DATABASE DESIGN

Tracy Mazelin

RE-DESIGN DECISIONS

I have decided to change my database from SQLite to PostgreSQL. The reason for this change is that I would like to host the application using Heroku. The Heroku platform is different to other deployment platforms because it has an ephemeral file system. This means that Heroku can reset my server at any time and since SQLite is saved to the filesystem, I would lose the database everytime Heroku recycles the servers. For this reason, I am switching to PostgreSQL.

Additionally, I modified my database design in the following ways (see diagram below):

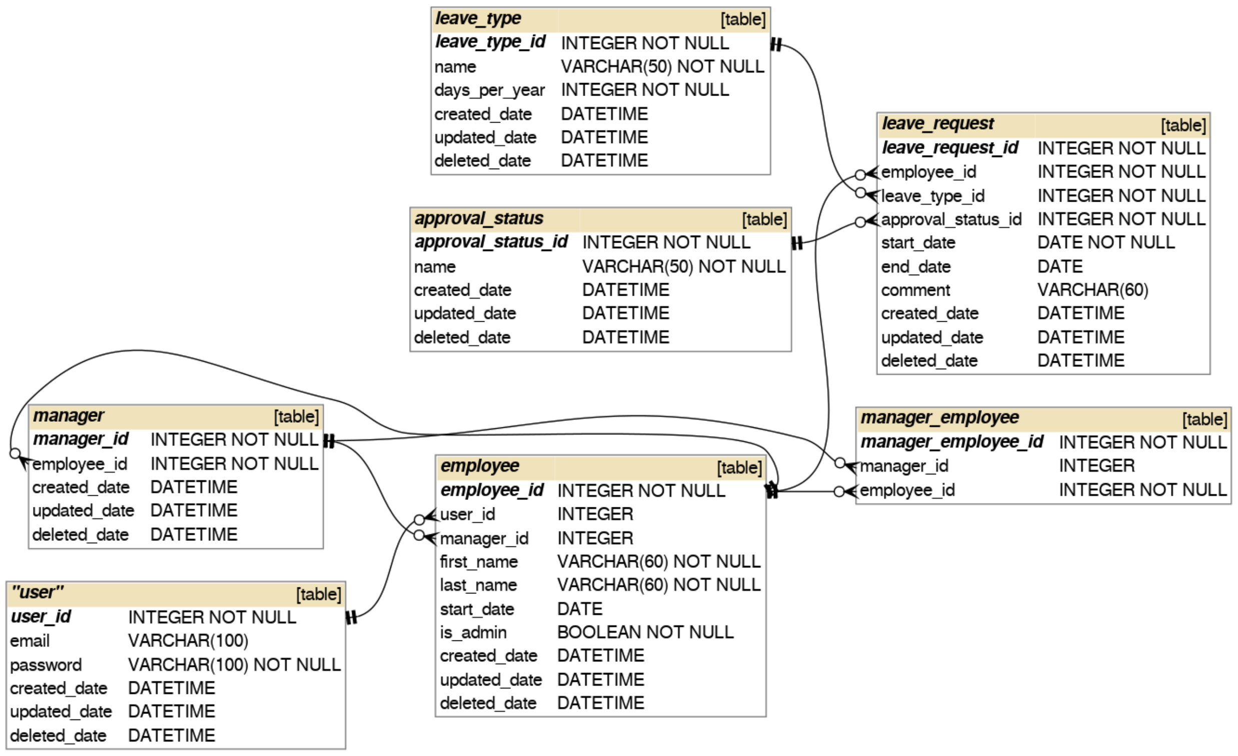
- Named the id columns in every table more explicitly. Instead of just “id” they are now “employee_id”, “leave_id” etc.
- Added more audit columns to many of the tables to keep track of updated and deleted dates
- Added a manager table and a manager-employee table to serve as a linking table for keeping track of the many to many relationship of the manager and employee entities. One manager can have many employees reporting to them and an employee can have many managers over the course of their tenure with the company.

TECHNOLOGY CHOICE WITH EXPLANATION

I am choosing to store my data in a relational database and the technology I would like to do this with is PostgreSQL. I am choosing to use a relational database because my data will have a lot of relationships and will need to be normalized. Also, when rows change in the leave request table, there will be cascading effects to the employee, leave_type and approval_status tables. Each of the tables in my application will be related to other tables.

I am choosing to use PostgreSQL for a number of reasons. First, PostgreSQL is free and open source and works well with Heroku. It is a powerful, object-relational database system with over 30 years of active development. It is a good technology to know how to work with. It also has python bindings and my application will be built with python. Lastly, it is an established technology and has community support so I will be able to find answers to any technical issues I may encounter.

TABLES



leave_type	[table]
leave_type_id	INTEGER NOT NULL
name	VARCHAR(50) NOT NULL
days_per_year	INTEGER NOT NULL
created_date	DATETIME
updated_date	DATETIME
deleted_date	DATETIME

leave_request	[table]
leave_request_id	INTEGER NOT NULL
employee_id	INTEGER NOT NULL
leave_type_id	INTEGER NOT NULL
approval_status_id	INTEGER NOT NULL
start_date	DATE NOT NULL
end_date	DATE
comment	VARCHAR(60)
created_date	DATETIME
updated_date	DATETIME
deleted_date	DATETIME

approval_status	[table]
approval_status_id	INTEGER NOT NULL
name	VARCHAR(50) NOT NULL
created_date	DATETIME
updated_date	DATETIME
deleted_date	DATETIME

manager	[table]
manager_id	INTEGER NOT NULL
employee_id	INTEGER NOT NULL
created_date	DATETIME
updated_date	DATETIME
deleted_date	DATETIME

manager_employee	[table]
manager_employee_id	INTEGER NOT NULL
manager_id	INTEGER
employee_id	INTEGER NOT NULL

employee	[table]
employee_id	INTEGER NOT NULL
user_id	INTEGER
manager_id	INTEGER
first_name	VARCHAR(60) NOT NULL
last_name	VARCHAR(60) NOT NULL
start_date	DATE
is_admin	BOOLEAN NOT NULL
created_date	DATETIME
updated_date	DATETIME
deleted_date	DATETIME

"user"	[table]
user_id	INTEGER NOT NULL
email	VARCHAR(100)
password	VARCHAR(100) NOT NULL
created_date	DATETIME
updated_date	DATETIME
deleted_date	DATETIME

USER TABLE:

The user table holds the user account information required for login. It is tied to the employee table. Each employee can only have one user record. The primary key of the user table is the id field which cannot be empty.

NOTE: I will be utilizing the flask_login package to handle authentication. Therefore, these methods are not included in my service layer because they will be handled by the package.

EMPLOYEE:

The employee table holds information about the employee such as name, start date, and whether they are an administrative employee. If they are an administrator, they will have access to add, edit, and remove employees. It also holds the employee's user id and manager id. If the employee is a manager, the manager id will be null. If the employee has a manager, the manager id will be the employee id of the manager. An employee can only have one associated user record.

LEAVE_TYPE:

The leave type table will hold the various types of leave an employee can request time off for. For example: sick, bereavement, jury duty or vacation. A leave request can only have one associated leave type.

APPROVAL_STATUS:

The approval status table will hold the status of a leave request. The status types will be approved, denied, or pending. A leave request can only have one approval status type.

LEAVE_REQUEST:

The leave request table will hold the details about a leave request such as the employee making the leave request, the leave request type, the leave request status and other data about the leave request. An employee can make many leave requests but a leave request can only be associated with one employee.

MANAGER:

The manager table will hold the employee id for an employee who is also a manager. Since the manager employee relationship is many to many, a linking table (manager_employee) will exist.