

API Web server for HR management

R1 - Identification of the problem you are trying to solve by building this particular app.

The HR management system is used as an important tool for strategic personnel management. It is an integrated HR management system that helps effective decision-making through integrated data management. Support performance management of each employee and increase employee satisfaction. Therefore, the HR department is intended to manage all employees in the company effectively. You can see at a glance the wages, job positions, dates of employment, and departments of workers. When a request to create, modify, or delete an employee from the frontend page of HR management system, each API in this application can manipulate the employee's data in the database.

This application is accessible only to staffs working at the HR department. This is because there is sensitive private information of workers.

R2 -Why is it a problem that needs solving?

A company that manages employees will have a demand to manage employees efficiently. There is a need for a management system that allows employees to focus on their work, such as employee's department movement, current salary, and current department.

Without such a system, it becomes very difficult to respond when creating next year's budget, creating new departments, or eliminating existing departments.

Therefore, this application is needed for CEO or staff in the HR department to make decisions quickly and manage employees efficiently.

R3 - Why have you chosen this database system. What are the drawbacks compared to others?

The most important characteristic of using a database to create a system is that it manages structured data. Data can be organised in columns such as 'ID' and 'password', sorted by some criteria, and filtered. Therefore, using the database to create this application is because adding/updating/deleting/retrieving data is more structured and faster. Among them, the reason RDB such as PostgreSQL was selected is that it is very suitable for the main database for providing service. In addition, all the schema and db models of this application are fixed and will be used within the HR department, so it does not need extremely high speed.

comparison to No SQL database

Because NoSQL is 'schemaless', data can be stored more flexibly, but RDBs used by this application rarely have schema change. On average, it is true that NoSQL is faster than RDB, and when using the same cost, NoSQL database is cost-effective in terms of performance. But this depends on which RDB you use. NoSQL is recommended when the exact data structure is unknown and data can be changed or expanded. However, if data duplication can occur, all collections must be modified when duplicate data is changed, so it is good for systems that do not have many updates. On the other hand, because RDB guarantees data integrity and is easy to change, it is suitable for this application where related data is frequently changed.

Pros and Cons of RDB

Benefits

As I mentioned above, data should be saved according to the established schema, ensuring a clear data structure. There is no duplication of data, so data consistency can be guaranteed.

Drawbacks

As the system grows, complex queries with many JOIN statement will make the system slow down. If you want to add columns in tables having a lot of data, you must alter table and create a new table. Moreover, improving hardware performance can be costly.

Reference: <https://peps.python.org/pep-0008/>

Reference: <https://docs.rackspace.com/support/how-to/choosing-between-rdbms-and-nosql/>

R4 - Identify and discuss the key functionalities and benefits of an ORM

What is ORM?

ORM(Object-Relational Mapping) is simply setting connections between objects in object_oriented programs and relational database. To use RDB, you must use SQL. However, ORM automatically converts the written Python code into a SQL query of the relational DB, allowing developers to manipulate the DB only by writing Python code without having to write a separate SQL query. For example, below is an example of a SQLAlchemy model definition. We create a class named Contact with SQLAlchemy

```
class Contact(db.Model):
    __tablename__ = 'contacts'
    id = db.Column(db.Integer, primary_key=True)
    first_name = db.Column(db.String(100))
    last_name = db.Column(db.String(100))
    phone_number = db.Column(db.String(32))
```

If we do the migration, we can get a Contact table in Database even though we don't write any create table statement.

```
CREATE TABLE CONTACTS(  
    ID INT PRIMARY KEY NOT NULL,  
    FIRST_NAME CHAR(100) NOT NULL,  
    LAST_NAME CHAR(100) NOT NULL,  
    PHONE_NUMBER CHAR(32) NOT NULL,  
);
```

Reference: <https://www.fullstackpython.com/sqlalchemy.html>

SQLAlchemy handle the table creation by using ORM. IT can be seen a table create statement was created so that a table could be created just like the class.

In addition, all records can be retrieved by using SQLAlchemy in Python code such as `contacts = Contact.query.all()` instead of a plain SQL, `SELECT * FROM contacts`.

Benefits of ORM

No need to create declaration, assignment in programs, so we can reduce development time. Once you write your data model, ORM creates Table automatically so we can improve the productivity. Also Model use OOP(Object-Oriented Programming), we can speed up development by extending and inheriting from Models. SQL injection is not easy as queries are sanitised.

Reference: <https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/>

Reference: <https://www.learnnowonline.com/blogs/2012/08/28/4-benefits-of-object-relational-mapping-orm>

Reference: <https://midnite.uk/blog/the-pros-and-cons-of-object-relational-mapping-orm>

R5 - Document all endpoints for your API

Documentation of all endpoints By POSTMAN

I created documentation of all endpoints by using POSTMAN.

Documentation By POSTMAN

- Login
 - HTTP request verb : POST
 - Required data where applicable : The email and password are needed in JSON
 - Expected response data : The email, encoded password and token using JWT

- Authentication methods where applicable: n/a
- Get all HR staffs
 - HTTP request verb : GET
 - Required data where applicable : N/A
 - Expected response data : 'id', 'is_admin', 'employees', 'employee_id', 'email'
 - Authentication methods where applicable: N/A
- Add a new HR staff
 - HTTP request verb : POST
 - Required data where applicable : employee_id, is_admin
 - Expected response data : 'id', 'is_admin', 'employees', 'employee_id', 'email'
 - Authentication methods where applicable: Token by JWT
- Delete a HR staff
 - HTTP request verb : DELETE
 - Required data where applicable : id parameter
 - Expected response data : Delete confirmation message
 - Authentication methods where applicable: Token by JWT
- Update a HR staff
 - HTTP request verb : PUT
 - Required data where applicable : One of fields in hrstaffs table and id parameter-
 - Expected response data : 'id', 'is_admin', 'employees', 'employee_id', 'email'
 - Authentication methods where applicable: Token by JWT
- Get all employees
 - HTTP request verb : GET
 - Required data where applicable : N/A
 - Expected response data : 'id', 'name', 'email', 'password', 'salary', 'hire_date', 'job_id', 'department_id', 'job', 'department'
 - Authentication methods where applicable: N/A
- Get a employee
 - HTTP request verb : GET
 - Required data where applicable : id parameter
 - Expected response data : 'id', 'name', 'email', 'password', 'salary', 'hire_date', 'job_id', 'department_id', 'job', 'department'
 - Authentication methods where applicable: N/A

- Add a new employee
 - HTTP request verb : POST
 - Required data where applicable : All fields of employees table
 - Expected response data : 'id', 'name', 'email', 'password', 'salary','hire_date', 'job_id', 'department_id', 'job', 'department'
 - Authentication methods where applicable: Token by JWT

- Delete a employee
 - HTTP request verb : DELETE
 - Required data where applicable : id parameter
 - Expected response data : Delete confirmation message
 - Authentication methods where applicable: Token by JWT

- Update a employee
 - HTTP request verb : PUT
 - Required data where applicable : fields of employees that need to update and id parameter
 - Expected response data : 'id', 'name', 'email', 'password', 'salary','hire_date', 'job_id', 'department_id', 'job', 'department'
 - Authentication methods where applicable: Token by JWT

- Get all department
 - HTTP request verb : GET
 - Required data where applicable : N/A
 - Expected response data : 'id', 'department_name'
 - Authentication methods where applicable: N/A

- Get a department
 - HTTP request verb : GET
 - Required data where applicable : id parameter
 - Expected response data : 'id', 'department_name'
 - Authentication methods where applicable: N/A

- Add a department
 - HTTP request verb : POST
 - Required data where applicable : fields in departments table
 - Expected response data : 'id', 'department_name'
 - Authentication methods where applicable: Token by JWT

- Delete a department
 - HTTP request verb : DELETE
 - Required data where applicable : id parameter
 - Expected response data : Delete confirmation message
 - Authentication methods where applicable: Token by JWT
- Update a department
 - HTTP request verb : PUT
 - Required data where applicable : fields of department table that need to update and id parameter
 - Expected response data : 'id', 'name', 'email', 'password', 'salary','hire_date', 'job_id', 'department_id', 'job', 'department'
 - Authentication methods where applicable: Token by JWT
- Get all jobs
 - HTTP request verb : GET
 - Required data where applicable : N/A
 - Expected response data : 'id', 'job_position'
 - Authentication methods where applicable: N/A
- Get a job
 - HTTP request verb : GET
 - Required data where applicable : id parameter
 - Expected response data : 'id', 'job_position'
 - Authentication methods where applicable: N/A
- Add a new job
 - HTTP request verb : POST
 - Required data where applicable : All fields in jobs table
 - Expected response data : 'id', 'job_position'
 - Authentication methods where applicable: Token by JWT
- Update a job
 - HTTP request verb : PUT
 - Required data where applicable : fields in jobs table need to update
 - Expected response data : 'id', 'job_position',
 - Authentication methods where applicable: Token by JWT
- Delete a job

- HTTP request verb : DELETE
- Required data where applicable : id parameter
- Expected response data : Delete confirmation message
- Authentication methods where applicable: Token by JWT

R6. An ERD for your app

Database Modeling

Analyse the requirements of my application

- Employees has Name, Email, Password, Hire_date, Salary, Job_id, and Department_id
- Department has department_name
- Job has job_position
- HR staffs have employee_id, is_admin
- Department has many employees
- Job_position be assigned to many employees
- Only Employees in HR department can register in HR staff

Conceptual Data Model

- We can create Conceptual data model based on the requirements analysis by extracting entities and attributes.

Entity	Attributes
Employees	<u>ID</u> , Name, Email, Password, Hire_date, Salary, Job_id, and Department_id
Departments	<u>ID</u> , department_name
Jobs	<u>ID</u> , job_position
Hrstaffs	<u>ID</u> , Employee_id, is_admin

Why did each entities have this structure?

Employees

The HR management system requires personal information of employees, 'salary' for annual salary management, 'job_position' for promotion management, 'department_id' for career management, and 'hire_date' information. Therefore, the employees entity which is the main entity of the HR management system is managed with this structured data.

Departments

For employees' movement between departments and career management, department entity has ID(PK), department_name.

Jobs

For promotion management and career management of employees, ID(PK), job_position attribute is in the jobs entity

Hrstaffs

It is necessary to manage the HR management system by registering the staff in HR department. Therefore, hrstaffs has ID(PK), 'employee_id' and 'is_admin' attributes

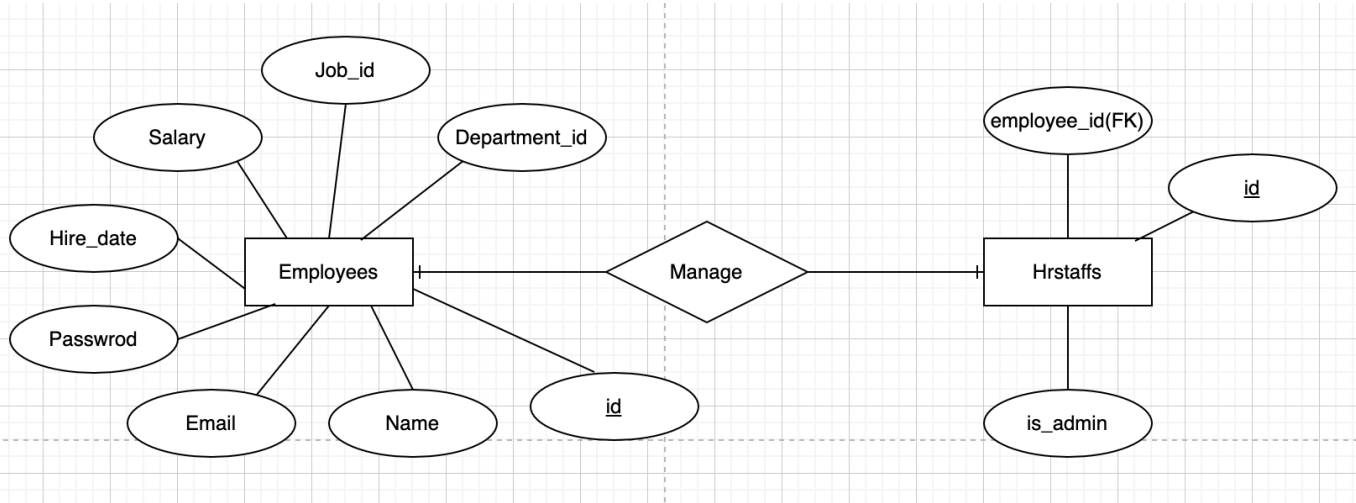
Establish a relationship between the entities based on the entities and the requirements above

Relation	Entities participating in the relation	Cardinality
manage	Hrstaffs : hrstaffs has one employess Employees: employees has one hrstaffs	1:1
have	Departments: departments has many employess Employees: employess has one deparments	1:N
have	Jobs: jobs has many employees Employees: employees have one jobs	1:N

Cardinality

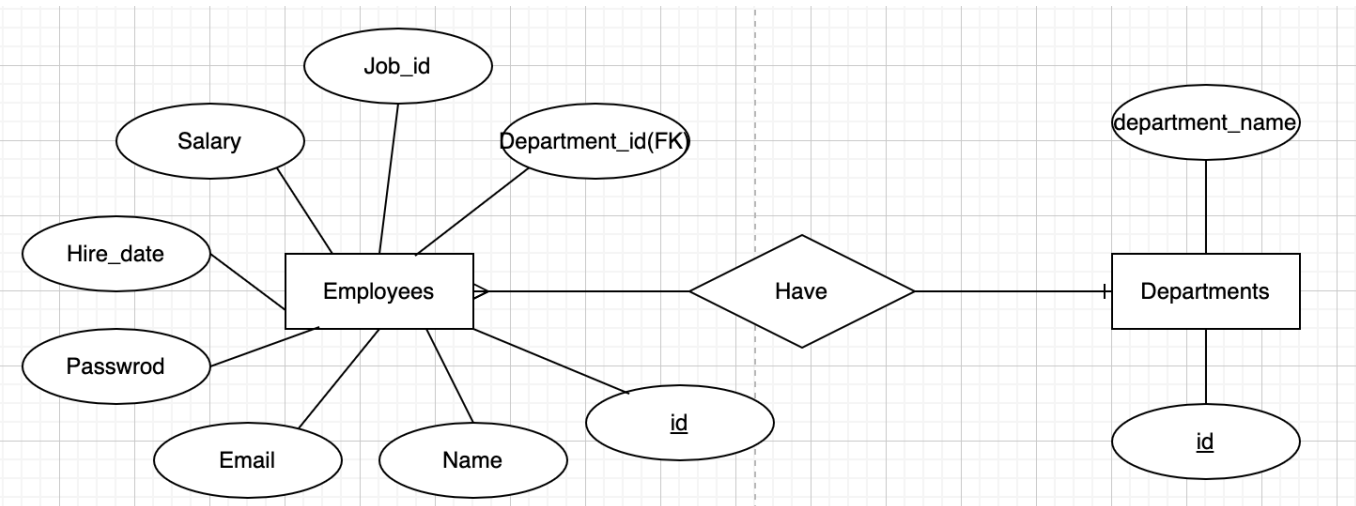
[Hrstaffs] : [Employees] = 1 : 1

The hrstaff entity is created in the employee entity. Only employees who work in the HR department among those registered in the employee entity are registered in hrstaffs entity. Thus, the relationship between both is bound to 1:1.



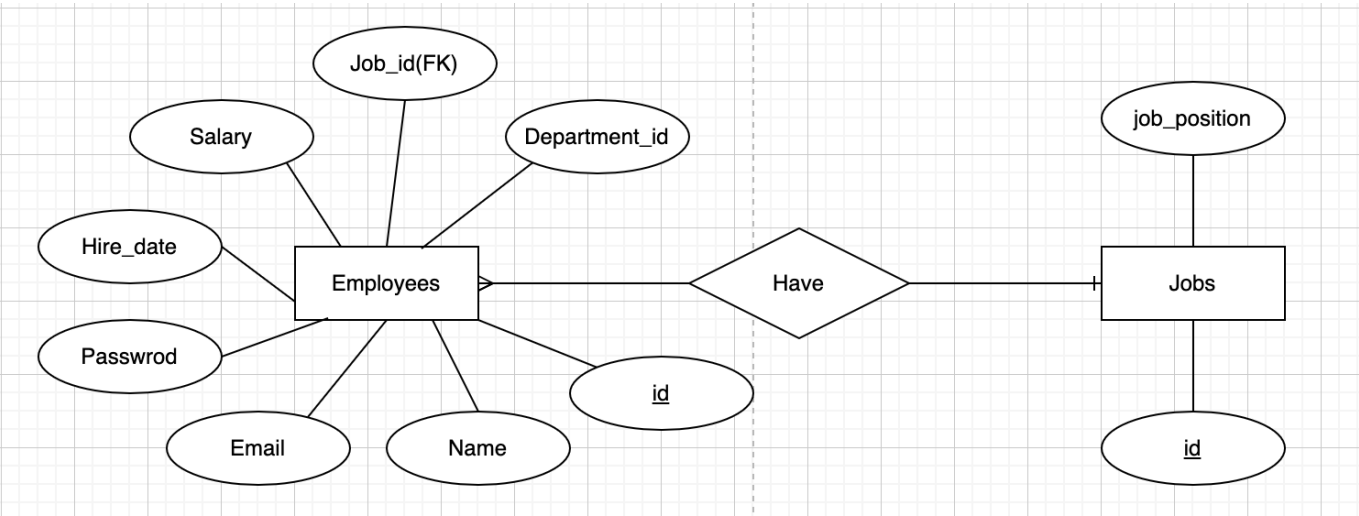
[Departments] : [Employees] = 1 : N

As I mentioned in the requirements analysis, a department has multiple employees. So this relationship is represented as 1:N. The department entity becomes the parent and the employee becomes the child.

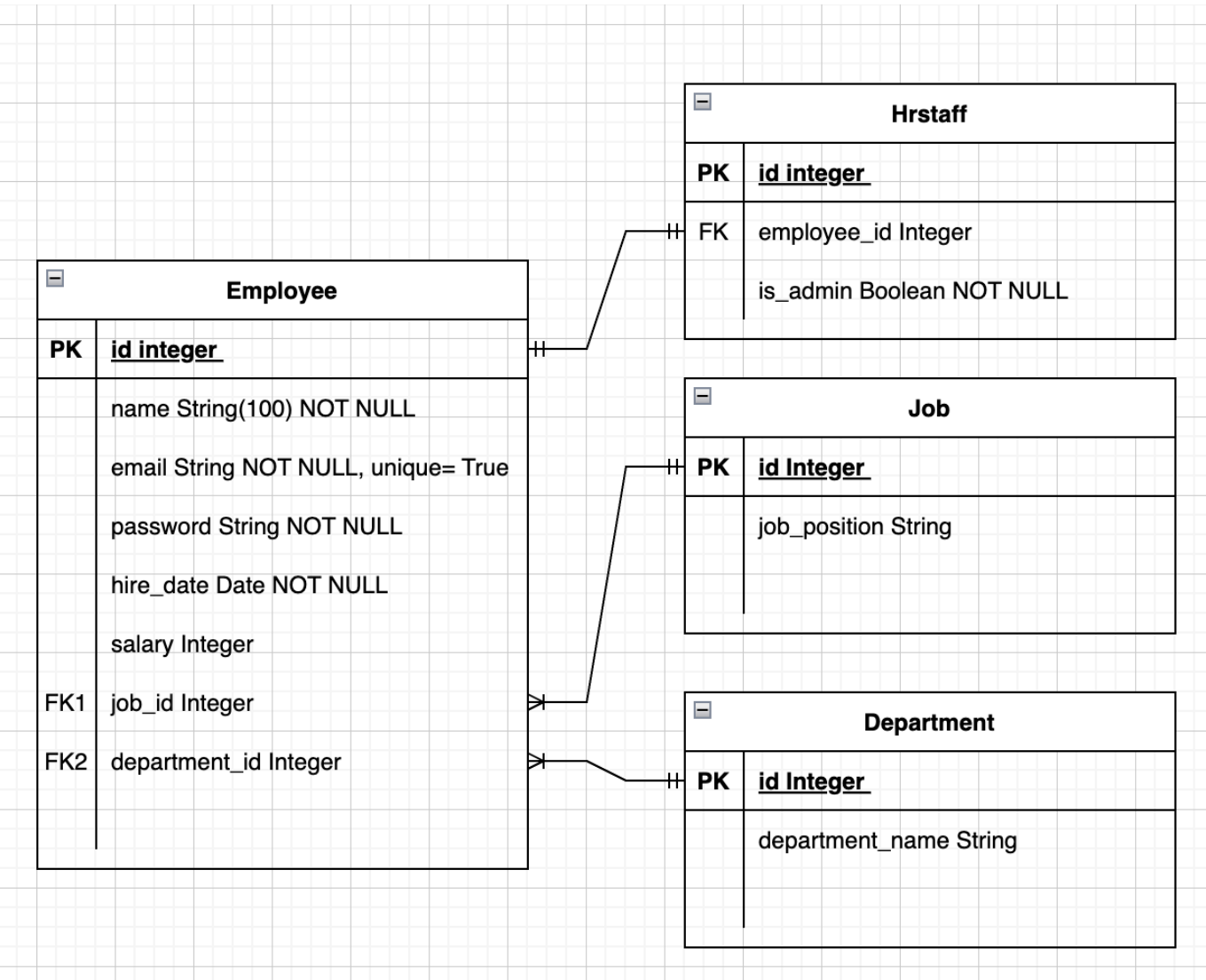


[Jobs] : [Employees] = 1 : N

The relationship between job entity and employee entity is also **1:N**. Because one job position has lots of employees. Similarly, jobs entity is the parent and employees entity becomes the child.



Logical Data Model



Physical Data Model

Reference: <https://www.lucidchart.com/pages/er-diagrams>

R7. Detail any third party services that your app will use

Flask_Bcrypt

Reference: <https://www.educba.com/flask-bcrypt/>

Flask-JWT-Extended

JWT stands for JSON Web Token. When a user logs in, how to keep login status has developed in various ways. The next step after cookies and sessions is the token-based user authentication method using JWT. It is a method of simply putting information that can identify user information in the token and using this JWT token when authentication is required. It consists of header, payload, and signature. Header and payload can be decoded and checked by anyone, so do not include information such as user's password. VerifySignature cannot be decrypted without knowing the secret key.

Structure of the JWT Token

eyJhbGciOiJIcXVjCj9.eyJzdWliOiBmNTMDIyYyQ.SflKxwRJSMeK6ydQssw5c

HEADER PAYLOAD VERIFY SIGNATURE

Process of JWT

- A Server receives a token and verify if signature is valid
- If it is considered valid, decode the claim set and open the data contained in the token
- It contains the expiration time, so check if the token is available and use it immediately if there is no problem

In my application, I used it to check whether user is an staff of HR team in the API that has features of 'update', 'delete', and 'insert',

Reference: <https://github.com/vimalloc/flask-jwt-extended>

Reference: <https://4geeks.com/lesson/what-is-JWT-and-how-to-implement-with-Flask>

Flask-marshmallow

Marshmallow is a utility that helps serialize, de-serialize, and validate Python objects. For example, you can use it to validate the payload of POST requests that come into the web server, or to convert Python objects to JSON to return them to response. Briefly, Deserialize input data to app-level objects. Serialize the objects to primitive Python types. The serialized objects can then be rendered to standard formats such as JSON for use in an HTTP API.

Reference: <https://github.com/marshmallow-code/marshmallow>

Flask-SQLAlchemy

SQLAlchemy is one of the ORMs available on Python. SQL had to be written to fetch the desired data from the database, but there is an ORM that replaces the SQL role in the application. In fact, you can connect to the database in the application and manage the data with only code without executing the query. In this way, SQLAlchemy interprets the Python class as a table in RDB and converts the language of SQL Alchemy similar to Python into SQL. It also makes connection of the database easier and automatically provides maintenance for the connection. In my application, models of each table and statements to manage DB's data in each API are used.

```
class Employee(db.Model):
    __tablename__ = 'employees'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String, nullable=False, unique=True)
    password = db.Column(db.String, nullable=False)
    hire_date = db.Column(db.Date)
    salary = db.Column(db.Integer)
```

Reference: <https://www.digitalocean.com/community/tutorials/how-to-use-flask-sqlalchemy-to-interact-with-databases-in-a-flask-application>

pip-review

pip-review is a python library that updates the currently installed pip list to the latest list. Run `pip-review` to display a list of packages with the latest version released. Then, run `pip-review -aC` to install the latest version of the currently installed pip list. As I learned in class, I check the latest version with `pip-review` before creating requirements.txt.

Reference: <https://pypi.org/project/pip-review/>

Reference: <https://stackabuse.com/python-update-all-packages-with-pip-review/>

psycopg2

Psycopg2 is a adapter that is used between PostgreSQL DB and flask application. In my applicationm, this is applied for connectiong postgresQL.

`DATABASE_URL=postgresql+psycopg2://{user ID}:{password}}@127.0.0.1:5432/t2a2_db`

python-dotenv

This Python module takes its configuration from environment variables. In other word, Rather than hardcoded sensitive information such as API key and db access information are put in the source code directly, it is stored separately in `.env` using an environment variable paired with key and value.

In my application, if you have a look at `main.py`

```
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL')
app.config['JWT_SECRET_KEY'] = os.environ.get('JWT_SECRET_KEY')
```

DB access information and set_key information are stored separately in `.env` file using an environment variable.

```
DATABASE_URL=postgresql+psycopg2://{user ID}:
{password}}@127.0.0.1:5432/t2a2_db
```

R8. Describe your projects models in terms of the relationships they have with each other

Based on logical ERD, we can generate physical ERD, which means we create Table on the database. Because SQLAlchemy is used in this application, each entity is modeled without actual SQL coding.

Employees model

The Employees table is a main table in the HR management system. So refer to this table in another table. belows are related tables.

```
class Employee(db.Model):
    __tablename__ = 'employees'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String, nullable=False, unique=True)
    password = db.Column(db.String, nullable=False)
    hire_date = db.Column(db.Date)
    salary = db.Column(db.Integer)

    job_id = db.Column(db.Integer, db.ForeignKey('jobs.id'),
        nullable=False)
    department_id = db.Column(db.Integer, db.ForeignKey('departments.id'),
        nullable=False)
```

Attributes

- ID: In the Employees table, the primary key is the **ID**. 'primary_key=True' was set As a constraint.
- Name: The employee's name is saved here. The constraint is **nullable=False** because the name should not be empty.
- Email: Email is put in this field. The constraints are 'nullable=False'.
- Job_id: The ID(PK) of the job table was set to foreign key. The constraint is **nullable=False**.
- Department_id: The ID(PK) of the job table was set to foreign key. The constraint is **nullable=False**.

Hrstaffs model

```
class Hrstaff(db.Model):
    __tablename__ = 'hrstaffs'

    id = db.Column(db.Integer, primary_key=True)
    employee_id = db.Column(db.Integer, db.ForeignKey('employees.id'),
        nullable=False)
    is_admin = db.Column(db.Boolean, default=False)
```

Attributes

- ID: The primary key is the **ID**. 'primary_key=True' is set As a constraint.

- employee_id: The ID(PK) of the employees table is set to foreign key. The constraint is `nullable=False`.
- is_admin: he boolean value was set to who is the admin of the HR management system. The constraint is `nullable=False`.

Departments model

```
class Department(db.Model):  
    __tablename__ = 'departments'  
  
    id = db.Column(db.Integer, primary_key=True)  
    department_name = db.Column(db.String)
```

Attributes

- ID: The primary key is the `ID`. 'primary_key=True' is set As a constraint.
- department_name: This attribute is filled with department name.

Jobs model

```
class Job(db.Model):  
    __tablename__ = 'jobs'  
  
    id = db.Column(db.Integer, primary_key=True)  
    job_position = db.Column(db.String)
```

Attributes

- ID: The primary key is the `ID`. 'primary_key=True' is set As a constraint.
- job_position: This attribute is filled with job position.

Relationship

Departments - Employees

All employees have a `department_id` on the employees table to indicate which department they are currently working in. The `ID`, which is the primary key of the parent departments, was set as the foreign key in the employees table.

The relationship with Departments is **one to many relationship**. Therefore, employees' parents is the Department table. It is defined in the models as follows.

Declaration of department relationship in Employees model

```
department = db.relationship('Department', back_populates='employees')
```

Declaration of employees relationship in Departments model

```
employees = db.relationship('Employee', back_populates='department',  
cascade='all, delete')
```

- **.relationship()** method defines the relationship between two tables. The relationship between the **departments** table and the **employees** table is each of parents and children, and this method is written in the both model.
- **back_populates**, unlike **backref**, **back_populates** must specify the relationship between both parent and child models, and specify the name of the table that is related. Then **SQLAlchemy** recognises the relationship between both with foreign key.
- **cascade='all, delete'** Once deleting an instance also automatically deletes the matching instance of the table in which it is related. It is used it in the departments table

Jobs - Employees

All employees have their current job positions. So to represent this, **job_id** is put in the employees table. The **ID**, which is the primary key of the parent jobs, was set to the employees table as the foreign key. The relationship between both tables is **one to many relationship** as well. Jobs table should be parents and employees has child relation.

Declaration of job relationship in Employees model

```
job = db.relationship('Job', back_populates='employees')
```

Declaration of employees relationship in jobs model

```
employees = db.relationship('Employee', back_populates='job',  
cascade='all, delete')
```

The parameters in .relationship method is explained the above.

Hrstaff - Employess

The relationship hrstaffs with employees models is **one-to-one relationships**, **employee_id** is put in the hrstaffs table. The **ID**, which is the primary key of the parent employees, is set to the hrstaffs table as the foreign key.

The relationship between both tables is **one to one relationship**.

Declaration of employees relationship in hrstaffs model

```
employees = db.relationship('Employee', back_populates='hrstaffs',
                             use_list=False, cascade='all, delete')
```

Declaration of hrstaffs relationship in employees model

```
hrstaffs = db.relationship('Hrstaff', back_populates='employees',
                             use_list=False, cascade='all, delete')
```

- **use_list=False** indicate one-to-one relationship

Schema

If a FrontEnd team creates a web page and try to connect to my application to take data they want, each endpoints in my application must return the data to JSON format. Schema defines the part that support this return data.

Employee Schema

The return field defined in the employees model using **marshmallow** is as follows.

```
class Meta:
    fields = ('id', 'name', 'email', 'password', 'salary', 'hire_date',
              'job_id', 'department_id', 'job', 'department')
    ordered = True
```

Among the fields, job and department represent the contents of **job_id** and **department_id** referred to as **foreign key**. Displaying job_id and department_id as code in a way causes users have to query it again to check the values. So you can add this part to the output field to help you understand by displaying its content of each ID. To do this, the following statements were added to the employee schema.

```
job = fields.Nested('JobSchema')
department = fields.Nested('DepartmentSchema')
```

As defined in the `job schema` and `department schema`, the values are returned in the `Nest` format as follows to make frontend team work easier.

```
"department": {
  "department_name": "Human Resources",
  "id": 1
},
"department_id": 1,
```

Hrstaffs Schema

In the Hrstaffs schema, as in the employee schema above, the corresponding information is shown by adding a field called `employees`.

```
employees = fields.Nested('EmployeeSchema', exclude=['password'])
class Meta:
    fields = ('id', 'is_admin', 'employees', 'employee_id', 'email')
```

In the employees model, the `password` must be excluded from the output field.

Department Schema

The department schema constitutes an output field just like the model information.

```
class Meta:
    fields = ('id', 'department_name')
```

Job Schema

The job schema also has the same fields of the model as below.

```
class Meta:
    fields = ('id', 'job_position')
```

R9. Discuss the database relations to be implemented in your application

If you look at how the model above is actually implemented, it will look like the below.

Employees table in database

```
t2a2_db=# \d employees;

               Table "public.employees"
   Column      |      Type      | Collation | Nullable |      Default
-----+-----+-----+-----+-----
 id            | integer        |           | not null | nextval('employees_id_seq'::regclass)
 name         | character varying(100) |           | not null |
 email        | character varying |           | not null |
 password     | character varying |           | not null |
 hire_date    | date           |           |          |
 salary       | integer        |           |          |
 job_id       | integer        |           | not null |
 department_id| integer        |           | not null |
Indexes:
    "employees_pkey" PRIMARY KEY, btree (id)
    "employees_email_key" UNIQUE CONSTRAINT, btree (email)
Foreign-key constraints:
    "employees_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(id)
    "employees_job_id_fkey" FOREIGN KEY (job_id) REFERENCES jobs(id)
Referenced by:
    TABLE "hrstaffs" CONSTRAINT "hrstaffs_employee_id_fkey" FOREIGN KEY (employee_id) REFERENCES employees(id)
```

The table is well implemented in SQLAlchemy as we define it. In the employees table, the id is the primary key, and the `department_id` of the department table and the `job_id` of the job table refer to each table as the foreign key. It also refers to the ID where `employee_id`, which is the foreign key of the hrstaffs table, is the primary key of the employees table.

Hrstaffs table in database

```
t2a2_db=# \d hrstaffs

               Table "public.hrstaffs"
   Column      |      Type      | Collation | Nullable |      Default
-----+-----+-----+-----+-----
 id            | integer        |           | not null | nextval('hrstaffs_id_seq'::regclass)
 employee_id   | integer        |           | not null |
 is_admin     | boolean        |           |          |
Indexes:
    "hrstaffs_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "hrstaffs_employee_id_fkey" FOREIGN KEY (employee_id) REFERENCES employees(id)
```

This table refers to the ID that is primary key. The foreign key, employee_id is referenced by the primary key of the employee table.

Departments table in database

```
t2a2_db=# \d departments

               Table "public.departments"
   Column      |      Type      | Collation | Nullable |      Default
-----+-----+-----+-----+-----
 id            | integer        |           | not null | nextval('departments_id_seq'::regclass)
 department_name| character varying |           |          |
Indexes:
    "departments_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "employees" CONSTRAINT "employees_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(id)
```

ID is set to primary key. Employee_id is referenced by departments whose ID is foreign key in the employees table. It is said that the departments table is parents and the employees table is child.

Jobs table in database

```
t2a2_db=# \d jobs
```

Column	Type	Table "public.jobs"	Collation	Nullable	Default
id	integer			not null	nextval('jobs_id_seq'::regclass)
job_position	character varying				

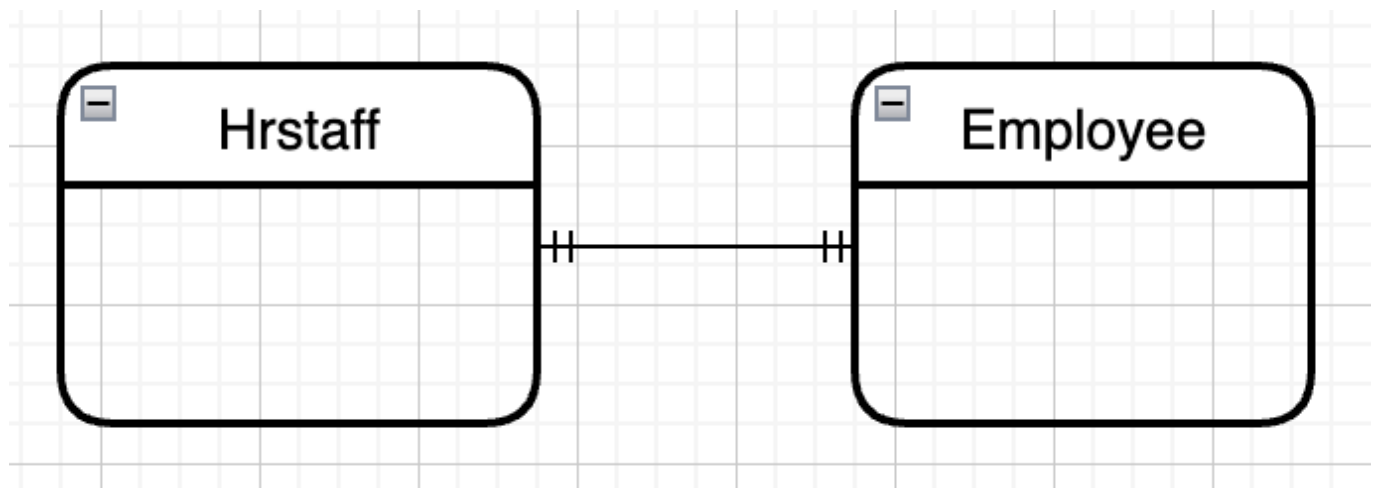
Indexes:
 "jobs_pkey" PRIMARY KEY, btree (id)

Referenced by:
 TABLE "employees" CONSTRAINT "employees_job_id_fkey" FOREIGN KEY (job_id) REFERENCES jobs(id)

ID is primary key in this table. The foreign key, job_id is referenced by the primary key of the jobs table. It refers to relationships that the jobs table is parents and the employees table is child.

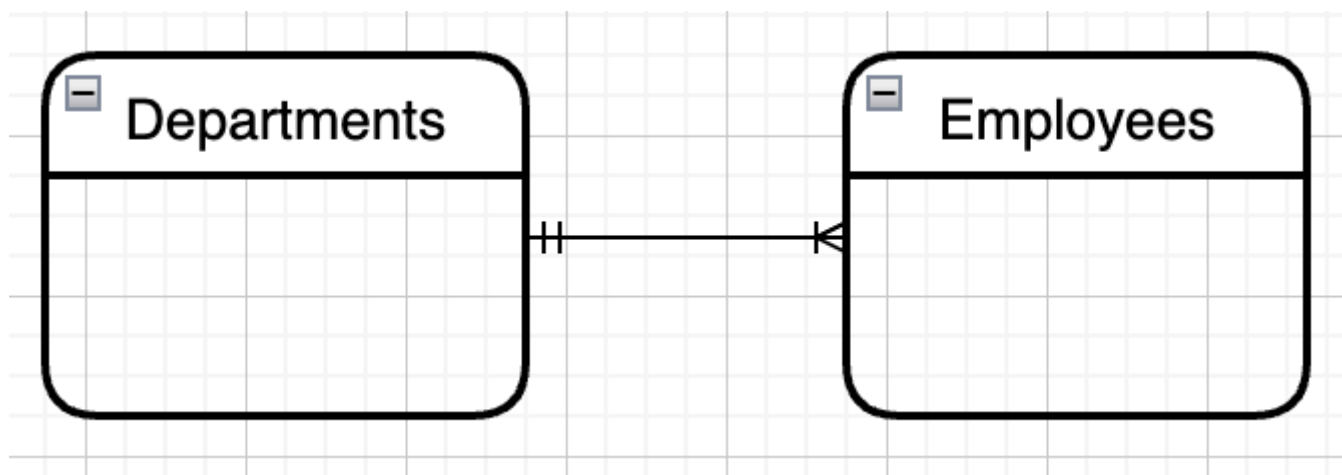
Relationship

Hrstaffs : Employees = one to one relationship



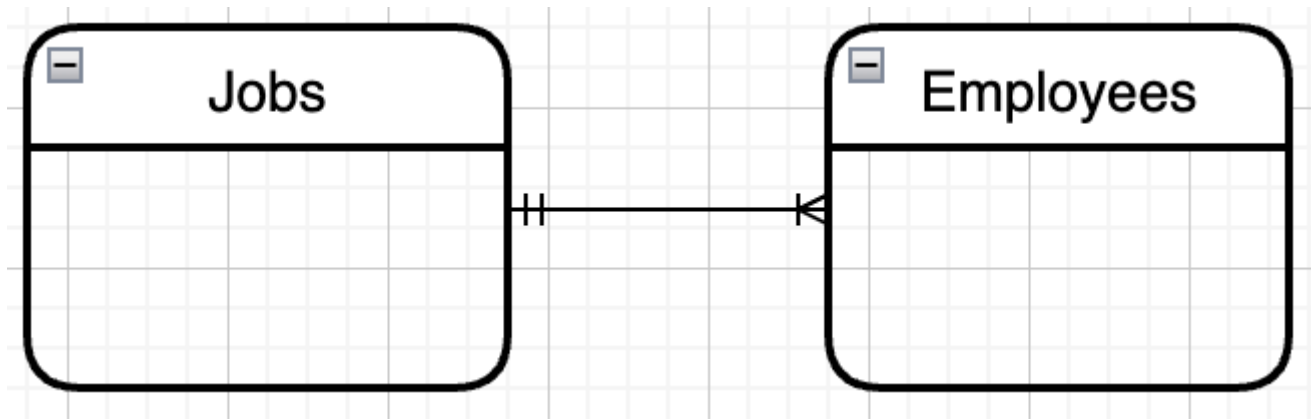
one Employees must have one Hrstaff. It's not a option but a mandatory. The reason for this relationship was explained above.

Departments : Employees = one to many relationship



one Departments must have one or more Employps.

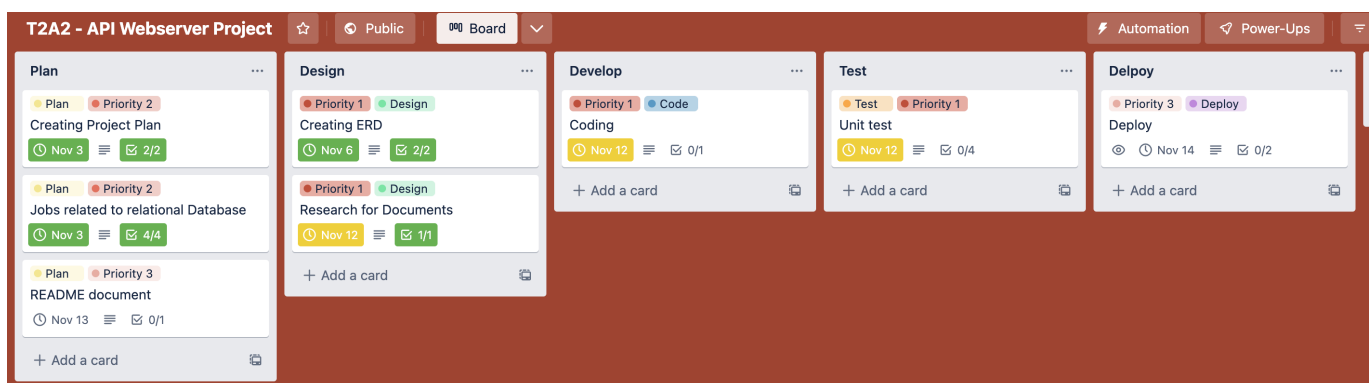
Jobs : Jobs = one to many relationship



one Jobs must have one or more Employees.

R10. Describe the way tasks are allocated and tracked in your project.

I used Agile methodology to create my application during the whole project. The processes of project consist of totally 5 parts such as **PLAN – DESIGN – DEVELOP – TEST – DEPLOY**. For the tasks of each stage, priority is set to determine what needs to be handled first. A checklist was placed for each to-do, and parts that should not be missed during the step. I proceeded with the project for each endpoint in the development and test stage. The API made in the design was developed and tested, and if an error occurs, retry developing, so the steps of development and test were repeated. Each project stage was created according to the agile methodology as follows.



Trello for Webserver Project

PLAN

It is the stage of establishing an overall schedule and plan for the project. For each step, decide on the due date and things to do, and plan to finish the project without any issues. The PLAN consists of **Creating Project Plan**, **Jobs related to relational Database**, and **README documentation**.

Creating Project Plan

Create the project plan and schedule in Trello according to Agile methodology.

- Create the due data and check list of each step in the project.
- Plan and schedule and prioritize.

Jobs related to relational database

- Analyze requirements for implementing ERD and application
- Accurate theoretical acquisition to construct db and table in practice.
- Practice to implement sqlalchemy and flask, etc
- Implement CRUD

README documentation

- Creating README document that includes requirements of this project on Canvas.



Design

Program and database design for implementing applications based on requirement analysis. All of these are top priority tasks.

Design ERD

To implement the database, define relationship between entities. Create Primary key and foreign key to make relationship between entities.

Design for the application

Establish the function of end points and relationship of database to be implemented.

Creating ERD

in list [Design](#)

Labels

Priority 1

Design

+

Due date

☒

Nov 6 at 11:12 PM

complete

▼

≡

Description

Edit

To represent the database, ERD must be created.
ERD should be represented the relationship between entities.

☒ Checklist

Hide checked items

Delete

100%

☒ clarify data types for each attributes

☒ Set up primary keys and foreign keys

☒ Define relationship between entities

Design for the application

in list [Design](#)

Labels

Priority 1

Design

+

Due date

☐

yesterday at 11:12 PM

overdue

▼

≡

Description

Edit

Design for API endpoints and DB

API endpoints

- Design for implementing source code
- Creating Program list
- testcase for unit test

Database

- Creating Table list
- Define Model

☒ Checklist

Hide checked items

Delete

100%

☒ Creating Program list


☒ Creating Table list

☒ Creating testcase for unit test

Develop

Coding

Implement db and programs based on design. Conduct functional testing during development. Following up the assignment requirement when coding.

 **Coding**
in list [Develop](#)

Labels


● Priority 1

● Code

+

Due date

☐ yesterday at 11:12 PM **overdue** ▼

 **Description** [Edit](#)

Code

- Implement appropriate functionalities as planned
- Use mode methods to query the database
- Catch errors and handle them for good user experience
- Return precise code and message for error
- Implement functions or methods to validate data
- Make sure D.R.Y
- Need to comment about all queries

☒ **Checklist**

[Hide checked items](#) [Delete](#)

100%

☒ ~~represent all condition above~~

[Add an item](#)

Test

Testing

Functional tests and unit tests are performed frequently during development according to the agile methodology, and when an error is found, the development and test stages are repeated until the development is completed by correcting the error part and performing the test. Perform the unit test according to the test case and verify that the function is accurately implemented.

Unit test

in list

Test

Labels

Test

Priority 1

+

Due date

✓

yesterday at 11:12 PM

complete

▼

☰

Description

Edit

Test

Perform Unit Test

• Ensure functionality is working properly as you planed

☑

Checklist

Hide checked items

Delete

100%

☑

Endpoints works well according to the design

☑

Return value and messages returned as you intended

☑

Each table is updated according to CRUD

☑

Data validation is working well

Deploy

Deployment

Ensure all requirements are carried out as planned.

25 / 26

Deploy

in list [Delpoy](#)

Labels

Priority 3

Deploy

+

Due date

☐ tomorrow at 11:12 PM

Description

Edit

Deploy

- Satisfy all conditions when submitting deliverables on Cavas

☒

Checklist

0%

☐ Ensure naming rule of each file

☐ Ensure making a single zip file including all deliverables

Add an item

Delete

26 / 26