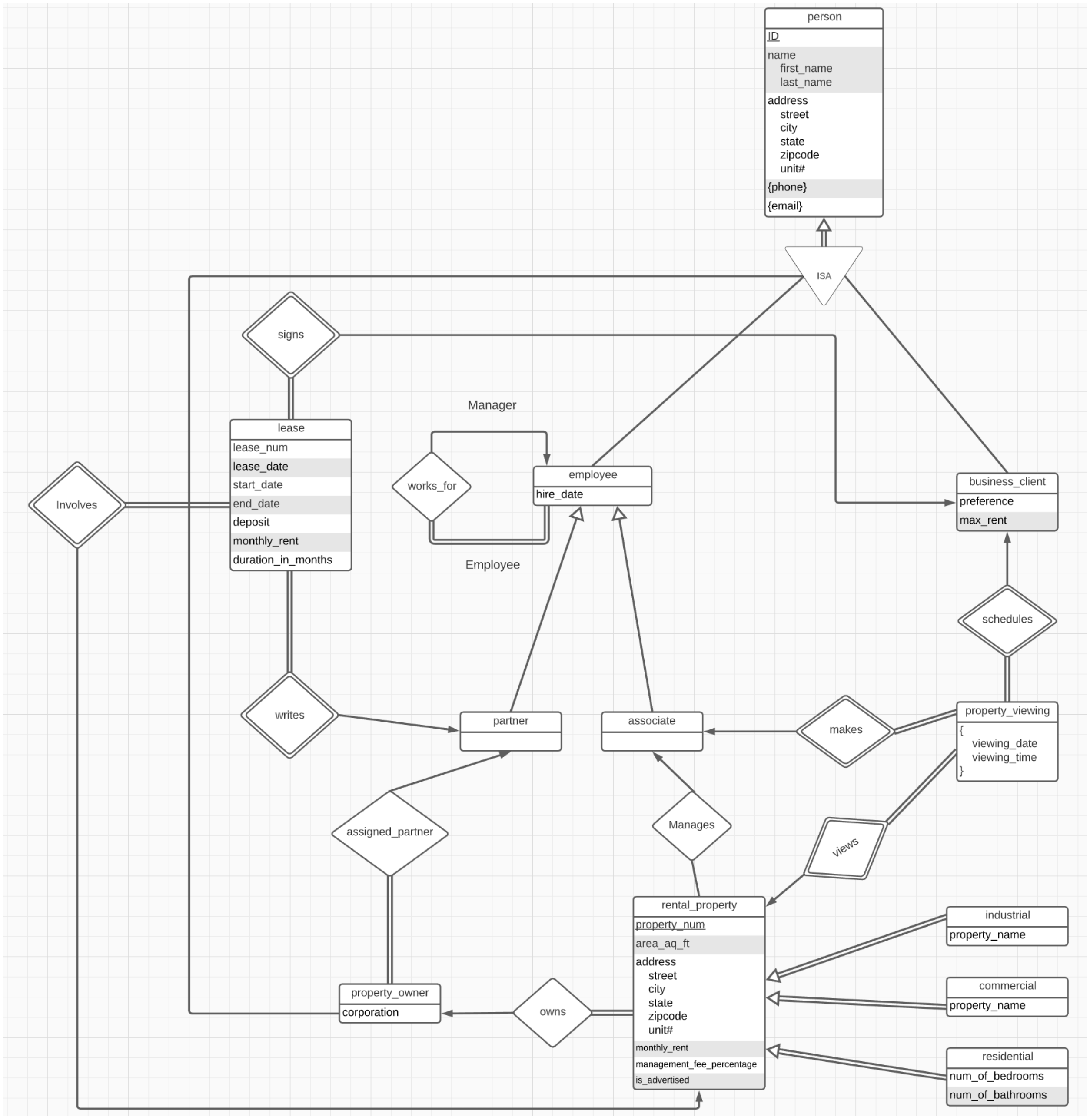
Project Phase A: Analysis of requirements and description of tasks

The requirements of this project will have us designing and implementing a database for the company Pluto Reality Inc. The database will store various data, and answer queries on Pluto Realty's Employees, Rental properties, Property Owners, Clients, Property Viewings, and Leases. The database should be able to handle queries and functions such as:

- Listing the names of all the unique clients
- Finding the unique names of owners and total square footage of all the properties they own.
- Finding the properties shown by each associate in a given month.
- Finding the most popular properties (in terms of number of viewings in a given year).
- Finding the total rent due to each property owner.
- Finding the unique names of associates supervised (directly or indirectly) by a given employee.
- Finding the unique names of owners that have a residential property in every city where Pat Doe owns a commercial property.
- Finding the top-3 partners with respect to number of properties leased in the current year.
- A SQL function to compute the total management fees due to Pluto in the last 3 months.
- A SQL trigger to automatically set to FALSE the advertisement flag of a property when it is leased.

The project will also require us to write up a project report that details our design of the database in several phases. Phase A Analysis of Requirements it will provide a high level analysis of the project requirements, Phase B Conceptual Design will plan out our Entity-Relationship model. Phase C Logical Design will have us converting our ER Model into a Relational data model and planning out our tables. Phase D Physical Design entails planning and developing the code to implement our database. And Phase E will have us implement, develop and test our database design. Phase F will provide documentation on the database's design and how to use it.

Project Phase B: Conceptual Design



Shared link to lucidchart ER design: https://lucid.app/invitations/accept/3297ba9f-809b-43ea-ada2-e0e5ea049610

Email and phone in person is multivalued. As well as date and time in property_viewing.

Project Phase C: Logical Design

- **person**(ID, first_name, last_name, street, city, state, zipcode,unit_num)
    - Primary Key: ID
- **employee**(ID, hire_date, manager_id)
    - Primary key: ID
    - Foreign key: employee(manager_id) -> employee(id),
      employee(ID) ->person(ID)
- **partner**(ID)
    - Primary Key: ID
    - Foreign key: partner(ID) -> employee(ID)
- **associate**(ID)
    - Primary Key: ID
    - Foreign key: associate(ID) -> employee(ID)
- **business_client**(ID, preference, max_rent)
    - Primary key: ID
    - Foreign Key: business_client(ID) -> person(ID)
- **property_owner**(ID, corporation, partner_id)
    - Primary key: ID
    - Foreign key: property_owner(partner_id) -> partner(ID)
      property_owner(ID) -> person(ID)
- **rental_property**(property_num, owner_id, manager_id, area_sq_ft, street, city, state, zipcode, unit#, monthly_rent, management_fee_percentage, is_advertised)
    - Primary key: property_num
    - Foreign key: rental_property(manager_id) -> associate(ID),
      rental_property(owner_id) -> property_owner(ID)
- **industrial(**property_num, property_name)
    - Primary Key: property_num
    - Foreign Key: industrial(property_num) -> rental_property(property_num)
- **commercial(**property_num, property_name)
    - Primary Key: property_num
    - Foreign Key: commercial(property_num) -> rental_property(property_num)
- **residential(**property_num, num_of_bedrooms, num_of_restrooms)
    - Primary Key: property_num
    - Foreign Key: residential(property_num) -> rental_property(property_num)

- **lease**(partner_id, property_num , client_id, lease_num, lease_date, start_date, end_date, deposit, monthly_rent, duration_in_months)
    - Primary key: partner_id, property_num , client_id, lease_num
    - Unique: lease_num, property_num
    - Foreign Key Ref: lease(partner_id) -> partner(ID),
      lease(client_id) -> business_client(ID),
      lease(property_num) -> rental_property(property_num)
- **property_viewing**(associate_id, client_id, property_num , viewing_date, viewing_time)
    - Primary key: associate_id, client_id, property_num, date
    - Foreign Key ref: property_viewing(associate_id) -> associate(ID),

      property_viewing(client_id) -> business_client(ID

      property_viewing(property_num) -> rental_property(property_num)
- **email**(ID, email_address)
    - Primary Key: ID, email_address
    - Unique: email_address
    - Foreign Key ref: email(ID) -> person(ID)
- **phone**(ID, phone_number, type)
    - Primary Key: ID, phone_number
    - Unique: phone_number
    - Foreign Key ref: phone (ID) -> person(ID)

- Since email and phone in the person entity is a multivalued attribute, email and phone was split into a separate relational schema that will include the ID of a person tuple as a primary key to link to an email or phone.
- Since a weak entity includes the primary key of the identifying strong entity, the relationship set linking the weak entity to a strong entity is redundant. So, the relationship sets “involves”, “signs”, “writes”, “views”, “makes”, “schedules” can be left out of the final reduced relation schema design. And the weak entities “lease” and “property_viewing” will have primary keys of their respective strong entities
- The works_for relational set is a many to one total participation from the employee entity to itself so we merge the “works_for” relationship schema into employee adding the attribute manager_id which points to the employee id of the manager of that respective employee. And give it appropriate foreign key restraints. And we can then discard the works_for relation.
- the "assigned_partner" relationship set is also a many to one total participation so we can merge it with the "property_owner" entity by adding an additional attribute to property_owner named partner_id that corresponds to the partners employee ID and we can then discard the “assigned_partner” relationship. partner_id will have a foreign reference constraint on the ID attribute from the partner table added to it.
- The “owns” relationship is also a many to one total participation so we can merge and discard it into the “rental_property” entity. “rental_property” will gain an attribute for the property owner ID and respective foreign constraints.
- The "manages" relationship set its many to one but is not a total participation what we can do is the same thing we did with the “owns”, “assigned_partner”, and “works_for” relations but we’ll have to deal with the possibility of some null values in the merged attribute.

Project Phase D: Physical Design

The SQL code for creating, deleting, and initial population of the database tables are in separate files named createAll, loadAll, and dropAll. There are a few things to note for the following tables:

- “**lease**” table:
    - Date values in the “lease” the table should be formatted as ‘yyyy-mm-d’
    - Duration_in_months column is auto calculated based on the inserted start and end dates values and the duration is calculated in months. therefore when inserting an entry there is no need to specify a value for the Duration_in_months column in the parameter list of the insert statement.
    - There is a check constraint that the duration is at least 3 months and at most 36 months long.
    - The “lease” property_num and client_id values is set to delete cascade and a “lease” entry will be deleted if a corresponding property or client is deleted.
- “**property_viewing**” table:
    - The “property_viewing” client_id and property_num values is set to delete cascade and a “property_viewing” entry will be deleted if a corresponding property or client is deleted.
- “**Rental_property**” table:
    - There is a trigger named max_assigned_properties that prevents more than 12 properties to be assigned to an associate as that is the requirements of the project.

- "**Employee**", "**Associate**", "**partner**" table
  - The "lease" and "property_owner" table has foreign key references to the "partner" table and will send an error if you try to delete a partner employee without first removing or redirecting the foriegn reference in "lease" and "property_owner". The "property_owner" table also has a foriegn key ref. from the "rental_property" table so take note of that if trying to clear out a property_owner reference to a partner employee.
  - The "rental_property" table also references the associate table however it will set the manager_id value to null if an associate employee is deleted instead of throwing an error.
  - The "property_viewing table" references the associate table and will send an error if trying to delete an associate if the corresponding property_view entry isn't redirected or removed.
  - Employees manager_id references another employee and may need to redirect other employees manager_id foreign constraints. if trying to delete an employee
- "**business_client**" table:
  - The "lease" and "property_viewing" table is set to Delete cascade if a corresponding business_client is deleted.
- **Phone** and **email** tables
  - Phone and email are set to delete cascade and will be removed if a person entry is removed.

Due to the layered foreign reference keys inserting can be a little tedious for some tables like "associate" and "partners". So the createAll file also has some stored procedures, That makes it easier to insert a partner, associate, client, and property owner entries, as well as industrial, commercial, and residential property entries. below are details of the following stored procedures:

- **insert_employee**(ID, first_name, last_name, street, city, state, zipcode, unit_num, hire_date, manager_id)
    - This procedure will take the appropriate values and first insert an entry into the "person" table to satisfy the "employee" table's foriegn constraints. then use the appropriate values to make a corresponding entry in the "employee" table. (**Note:** this procedure may require that there be as least one employee in the "person" table to give a value for the manager_id)
    - The parameters are of the following data types:
        - INT ID
        - VARCHAR(50) first_name
        - VARCHAR(50) last_name
        - VARCHAR(50) street
        - VARCHAR(50) city
        - VARCHAR(50) state
        - INT zipcode
        - INT unit_num
        - DATE hire_date
        - INT manager_id
- **insert_partner**(ID, first_name, last_name, street, city, state, zipcode, unit_num, hire_date, manager_id)
    - This procedure will take the appropriate values and first insert an entry into the "person" table to satisfy the "employee" table's foriegn constraints. then use the appropriate values to make an entry in the "employee" table to satisfy its foriegn constraints. then adds a corresponding entry into the "partner" table. (**Note:** this procedure may require that there be as least one employee in the "person" table to give a value for the manager_id)
    - The parameters are of the following data types:
        - INT ID
        - VARCHAR(50) first_name
        - VARCHAR(50) last_name
        - VARCHAR(50) street
        - VARCHAR(50) city
        - VARCHAR(50) state
        - INT zipcode
        - INT unit_num
        - DATE hire_date
        - INT manager_id

- **insert_associate**(ID, first_name, last_name, street, city, state, zipcode, unit_num, hire_date, manager_id)
    - This procedure will take the appropriate values and first insert an entry into the “person” table to satisfy the “employee” table’s foriegn constraints. then use the appropriate values to make an entry in the “employee” table to satisfy its foriegn constraints. then adds a corresponding entry into the “associate” table. (**Note:** this procedure may require that there be as least one employee in the “person” table to give a value for the manager_id)
    - The parameters are of the following data types:
        - INT ID
        - VARCHAR(50) first_name
        - VARCHAR(50) last_name
        - VARCHAR(50) street
        - VARCHAR(50) city
        - VARCHAR(50) state
        - INT zipcode
        - INT unit_num
        - DATE hire_date
        - INT manager_id

- **insert_property_owner**(ID, first_name, last_name, street, city, state, zipcode, unit_num, corporation, partner_id)
    - This procedure will take the appropriate values and first insert an entry into the “person” table to satisfy the “property_owner” table’s foriegn constraints. then use the appropriate values to make a corresponding entry in the “property_owner” table.
    - The parameters are of the following data types:
        - INT ID
        - VARCHAR(50) first_name
        - VARCHAR(50) last_name
        - VARCHAR(50) street
        - VARCHAR(50) city
        - VARCHAR(50) state
        - INT zipcode
        - INT unit_num
        - TEXT corporation
        - INT partner_id

- **insert_business_client**(ID, first_name, last_name, street, city, state, zipcode, unit_num, preference, max_rent)
    - This procedure will take the appropriate values and first insert an entry into the "person" table to satisfy the "business_client" table's foriegn constraints. then use the appropriate values to make a corresponding entry in the "business_client" table.
    - The parameters are of the following data types:
        - INT ID
        - VARCHAR(50) first_name
        - VARCHAR(50) last_name
        - VARCHAR(50) street
        - VARCHAR(50) city
        - VARCHAR(50) state
        - INT zipcode
        - INT unit_num
        - TEXT preference
        - INT max_rent

- **insert_industrial_property**(property_num, owner_id, manager_id, area_sq_ft, street, city, state, zipcode, unit_num, monthly_rent, management_fee_percentage, is_advertised, property_name)
    - This procedure will take the appropriate values and first insert an entry into the "rental_property" table to satisfy the "industrial" table's foriegn constraints. then use the appropriate values to make a corresponding entry in the "industrial" table.
    - management_fee_percentage should be a decimal representation of a percentage
    - The parameters are of the following data types:
        - INT property_num
        - INT owner_id
        - INT manager_id
        - INT area_sq_ft
        - VARCHAR(50) street
        - VARCHAR(50) city
        - VARCHAR(50) state
        - INT zipcode
        - INT unit_num
        - DOUBLE monthly_rent
        - DOUBLE management_fee_percentage
        - BOOL is_advertised
        - VARCHAR(50) property_name

- **insert_commercial_property**(property_num, owner_id, manager_id, area_sq_ft, street, city, state, zipcode, unit_num, monthly_rent, management_fee_percentage, is_advertised, property_name)
    - This procedure will take the appropriate values and first insert an entry into the "rental_property" table to satisfy the "commercial_property" table's foriegn constraints. then use the appropriate values to make a corresponding entry in the "commercial_property" table.
    - management_fee_percentage should be a decimal representation of a percentage
    - The parameters are of the following data types:
        - INT property_num
        - INT owner_id
        - INT manager_id
        - INT area_sq_ft
        - VARCHAR(50) street
        - VARCHAR(50) city
        - VARCHAR(50) state
        - INT zipcode
        - INT unit_num
        - DOUBLE monthly_rent
        - DOUBLE management_fee_percentage
        - BOOL is_advertised
        - VARCHAR(50) property_name
- **insert_residential_property**(property_num, owner_id, manager_id, area_sq_ft, street, city, state, zipcode, unit_num, monthly_rent, management_fee_percentage, is_advertised, num_of_bedrooms, num_of_restrooms)
    - This procedure will take the appropriate values and first insert an entry into the "rental_property" table to satisfy the "residential_property" table's foriegn constraints. then use the appropriate values to make a corresponding entry in the "residential_property" table.
    - management_fee_percentage should be a decimal representation of a percentage
    - The parameters are of the following data types:
        - INT property_num
        - INT owner_id
        - INT manager_id
        - INT area_sq_ft
        - VARCHAR(50) street
        - VARCHAR(50) city
        - VARCHAR(50) state
        - INT zipcode
        - INT unit_num
        - DOUBLE monthly_rent
        - DOUBLE management_fee_percentage
        - BOOL is_advertised
        - INT num_of_bedrooms

        - INT num_of_restrooms

Additionally There will be a user interface made in a jupyter notebook that will implement various SQL queries to return specific data and functional requirements listed in the design specifications. including:

- List the names of all the unique clients.
- Find the unique names of owners and total square footage of all the properties they own.
- Find the properties shown by each associate in a given month.
- Find the most popular properties (in terms of number of viewings in a given year).
- Find the total rent due to each property owner.
- Find the unique names of associates supervised (directly or indirectly) by a given employee.
- Find the unique names of owners that have a residential property in every city where Pat Doe owns a commercial property.
- Find the top-3 partners with respect to number of properties leased in the current year.
- SQL function to compute the total management fees due to Pluto in the last 3 months.
- a SQL trigger to automatically set to FALSE the advertisement flag of a property when it is leased.

Project Phase E & F: Prototype, Development, Testing, and User Guide

There is a Jupyter notebook named User Guide.ipynb with code to access, populate, update, and administer the SQL tables

There is a Jupyter notebook named Functional Requirement Interface.ipynb to serve as an interface for the functional requirements.

The createAll.sql file also has some index definitions.

There was no usage of transactions for the inital creation of the database so the transactions.sql file is empty