CATS: CLIENT – ACCOUNTANT TAX SYSTEM

Documentation Prepared by Makayla Lerner

For INSC 484

On December 2, 2022

DESCRIPTION:

The accounting firm, Cool Cats Do Taxes, has hired a consultant, Makayla Lerner, to design a system to track their various business practices.

REQUIREMENTS:

The database must store:

* Information about CPAs (Certified Public Accountants)
* Information about clients, including name, address, income, and if they have provided materials to the firm.
* Information about the tax return, including status, timestamp of filing, and if it was checked by a CPA

The user must be able to:

* Add clients
* Add tax filing assistants
* Add CPAs
* Mark if a client has provided their required materials
* Check if a client has provided their required materials
* Mark if a tax return is filed
* Check if a tax return is filed
* Mark if a CPA checked a return
* Check if a CPA checked a return

ASSUMPTIONS:

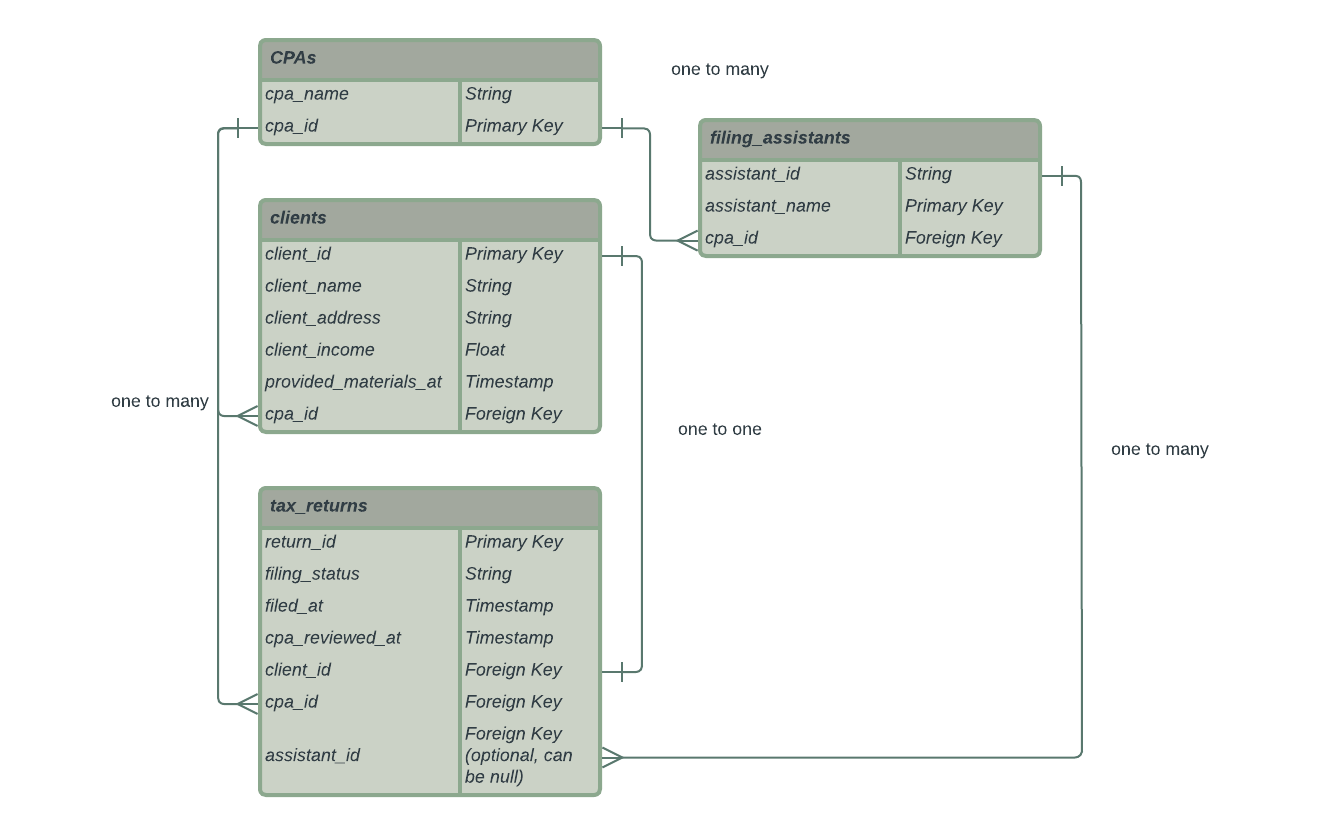
The following assumptions were considered in the design of the system:

* Each client only has one tax return, each tax return has one CPA and one assistant
* Users update regularly when status changes occur
* Each CPA can have multiple clients, multiple tax returns, and multiple assistants

DESIGN

The database:

Below is an entity relationship diagram of the system, where there is a one-to-many relationship between CPAs and clients, CPAs and tax returns, CPAs and assistants, and assistants and tax returns. There is a one-to-one relationship between clients and tax returns.



The software:

The system consists of 8 files:

* .env file contains the link to the database using ElephantSQL
* requirements.txt contains the library versions needed to implement the code
* schema.sql contains the schema of the database to set it up
* reset.sql contains the queries necessary if the database needs to be reset
* main.py holds the user interface and is the file to be run when executing
* orm.py holds the object relational model, which consists of the object classes
* body.py holds most functions necessary for the system, including prompts and tasks
* connection\_pool.py holds the database connection that is used when a user makes changes in the system

The software was designed with modularity in mind. Functions were created for each prompt sent to the user and for each task the user would like to complete. The basic interface is created from a simple run function in the main.py file that contains a while loop to prompt the user depending on their input.

Statements are shown to the user after each action to ensure that the user does not attempt to repeat actions due to no feedback.

The object relational model consists of a parent class, DatabaseObject, which utilizes class methods that are common between the entities: loading from the database, searching by name, finding all objects in the database, creating an object from a row, and saving to the database. The parent class also includes three abstracted methods, denoted with an underscore before the class name (ex. \_insert), this is because Python does not have private methods so this is a way to simulate a private method within the class. Each entity has its own class that holds its respective unique attributes. An exception is seen within the TaxReturn object, which has its own method for searching the database by name because the clients and tax return tables needed to be joined.

Nearly all functions require an ID to access or update values, so a search ID by name function is provided in the main interface.

The connection pool file uses the connection factory configuration to have cursors return dictionary values. This was done to ensure that when we are loading objects from the database, the column names are saved as keys and can be seen in an output to the user.