7PAM2005-0901-2023

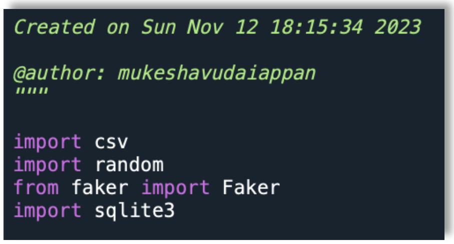
Data Mining & Discovery

SQL Assignment

Student Name : Mukesh Avudaiappan
Student ID : 22024161
GitHub Link : https://github.com/MukeshAvudaiappan/Data-Mining-Discovery

A. How you generated the data ?

I generated the data using a Python script that utilizes the Faker module and random routines to create artificial data for an employee database. This is how the information is produced,



```
Created on Sun Nov 12 18:15:34 2023

@author: mukeshavudaiappan
"""

import csv
import random
from faker import Faker
import sqlite3
```

-------------------------------<u>Employee Data Generation</u>------------------------------

1. ***Ordinal Data** (Roles):*
- Function: `generate_ordinal_data()`
- Roles: "Manager," "Developer," "Analyst," "Designer," "HR"

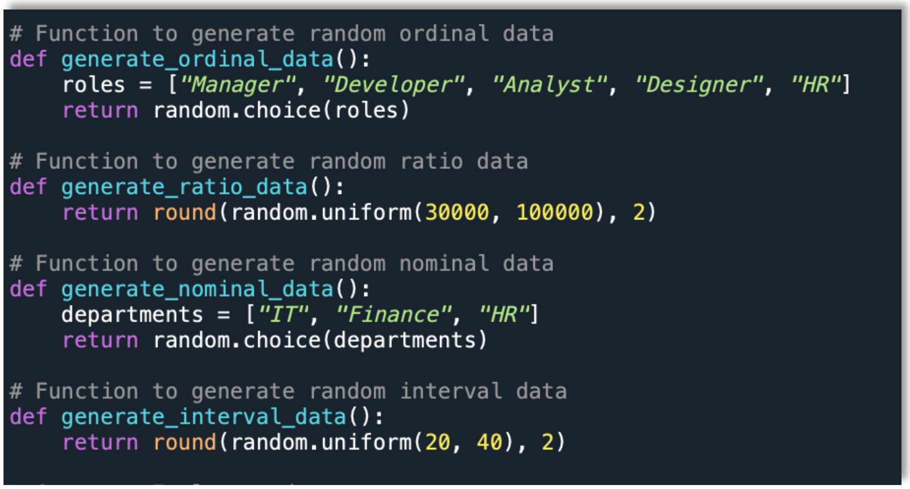
2. ***Ratio Data** (Salary):*
- Function: `generate_ratio_data()`
- Salary: Random float between 30,000 and 100,000, rounded to 2 decimal places.

3. ***Nominal Data** (Departments):*
- Function: `generate_nominal_data()`
- Departments: "IT," "Finance," "HR"

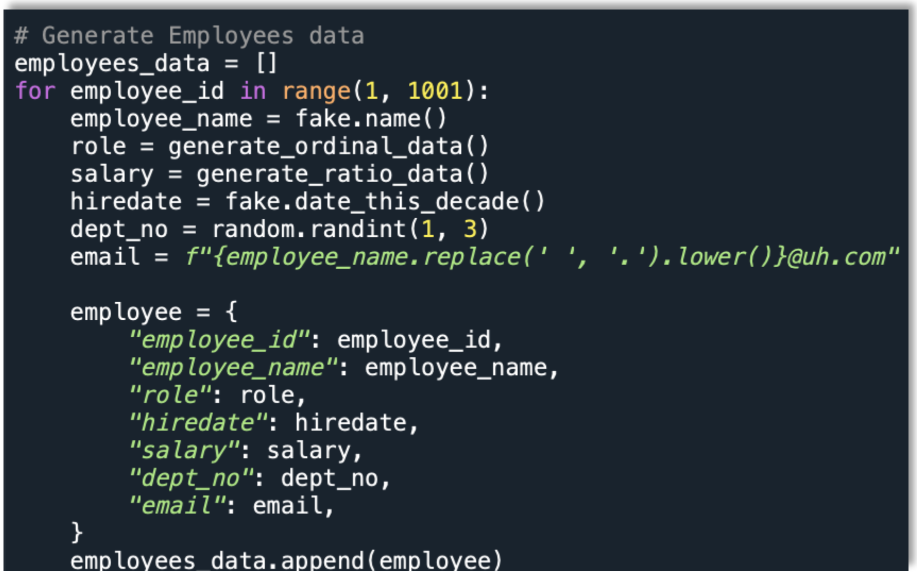
4. ***Interval Data**:*
- Function: `generate_interval_data()`
- Interval: Random float between 20 and 40, rounded to 2 decimal places

5. ***Email Generation**:*
- Custom email is generated using the employee's name in a specific format.

6. ***Employee Data Structure:***
- Fields: `employee_id`, `employee_name`, `role`, `hiredate`, `salary`, `dept_no`, `email`



```python
# Function to generate random ordinal data
def generate_ordinal_data():
    roles = ["Manager", "Developer", "Analyst", "Designer", "HR"]
    return random.choice(roles)

# Function to generate random ratio data
def generate_ratio_data():
    return round(random.uniform(30000, 100000), 2)

# Function to generate random nominal data
def generate_nominal_data():
    departments = ["IT", "Finance", "HR"]
    return random.choice(departments)

# Function to generate random interval data
def generate_interval_data():
    return round(random.uniform(20, 40), 2)
```

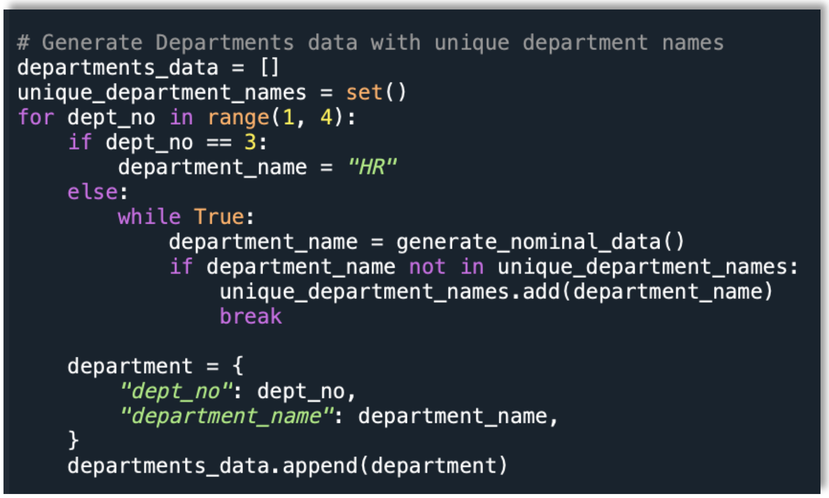


```python
# Generate Employees data
employees_data = []
for employee_id in range(1, 1001):
    employee_name = fake.name()
    role = generate_ordinal_data()
    salary = generate_ratio_data()
    hiredate = fake.date_this_decade()
    dept_no = random.randint(1, 3)
    email = f"{employee_name.replace(' ', '.').lower()}@uh.com"

    employee = {
        "employee_id": employee_id,
        "employee_name": employee_name,
        "role": role,
        "hiredate": hiredate,
        "salary": salary,
        "dept_no": dept_no,
        "email": email,
    }
    employees_data.append(employee)
```

--------------------------------**Department Data Generation**--------------------------

1. *Unique Department Names:*
   - Ensures that each department has a unique name.
   - The "HR" department is manually assigned to `dept_no` 3.

2. *Department Data Structure:*
   - Fields: `dept_no`, `department_name`

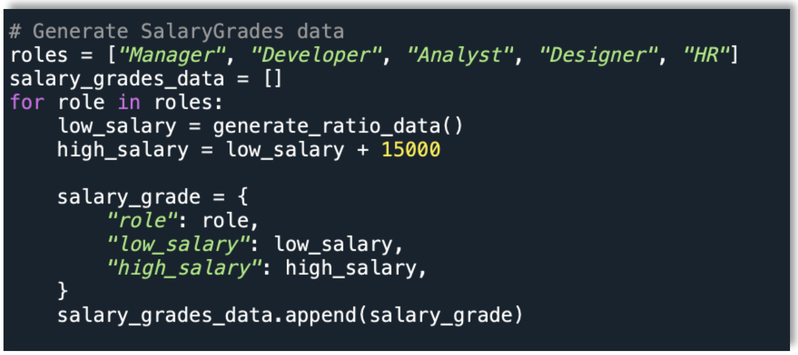


```python
# Generate Departments data with unique department names
departments_data = []
unique_department_names = set()
for dept_no in range(1, 4):
    if dept_no == 3:
        department_name = "HR"
    else:
        while True:
            department_name = generate_nominal_data()
            if department_name not in unique_department_names:
                unique_department_names.add(department_name)
                break

    department = {
        "dept_no": dept_no,
        "department_name": department_name,
    }
    departments_data.append(department)
```

------------------------------Salary Grades Data Generation-------------------------

1. *Salary Grade Ranges:*
   - Salary grades are defined for each role, with a low salary and a high salary.
   - The high salary is calculated as the low salary plus 15,000.

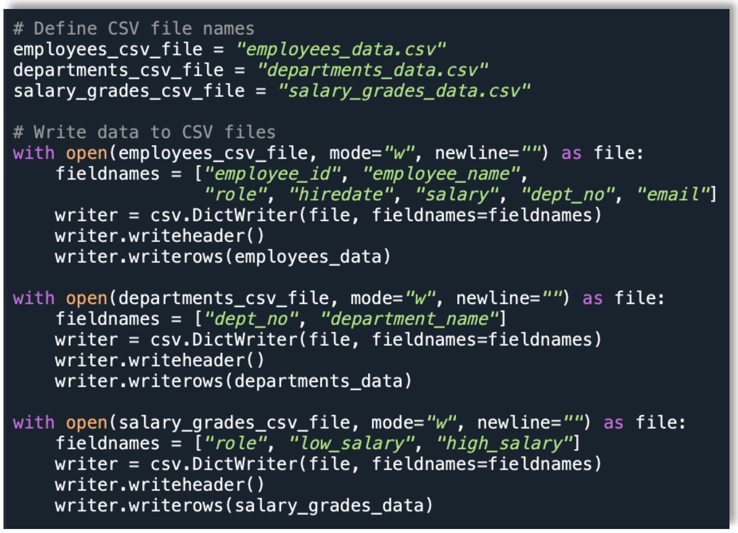
2. *Salary Grades Data Structure:*
   - Fields: `role`, `low_salary`, `high_salary`



```python
# Generate SalaryGrades data
roles = ["Manager", "Developer", "Analyst", "Designer", "HR"]
salary_grades_data = []
for role in roles:
    low_salary = generate_ratio_data()
    high_salary = low_salary + 15000

    salary_grade = {
        "role": role,
        "low_salary": low_salary,
        "high_salary": high_salary,
    }
    salary_grades_data.append(salary_grade)
```

Creation of CSV files:

- The "employees_data.csv," "departments_data.csv," and "salary_grades_data.csv" CSV files are produced.
- The `csv.DictWriter` module is used to write data to these files.

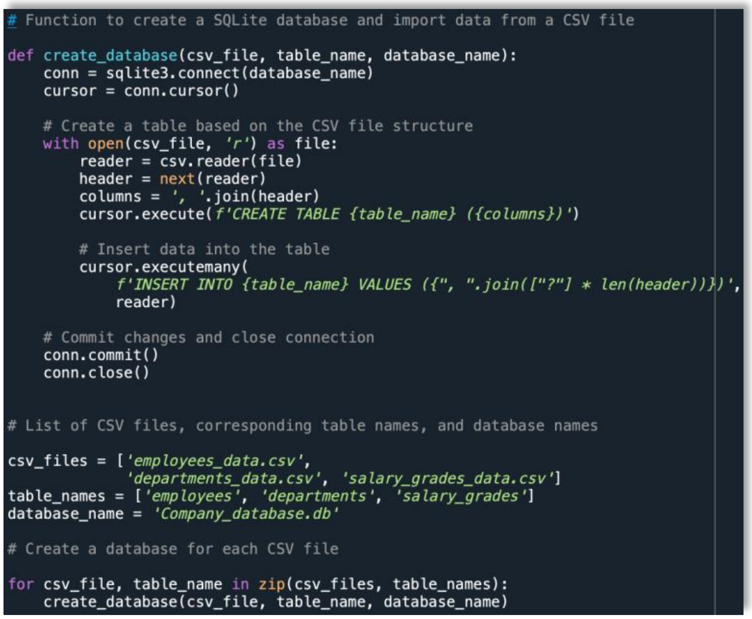


```
# Define CSV file names
employees_csv_file = "employees_data.csv"
departments_csv_file = "departments_data.csv"
salary_grades_csv_file = "salary_grades_data.csv"

# Write data to CSV files
with open(employees_csv_file, mode="w", newline="") as file:
    fieldnames = ["employee_id", "employee_name",
                  "role", "hiredate", "salary", "dept_no", "email"]
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(employees_data)

with open(departments_csv_file, mode="w", newline="") as file:
    fieldnames = ["dept_no", "department_name"]
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(departments_data)

with open(salary_grades_csv_file, mode="w", newline="") as file:
    fieldnames = ["role", "low_salary", "high_salary"]
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(salary_grades_data)
```

Creation of a SQLite Database:

The creation of a SQLite database with the name "Company_database.db".

- A function called `create_database` is defined in the script to create a table and import data into it using the format of the CSV file.
- In order to construct tables for personnel, departments, and compensation grades, the function is run for each CSV file.

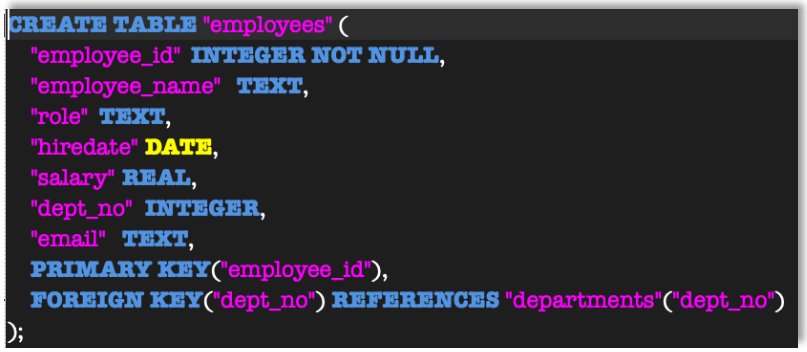


```
# Function to create a SQLite database and import data from a CSV file

def create_database(csv_file, table_name, database_name):
    conn = sqlite3.connect(database_name)
    cursor = conn.cursor()

    # Create a table based on the CSV file structure
    with open(csv_file, 'r') as file:
        reader = csv.reader(file)
        header = next(reader)
        columns = ', '.join(header)
        cursor.execute(f'CREATE TABLE {table_name} ({columns})')

        # Insert data into the table
        cursor.executemany(
            f'INSERT INTO {table_name} VALUES ({", ".join(["?"] * len(header))})',
            reader)

    # Commit changes and close connection
    conn.commit()
    conn.close()


# List of CSV files, corresponding table names, and database names

csv_files = ['employees_data.csv',
             'departments_data.csv', 'salary_grades_data.csv']
table_names = ['employees', 'departments', 'salary_grades']
database_name = 'Company_database.db'

# Create a database for each CSV file

for csv_file, table_name in zip(csv_files, table_names):
    create_database(csv_file, table_name, database_name)
```

B. The schema of your database-

The below SQL script defines the schema for a relational database with three tables: employees, departments & salary_grades. Here's an explanation of each table's structure:

1.Table: employees

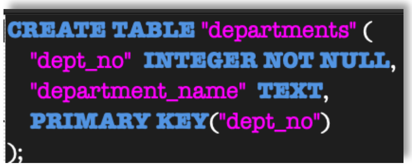


```
CREATE TABLE "employees" (
  "employee_id" INTEGER NOT NULL,
  "employee_name"  TEXT,
  "role" TEXT,
  "hiredate" DATE,
  "salary" REAL,
  "dept_no"  INTEGER,
  "email"  TEXT,
  PRIMARY KEY("employee_id"),
  FOREIGN KEY("dept_no") REFERENCES "departments"("dept_no")
);
```

Columns:

- employee_id (INTEGER, NOT NULL): Primary key representing the unique identifier for each employee.
- employee_name (TEXT): Stores the name of the employee.
- role (TEXT): Represents the role of the employee (e.g., Manager, Developer).
- hiredate (DATE): Records the date when the employee was hired.
- salary (REAL): Stores the salary of the employee as a floating-point number.
- dept_no (INTEGER): Foreign key referencing the dept_no column in the departments table. Establishes a relationship between employees and departments.
- email (TEXT): Stores the email address of the employee.
- Constraints:
- PRIMARY KEY("employee_id"): Defines employee_id as the primary key for the table, ensuring its uniqueness.
- FOREIGN KEY("dept_no") REFERENCES "departments"("dept_no"): Establishes a foreign key relationship with the departments table, linking the dept_no column in employees to the dept_no column in departments.

2.Table: departments

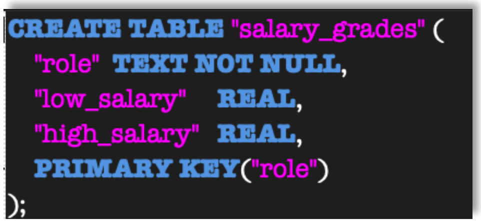


```
CREATE TABLE "departments" (
  "dept_no"  INTEGER NOT NULL,
  "department_name"  TEXT,
  PRIMARY KEY("dept_no")
);
```

Columns:

- dept_no (INTEGER, NOT NULL): Primary key representing the unique identifier for each department.
- department_name (TEXT): Stores the name of the department.
- Constraints:
- PRIMARY KEY("dept_no"): Defines dept_no as the primary key for the table, ensuring its uniqueness.

3.Table: salary_grades



```
CREATE TABLE "salary_grades" (
  "role"  TEXT NOT NULL,
  "low_salary"    REAL,
  "high_salary"   REAL,
  PRIMARY KEY("role")
);
```

Columns:

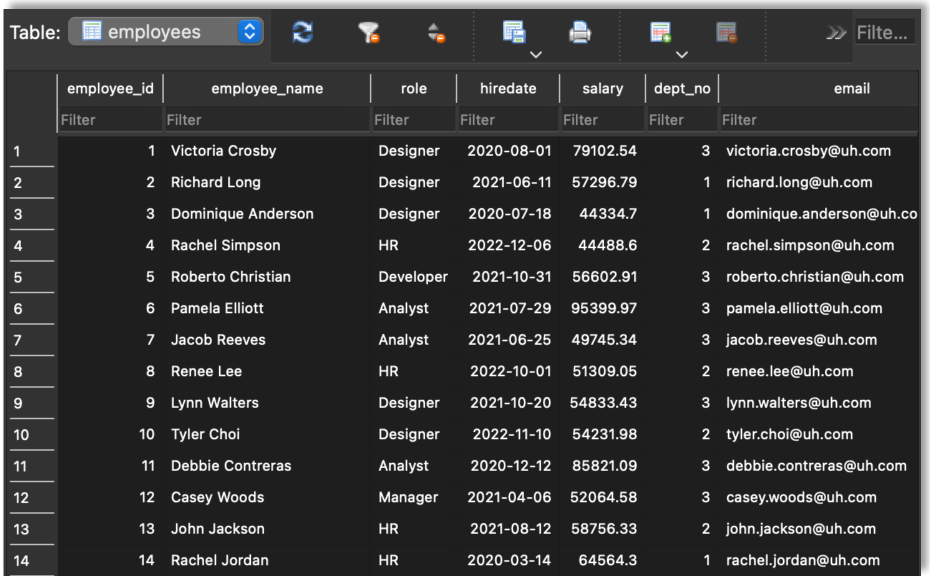
- role (TEXT, NOT NULL): Primary key representing the role (e.g., Manager, Developer).
- low_salary (REAL): Represents the lower bound of the salary range for a given role.
- high_salary (REAL): Represents the higher bound of the salary range for a given role.
- Constraints:
- PRIMARY KEY("role"): Defines role as the primary key for the table, ensuring its uniqueness.

C. Your justification for any separate tables and ethical discussion.

The tables in the given database schema are divided in accordance with database normalization principles, which aim to reduce dependency and redundancy and enhance data integrity. The following explains the reasons behind the three tables and addresses the moral issues surrounding database design:

1. **<u>Separations of Tables –</u>**

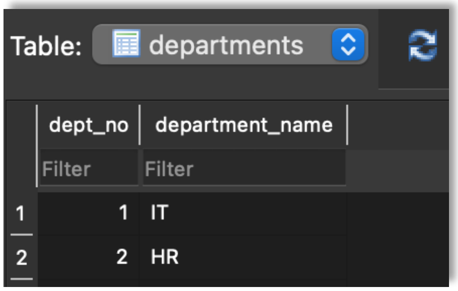
Employees Table (employees):



	employee_id	employee_name	role	hiredate	salary	dept_no	email
1	1	Victoria Crosby	Designer	2020-08-01	79102.54	3	victoria.crosby@uh.com
2	2	Richard Long	Designer	2021-06-11	57296.79	1	richard.long@uh.com
3	3	Dominique Anderson	Designer	2020-07-18	44334.7	1	dominique.anderson@uh.co
4	4	Rachel Simpson	HR	2022-12-06	44488.6	2	rachel.simpson@uh.com
5	5	Roberto Christian	Developer	2021-10-31	56602.91	3	roberto.christian@uh.com
6	6	Pamela Elliott	Analyst	2021-07-29	95399.97	3	pamela.elliott@uh.com
7	7	Jacob Reeves	Analyst	2021-06-25	49745.34	3	jacob.reeves@uh.com
8	8	Renee Lee	HR	2022-10-01	51309.05	2	renee.lee@uh.com
9	9	Lynn Walters	Designer	2021-10-20	54833.43	3	lynn.walters@uh.com
10	10	Tyler Choi	Designer	2022-11-10	54231.98	2	tyler.choi@uh.com
11	11	Debbie Contreras	Analyst	2020-12-12	85821.09	3	debbie.contreras@uh.com
12	12	Casey Woods	Manager	2021-04-06	52064.58	3	casey.woods@uh.com
13	13	John Jackson	HR	2021-08-12	58756.33	2	john.jackson@uh.com
14	14	Rachel Jordan	HR	2020-03-14	64564.3	1	rachel.jordan@uh.com

This table contains personal data about each employee, such as their email address, department number, name, role, and date of hiring. I make sure that every employee is individually recognized by an employee ID by segregating this data into a separate table. Furthermore, a connection is established between employees and the departments to which they are assigned through the foreign key relationship with the departments table.

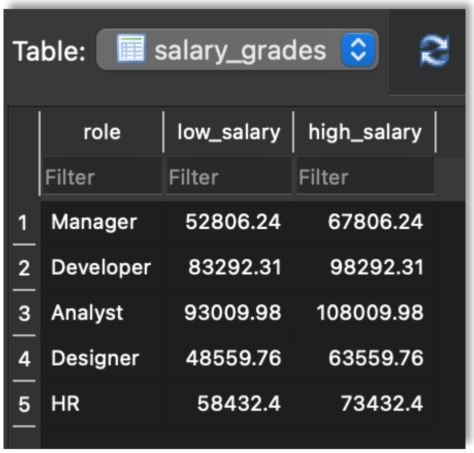
Departments Table (departments):



	dept_no	department_name
1	1	IT
2	2	HR

This table contains information on departments, including names and unique department numbers. Maintaining data consistency and preventing duplication are achieved by segregating department-specific information into a separate table.

Salary Grades Table (salary_grades):



	role	low_salary	high_salary
1	Manager	52806.24	67806.24
2	Developer	83292.31	98292.31
3	Analyst	93009.98	108009.98
4	Designer	48559.76	63559.76
5	HR	58432.4	73432.4

This table contains salary ranges for different roles. By isolating salary-related information, the design avoids duplicating salary data for each employee, ensuring that updates to salary grades only need to be made in one place.

2. **Justification –**

Normalization: The separation of tables adheres to the principles of normalization, specifically aiming for the third normal form (3NF). Each table has a clear primary key, and data is organized to minimize redundancy and dependency.

Foreign Key Relationships: The use of foreign keys, such as the dept_no in the employees table referencing the dept_no in the departments table, establishes relationships between entities, promoting data integrity and consistency

3. Ethical Considerations:

Data Privacy: Ensuring the separation of tables helps maintain data privacy by storing sensitive information (e.g., salary, email) in a structured and controlled manner. Access controls can be implemented to restrict unauthorized access to certain data.

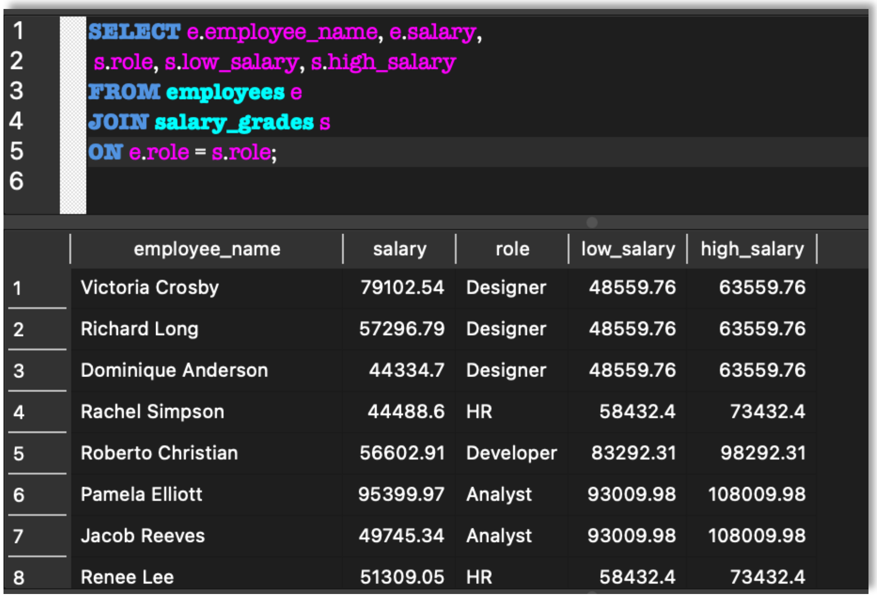
Data Accuracy: The database design supports data accuracy by avoiding redundancy and ensuring that updates or modifications are made in one place. This contributes to fair and accurate representation of information.

Consistency and Fairness: Separating tables based on entities promotes consistency in data representation, preventing inconsistencies that could lead to unfair treatment of employees or misrepresentation of organizational structure.

Security: The ethical design of the database considers security measures to protect against unauthorized access and potential misuse of sensitive information. Implementing security measures aligns with ethical standards to safeguard data.

D. Example queries of your database including joins and selections, demonstrating different data types.

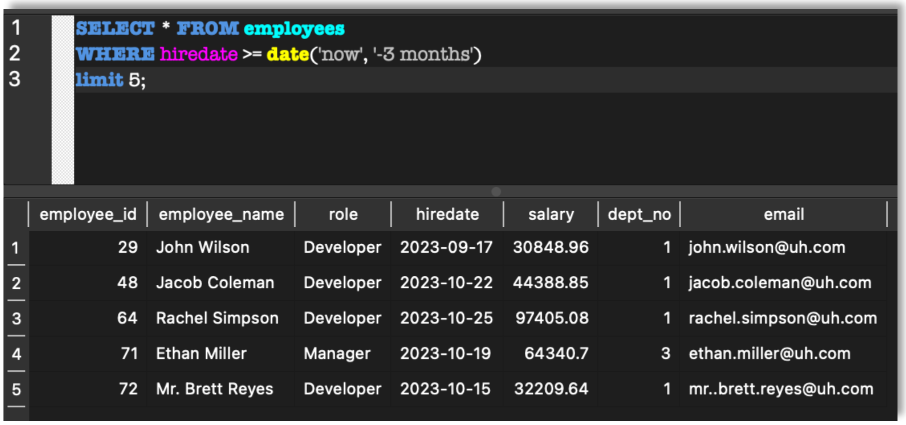
1. Select employees and their salary grades:



```sql
SELECT e.employee_name, e.salary,
 s.role, s.low_salary, s.high_salary
FROM employees e
JOIN salary_grades s
ON e.role = s.role;
```

	employee_name	salary	role	low_salary	high_salary
1	Victoria Crosby	79102.54	Designer	48559.76	63559.76
2	Richard Long	57296.79	Designer	48559.76	63559.76
3	Dominique Anderson	44334.7	Designer	48559.76	63559.76
4	Rachel Simpson	44488.6	HR	58432.4	73432.4
5	Roberto Christian	56602.91	Developer	83292.31	98292.31
6	Pamela Elliott	95399.97	Analyst	93009.98	108009.98
7	Jacob Reeves	49745.34	Analyst	93009.98	108009.98
8	Renee Lee	51309.05	HR	58432.4	73432.4

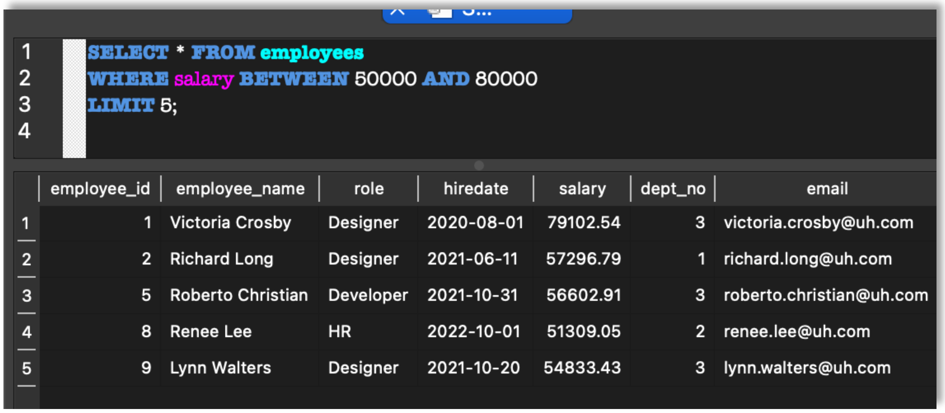
2. Select employees hired in the last three months:



```sql
SELECT * FROM employees
WHERE hiredate >= date('now', '-3 months')
limit 5;
```

employee_id	employee_name	role	hiredate	salary	dept_no	email
29	John Wilson	Developer	2023-09-17	30848.96	1	john.wilson@uh.com
48	Jacob Coleman	Developer	2023-10-22	44388.85	1	jacob.coleman@uh.com
64	Rachel Simpson	Developer	2023-10-25	97405.08	1	rachel.simpson@uh.com
71	Ethan Miller	Manager	2023-10-19	64340.7	3	ethan.miller@uh.com
72	Mr. Brett Reyes	Developer	2023-10-15	32209.64	1	mr..brett.reyes@uh.com

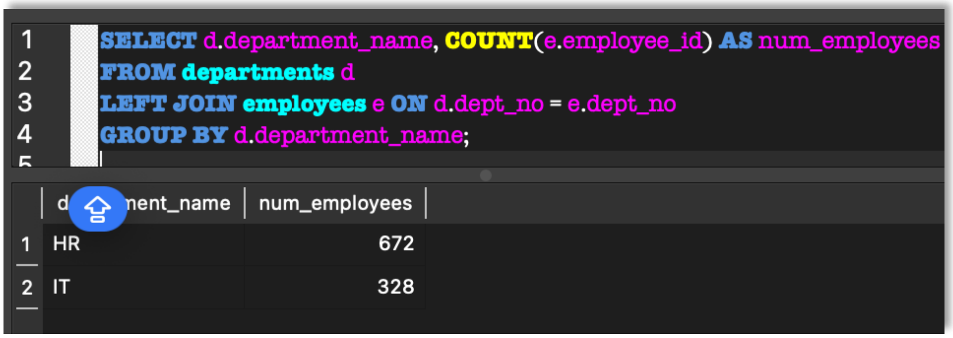
3. Select employees with salaries within a specific range:



```sql
SELECT * FROM employees
WHERE salary BETWEEN 50000 AND 80000
LIMIT 5;
```

employee_id	employee_name	role	hiredate	salary	dept_no	email
1	Victoria Crosby	Designer	2020-08-01	79102.54	3	victoria.crosby@uh.com
2	Richard Long	Designer	2021-06-11	57296.79	1	richard.long@uh.com
5	Roberto Christian	Developer	2021-10-31	56602.91	3	roberto.christian@uh.com
8	Renee Lee	HR	2022-10-01	51309.05	2	renee.lee@uh.com
9	Lynn Walters	Designer	2021-10-20	54833.43	3	lynn.walters@uh.com

4. Select departments and the number of employees in each department:



```sql
SELECT d.department_name, COUNT(e.employee_id) AS num_employees
FROM departments d
LEFT JOIN employees e ON d.dept_no = e.dept_no
GROUP BY d.department_name;
```

d nent_name	num_employees
HR	672
IT	328