Assignment 2: Database Normalization

Sarah Gillard

CS303: Database Management

February 29, 2024

Project Overview

Part 2 of the CS303 final project involves normalizing a provided database table. The given sample data table currently lacks the third normal form (3NF). In a 1,100-2,300 word paper, you must:

1. Identify functional and transitive dependencies
2. Determine a primary key for the table
3. Explain why the table is not currently in 3NF
4. Analyze the current normalization state
5. Create a new table in third normal form (3NF)

Below is the provided database table:

Employee ID	Employee Last Name	Employee First Name	Street Address	City	State	ZIP Code	Department	Manager ID	Position	Salary
1005	Doe	Janet	312 Maple Drive	Anytown	FL	32829	Board of Directors	1000	President	$100,000
1010	Eyre	Jane	1200 First Street	Anytown	FL	32829	Administration	1005	Vice President	$95,000
1011	Bronte	Charlotte	4989 Fleur de Lane	Sometown	FL	32829	Administration	1005	Vice President	$75,000
2060	Poe	Edgar	12 Arcadia Avenue	Anytown	FL	32829	Information Technology	1015	Programmer II	$70,000
2090	Dickens	Charles	687 Gulf View Street	Sometown	FL	32830	Information Technology	1015	Programmer I	$45,000
2100	Doyle	AC	1209 Pine Tree Lane	Sometown	FL	32831	Information Technology	1015	Programmer I	$60,000
3230	Uberville	Tess	5435 Main Street	Anytown	FL	32831	Sales	1021	Sales Representative	$50,000
3330	Dumas	Alex	3 Post Drive	Sometown	FL	32831	Sales	1021	Sales Representative	$35,000

Part 1: Identify Functional and Transitive Dependencies

In the world of database design, understanding the relationships between attributes or columns is paramount. Two fundamental concepts that play a role in this domain are functional dependency and transitive dependency. Functional and transitive dependencies are crucial in database design. They reveal relationships between attributes, optimize structure, ensure data integrity, and facilitate normalization for an efficient database.

In a database table, functional dependency and transitive dependence refer to relationships between attributes or columns. Transitive dependency occurs when the value of a column or field depends on another column in the same table, and this dependency is facilitated through an intermediary column located between them. Functional dependence occurs when the value of one attribute uniquely determines the value of another attribute. Functional dependence is a direct relationship and transitive dependence involves more than two attributes.

In the provided database, Manager ID and Department showcase a functional dependency. The Department uniquely determines the manager overseeing that specific department. For instance, the Administration Department corresponds to Manager ID 1005, while the Sales Department aligns with Manager ID 1021. This direct correlation is a relationship between these two attributes.

The Employee ID plays a crucial role in our table through its functional dependency that uniquely determines all other attributes. The employee's name, address, and professional details such as department, manager's ID, position, and salary are all determined by a unique Employee ID.

Looking at transitive dependencies, one exists between the Position and Salary columns. While the salary relies on the employee's position, it also intertwines with the unique identity of the employee, represented by the Employee ID. The variance in salaries for employees holding the same position, such as Vice President, showcases the transitive dependency on Employee ID. Looking at the table we can see that Jane Eyre, employee number 1010 has a position of vice president and a salary of $95,000. Employee 1011, Charlotte Bronte, is also in the position of vice president but her salary is $75,000.

The Employee ID has a transitive dependency on the Department attribute. This connection is possible through the Manager ID column. The Employee ID attribute uses the Manager ID to assign the employee's department name. This is a transitive dependence because the dependency of the Employee ID on the Department is facilitated through the Manager ID column.

The Manager ID transitively determines an employee’s position. By examining the Manager ID, we can see the position of each employee under that manager. For instance, the employee working under MAnager ID 1021 is associated with the Sales Department and holds the position of sales representative. Employee number 1010, Jane Eyre, works for manager number 1005. Manager 1005 works for the Administration department and their employees are presidents.

Identifying functional and transitive dependencies in a database table is an important step in understanding the relationships between the data in the table. Uncovering these dependencies enhances our comprehension of the database structure and helps us optimize the database design, and work toward ensuring data integrity and normalization.

Part 2: Identify a Primary Key

A primary key is a unique value for each record (row) in a table. A primary key must be unique and cannot contain null values. The primary key for this table should be the Employee ID because it uniquely identifies each employee in the table, is a unique value (no two employees share the same ID), and is not null (each employee has an employee ID). Using Employee ID as the primary key will allow for efficient retrieval of employee information and help maintain data integrity.

Part 3: Explain Why Table is Not 3NF

Normalization involves eliminating anomalies, data redundancies, and inaccuracies from a database, and restricting tables to specific purposes or entities (Odugbesan, T., 2017). Established rules in database management system design aim to enhance table organization and minimize anomalies. The level of organization a table achieves is referred to as its form or stage of normalization. The three primary stages of normalization are identified as the first normal form (1NF), second normal form (2NF), and third normal form (3NF).

The third normal form (3NF) involves structuring database tables to eliminate transitive dependencies within a relational system. Transitive dependency occurs when a column or field's value in a table depends on another column in the same table, with this dependency facilitated through an intermediary column between them (Gibbs, M., 2020). To achieve the third normal form, a table must first be in the second normal form (2NF) and feature only columns or fields which have no transitive dependencies.

A table achieves the second normal form (2NF) when it satisfies the conditions of the first normal form (1NF) and has no partial dependencies of any column on the primary key (Odugbesan, T., 2017). Achieving compliance with 2NF often requires segregating the columns of a 1NF table and establishing relationships between them using primary and foreign keys.

For a table to meet the criteria of the first normal form (1NF), it should have a primary key, columns with single values, unique column names, uniform data types for attribute values, and no redundant data, ensuring that no two records (rows) are identical, including the absence of composite keys.

The table is currently in first normal form (1NF), indicated by the presence of a value that meets the criteria to be a suitable primary key (Employee ID), unique field names, no repeated data across fields, and the absences of composite keys (Gibbs, M., 2020). Employee ID serves as the unique identifier for each row, and the field names maintain uniqueness. The absence of composite keys ensures that no field is a combination of others. The adherence to 1NF principles is evident in the distinct attributes represented by each column, preventing a repetition of related attributes for the same employee.

To move from the first normal form to the second normal form, we need to ensure that there are no partial dependencies of any column on the primary key which is likely the Employee ID. The columns that depend only on the Employee ID are Employee Last Name, Employee First Name, Street Address, City, State, and ZIP Code. These columns are functionally dependent on the primary key and there are no partial dependencies within them. However, the Department, Manager ID, Position, and Salary fields depend on part of the primary key, Employee ID, since they appear to be related to the employee’s role and position. To address this, we can create separate tables for Employee Information and Employee Position. The Employee Information table will have columns for the Employee ID, Employee Last Name, Employee First Name, Street Address, City, State, and ZIP Code. The Employee Position table will have columns for the Employee ID, Department, Manager ID, Position, and Salary. The Employee ID will be used as a foreign key in the Employee Position table, linking it to the Employee Information table. This will eliminate partial dependencies and satisfy the requirements of the second normal form so that we can keep moving toward the third normal form (3NF).

The table is not in the third normal form because it shows transitive dependencies. Transitive dependency occurs when the value of a column or field depends on another column in the same table, and this dependency is facilitated through an intermediary column located between them (Gibbs, M., 2020). As we previously established, there is a transitive dependency between the Position and Salary columns, between the Employee ID and Department columns, and between the Manager ID and Position columns. We need to remove these transitive dependencies to achieve 3NF.

The Position attribute is functionally dependent on the Manager ID. The Manager ID is dependent on the Employee ID, which is the primary key. Therefore, the Position field is transitively dependent on the Employee ID through the Manager ID field. This violates the third normal form, as attributes should be directly dependent on the primary key, not other attributes. There are also transitive dependencies between the Position and Salary columns. Although salary relies on the employee’s position, it is tied to the Employee ID. The variation in salaries for employees in the same position, like Vice President, highlights the transitive dependency on Employee ID.

To bring the table into the third normal form (3NF), we need to eliminate transitive dependencies (Gibbs, M., 2020). We can achieve this by creating separate tables for various entities such as Employee for the employee names, Employee_Address for address information, Employee_Salary for salary information, Employee_Position for each position an employee has, Position for the titles and corresponding IDs, Employee_Department for which department(s) each employee works for, Department for who manages each department, and Department_Manager for the names of employees that manage each department. The tables will each have a unique primary key and tables that should be related will be through foreign keys.

Now, each table represents a distinct entity, and transitive dependencies have been eliminated. The database is now in the third normal form (3NF). The third normal form will be beneficial as it helps maintain data consistency, eliminate redundant data, simplify database maintenance, improve query performance, enhance the understanding of data relationships between entities, and increase the scalability of the database (Gibbs, M., 2020).

Part 4: Explain the Current Normalization Status

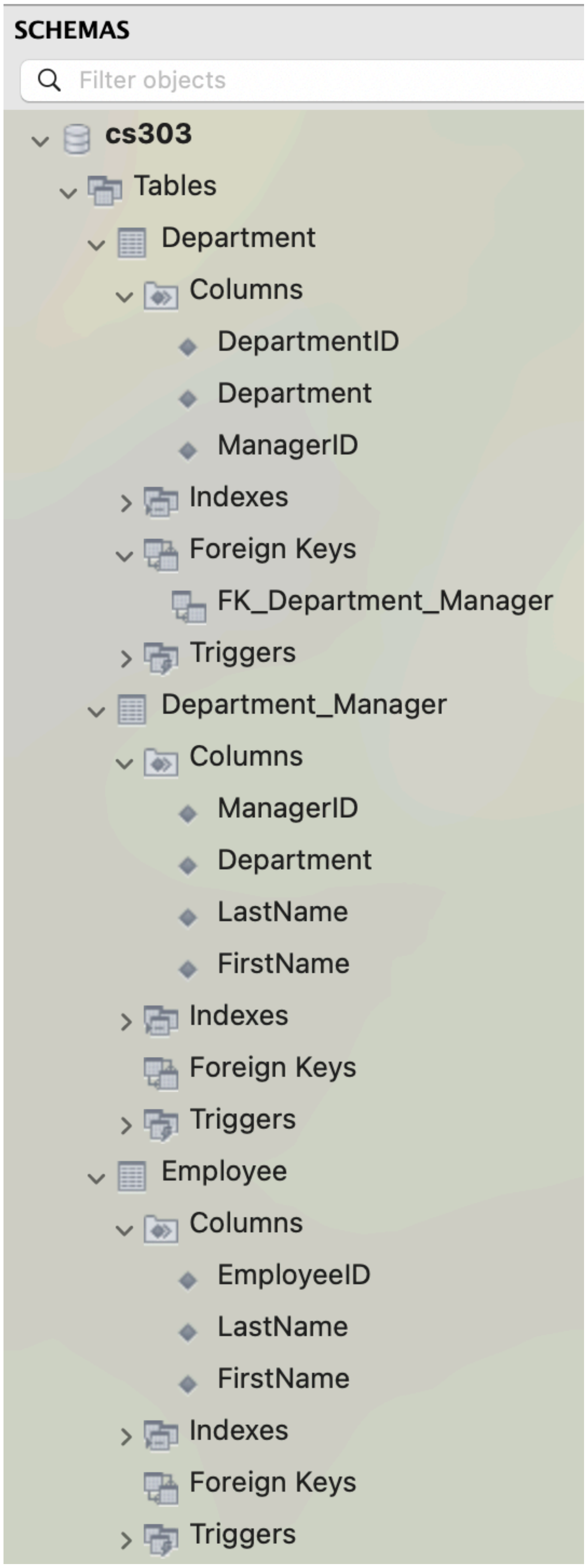
The table meets the criteria for the first normal form (1NF). A table in the first normal form will have a primary key, unique field names, non-repeated data across fields, and the absence of composite keys (Gibbs, M., 2020). The primary key is a unique identifier that corresponds to each row of data in the data. It is usually a numerical value. In this table, the Employee ID meets the criteria to be its primary key. The field names maintain uniqueness, consisting of Employee ID, Employee Last Name, Employee First Name, Street Address, City, State, ZIP Code, Department, Manager, and Position. A composite key is a field that is a combination of other fields, and this table shows no composite keys (Wert, M., 2018). There are no groups of columns repeated across the fields. Each column represents a distinct attribute about the employee entity being described, and there are no sets of related attributes that are repeated for multiple occurrences of the same employee.

While the table meets the criteria for the first normal form, it also adheres to the second normal form principles. According to the requirements for 2NF, a table must first be in 1NF and should not have partial dependencies on the primary key (Odugbesan, T., 2017). A partial dependency occurs when a non-prime attribute, or an attribute that is not part of the primary key, depends on only part of the primary key. Each non-prime attribute in the table depends on the entire primary key (Employee ID). There is no clear evidence of partial dependencies in the dataset, so it aligns with the principles of 2NF.

Part 5: Create the Tables in MySQL

```
CREATE TABLE Employee (

EmployeeID INT NOT NULL PRIMARY KEY UNIQUE AUTO_INCREMENT,

LastName VARCHAR(50),

FirstName VARCHAR(50)

);


CREATE TABLE Employee_Address (

EmployeeAddressID INT NOT NULL PRIMARY KEY UNIQUE AUTO_INCREMENT,

EmployeeID INT, #Foreign key referencing the PK in the Employee table

StreetAddress VARCHAR(100),

City VARCHAR(50),

State CHAR(2),

ZIPCode INT(5)

);


CREATE TABLE Employee_Salary (

EmployeeSalaryID INT NOT NULL,

Salary FLOAT(2),

EmployeeID INT #Foreign key referencing the PK in the Employee table

);


CREATE TABLE Employee_Position (

EmployeePositionID INT NOT NULL PRIMARY KEY,

PositionTitle VARCHAR(50),
EmployeeID INT, #Foreign key referencing the PK in the Employee table

PositionID INT #Foreign key referencing the PK in the Position table

);


CREATE TABLE Position (

PositionID INT NOT NULL PRIMARY KEY UNIQUE,

PositionTitle VARCHAR(50)

);


CREATE TABLE Employee_Department (

EmployeeDepartmentID INT NOT NULL PRIMARY KEY,

DepartmentID INT, #Foreign key referencing the PK in the Department table

EmployeeID INT

);


CREATE TABLE Department (

DepartmentID INT NOT NULL PRIMARY KEY,

Department VARCHAR(100),

ManagerID INT #Foreign key referencing the PK in the Department_Manager table

);


CREATE TABLE Department_Manager (

ManagerID INT NOT NULL PRIMARY KEY,

Department VARCHAR(100),

LastName VARCHAR(50),
FirstName VARCHAR(50)

);


# Add foreign key constraint for EmployeeID in the Employee_Address table

ALTER TABLE Employee_Address

ADD CONSTRAINT FK_EmployeeAddress_Employee

FOREIGN KEY (EmployeeID)

REFERENCES Employee(EmployeeID);


# Add foreign key constraint for EmployeeID in the Employee_Salary table

ALTER TABLE Employee_Salary

ADD CONSTRAINT FK_EmployeeSalary_Employee

FOREIGN KEY (EmployeeID)

REFERENCES Employee(EmployeeID);


# Add foreign key constraint for EmployeeID in the Employee_Position table

ALTER TABLE Employee_Position

ADD CONSTRAINT FK_EmployeePosition_Employee

FOREIGN KEY (EmployeeID)

REFERENCES Employee(EmployeeID);


# Add foreign key constraint for PositionID in the Employee_Position table

ALTER TABLE Employee_Position

ADD CONSTRAINT FK_EmployeePosition_Position

FOREIGN KEY (PositionID)
```

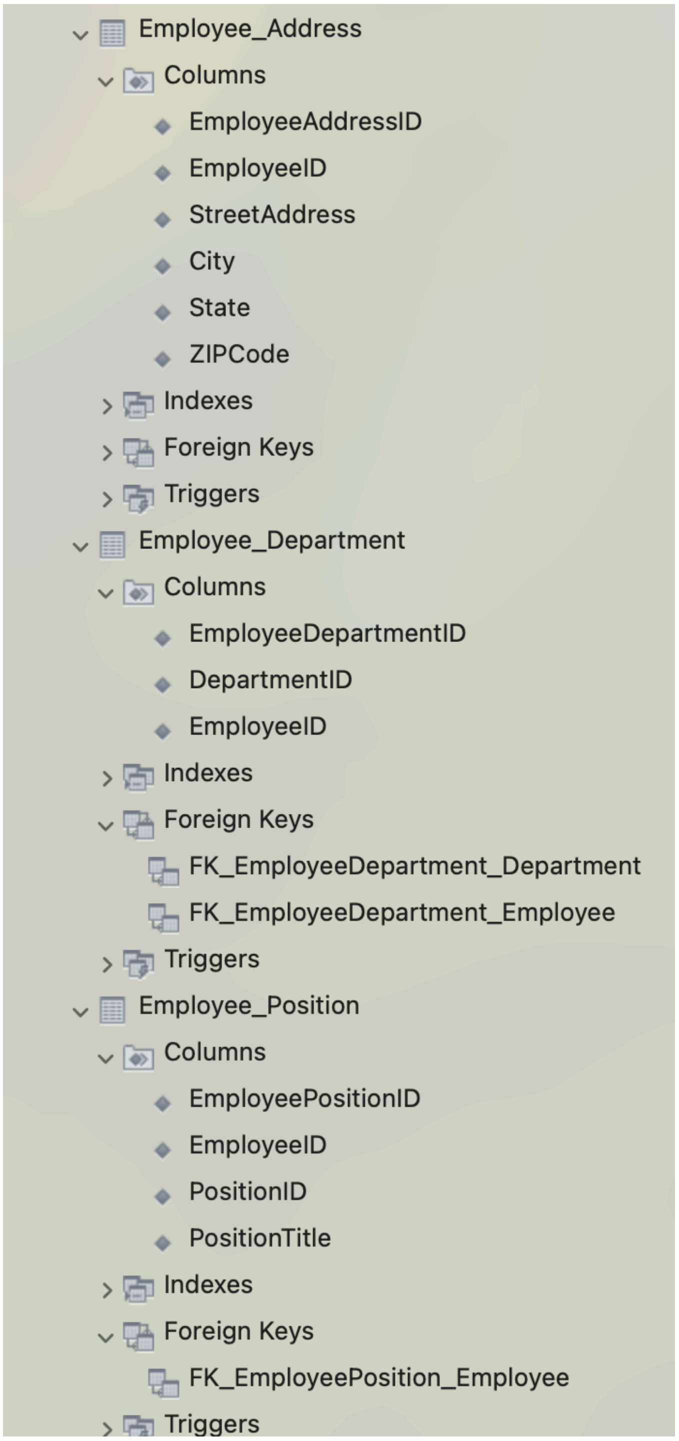
```
REFERENCES Position(PositionID);


# Add foreign key constraint for DepartmentID

ALTER TABLE Employee_Department

ADD CONSTRAINT FK_EmployeeDepartment_Department

FOREIGN KEY (DepartmentID)

REFERENCES Department(DepartmentID);


# Add foreign key constraint for EmployeeID

ALTER TABLE Employee_Department

ADD CONSTRAINT FK_EmployeeDepartment_Employee

FOREIGN KEY (EmployeeID)

REFERENCES Employee(EmployeeID);


# Add foreign key constraint for ManagerID

ALTER TABLE Department

ADD CONSTRAINT FK_Department_Manager

FOREIGN KEY (ManagerID)

REFERENCES Department_Manager(ManagerID);
```



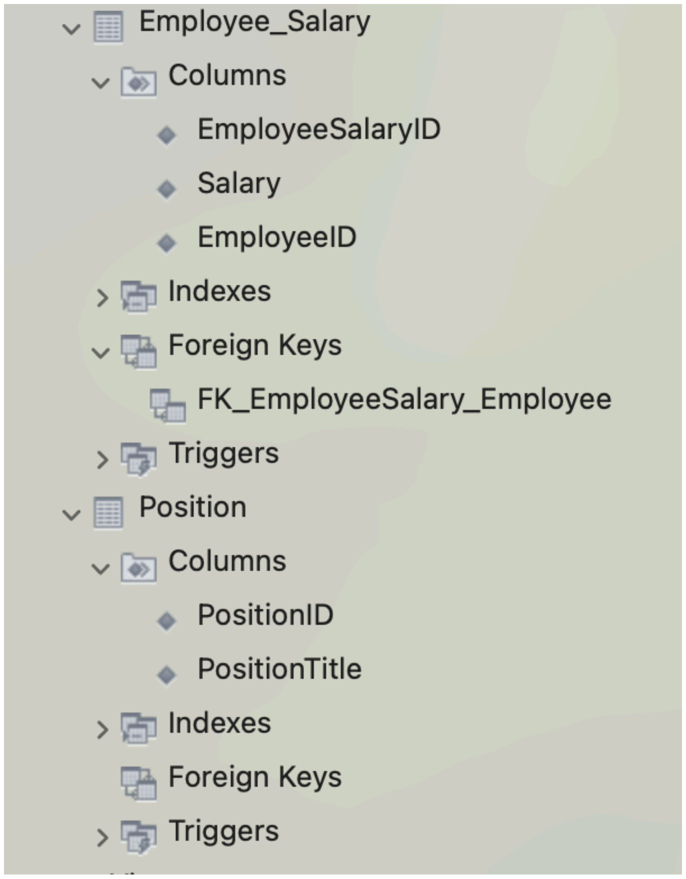
SCHEMAS

Filter objects

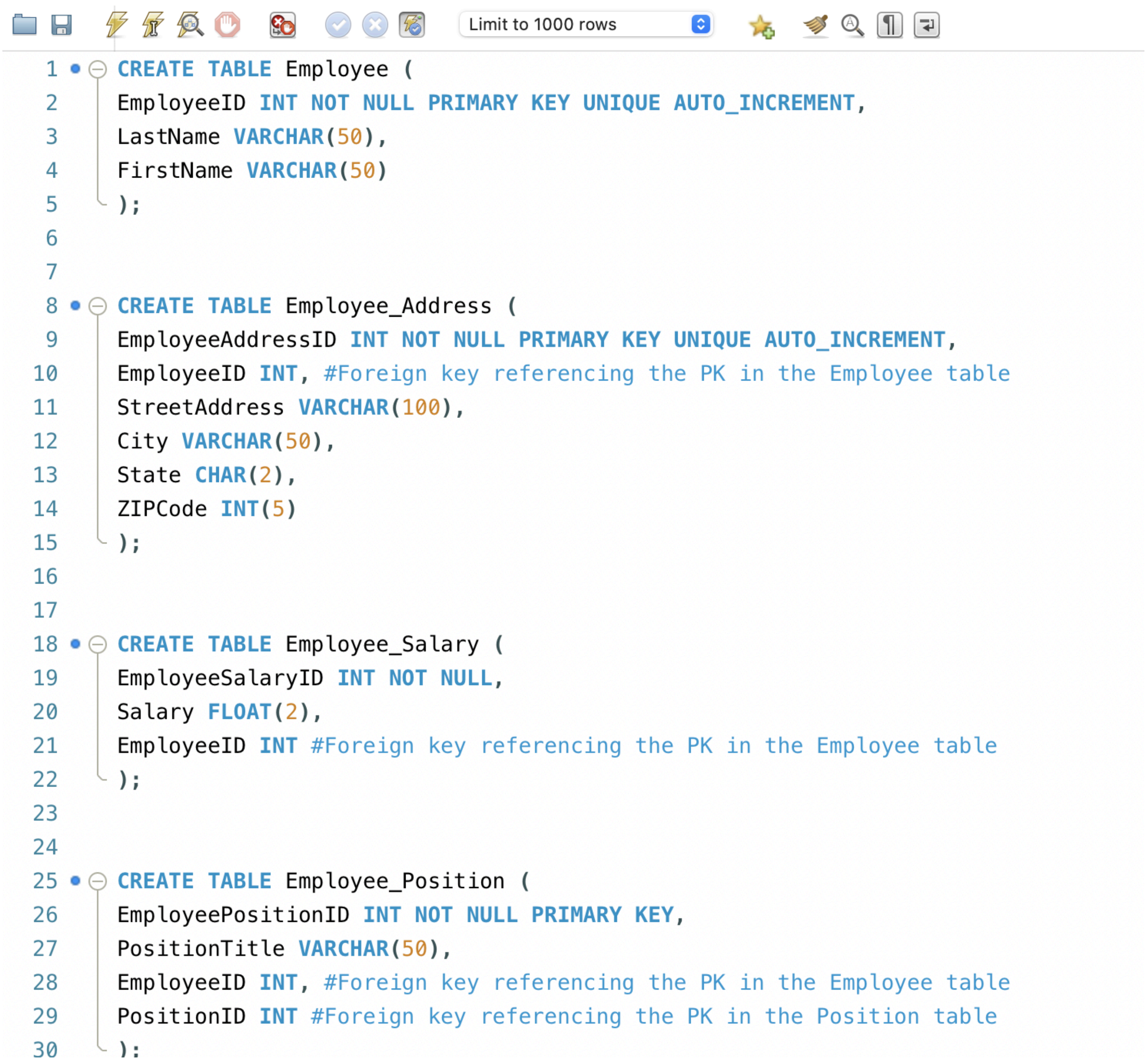
- cs303
  - Tables
    - Department
      - Columns
        - DepartmentID
        - Department
        - ManagerID
      - Indexes
      - Foreign Keys
        - FK_Department_Manager
      - Triggers
    - Department_Manager
      - Columns
        - ManagerID
        - Department
        - LastName
        - FirstName
      - Indexes
      - Foreign Keys
      - Triggers
    - Employee
      - Columns
        - EmployeeID
        - LastName
        - FirstName
      - Indexes
      - Foreign Keys
      - Triggers



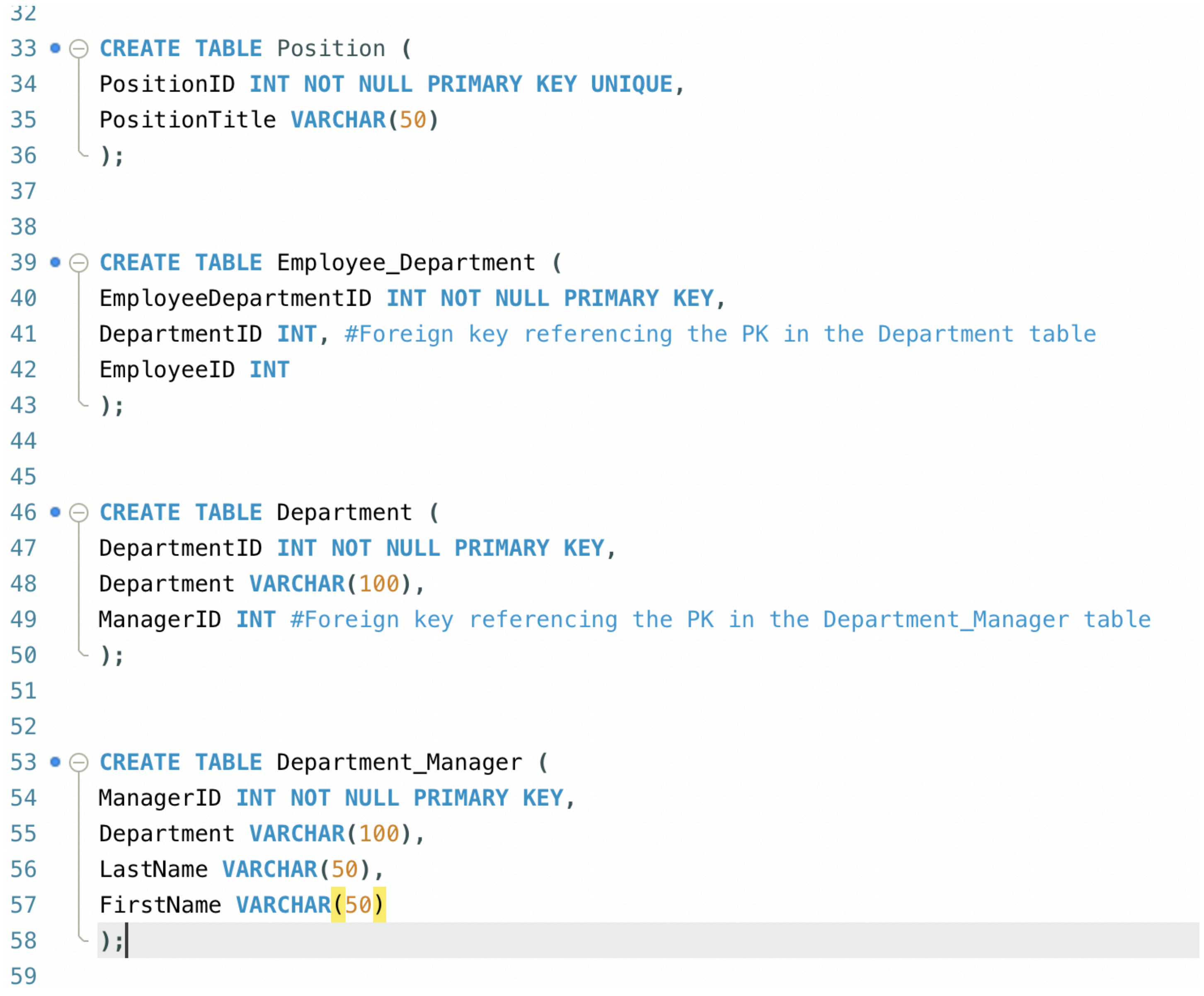
- Employee_Address
  - Columns
    - EmployeeAddressID
    - EmployeeID
    - StreetAddress
    - City
    - State
    - ZIPCode
  - Indexes
  - Foreign Keys
  - Triggers
- Employee_Department
  - Columns
    - EmployeeDepartmentID
    - DepartmentID
    - EmployeeID
  - Indexes
  - Foreign Keys
    - FK_EmployeeDepartment_Department
    - FK_EmployeeDepartment_Employee
  - Triggers
- Employee_Position
  - Columns
    - EmployeePositionID
    - EmployeeID
    - PositionID
    - PositionTitle
  - Indexes
  - Foreign Keys
    - FK_EmployeePosition_Employee
  - Triggers



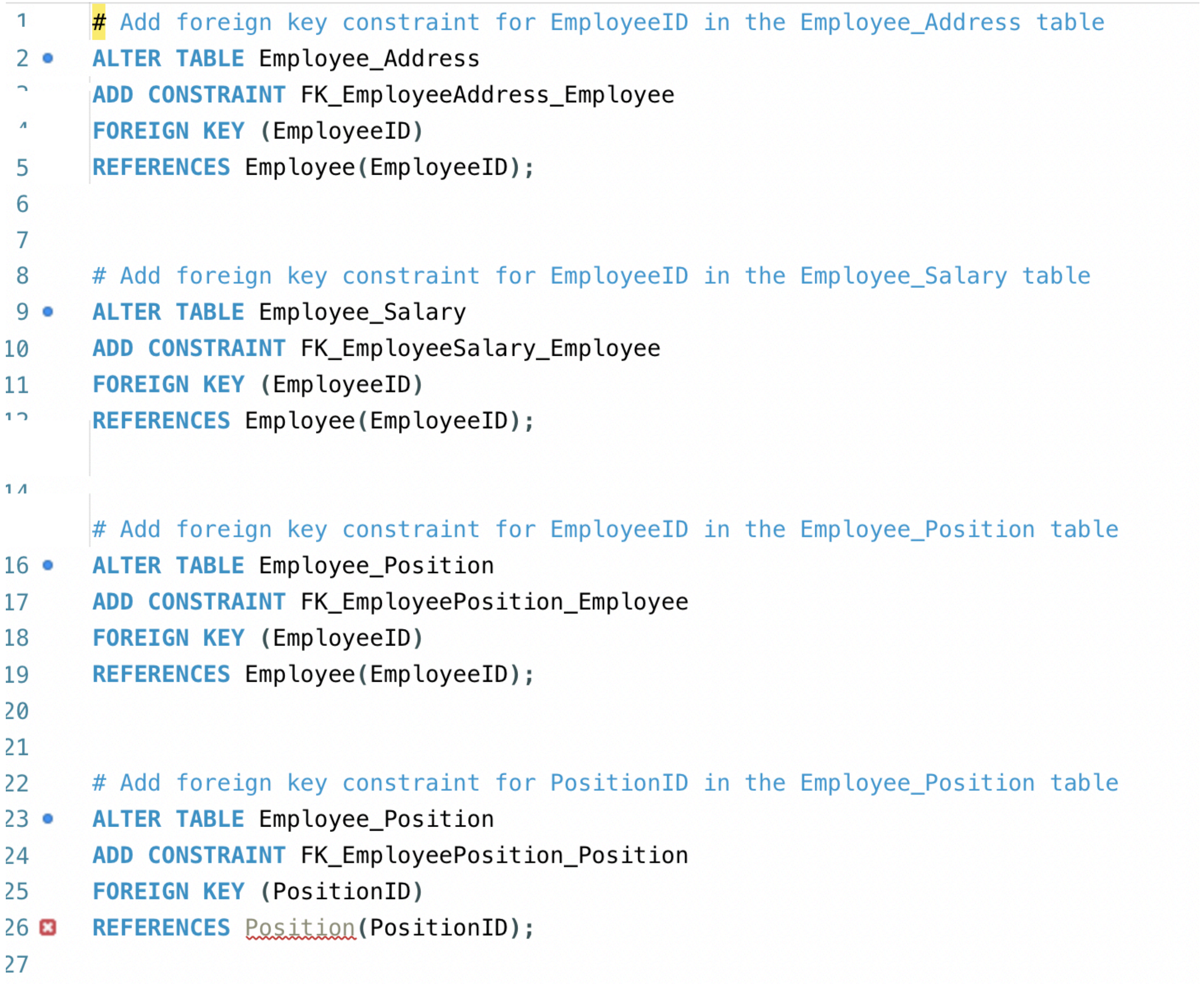
Employee_Salary
Columns
EmployeeSalaryID
Salary
EmployeeID
Indexes
Foreign Keys
FK_EmployeeSalary_Employee
Triggers
Position
Columns
PositionID
PositionTitle
Indexes
Foreign Keys
Triggers



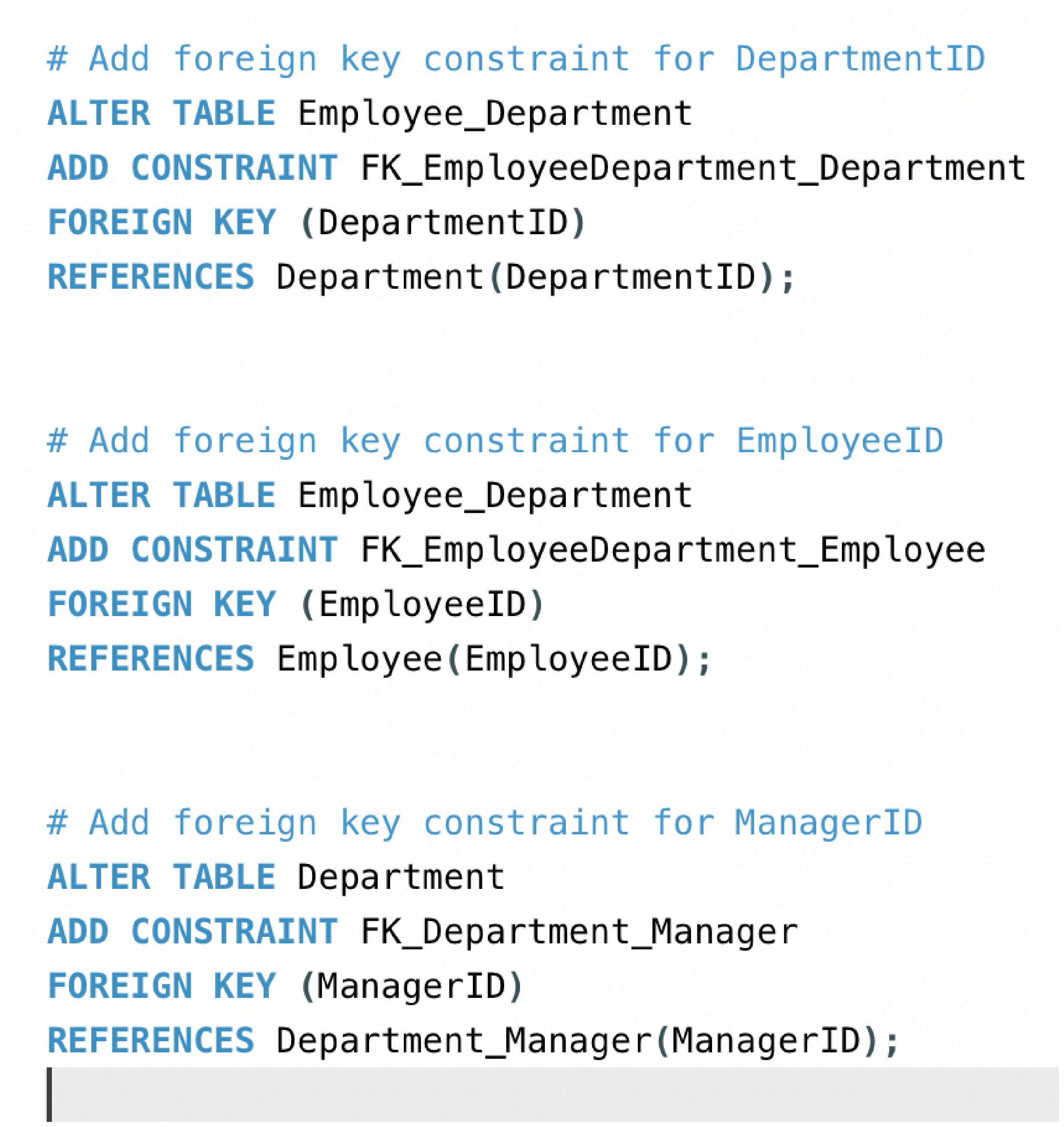
```sql
CREATE TABLE Employee (
EmployeeID INT NOT NULL PRIMARY KEY UNIQUE AUTO_INCREMENT,
LastName VARCHAR(50),
FirstName VARCHAR(50)
);


CREATE TABLE Employee_Address (
EmployeeAddressID INT NOT NULL PRIMARY KEY UNIQUE AUTO_INCREMENT,
EmployeeID INT, #Foreign key referencing the PK in the Employee table
StreetAddress VARCHAR(100),
City VARCHAR(50),
State CHAR(2),
ZIPCode INT(5)
);


CREATE TABLE Employee_Salary (
EmployeeSalaryID INT NOT NULL,
Salary FLOAT(2),
EmployeeID INT #Foreign key referencing the PK in the Employee table
);


CREATE TABLE Employee_Position (
EmployeePositionID INT NOT NULL PRIMARY KEY,
PositionTitle VARCHAR(50),
EmployeeID INT, #Foreign key referencing the PK in the Employee table
PositionID INT #Foreign key referencing the PK in the Position table
);
```



```sql

CREATE TABLE Position (
PositionID INT NOT NULL PRIMARY KEY UNIQUE,
PositionTitle VARCHAR(50)
);


CREATE TABLE Employee_Department (
EmployeeDepartmentID INT NOT NULL PRIMARY KEY,
DepartmentID INT, #Foreign key referencing the PK in the Department table
EmployeeID INT
);


CREATE TABLE Department (
DepartmentID INT NOT NULL PRIMARY KEY,
Department VARCHAR(100),
ManagerID INT #Foreign key referencing the PK in the Department_Manager table
);


CREATE TABLE Department_Manager (
ManagerID INT NOT NULL PRIMARY KEY,
Department VARCHAR(100),
LastName VARCHAR(50),
FirstName VARCHAR(50)
);

```



```
# Add foreign key constraint for EmployeeID in the Employee_Address table
ALTER TABLE Employee_Address
ADD CONSTRAINT FK_EmployeeAddress_Employee
FOREIGN KEY (EmployeeID)
REFERENCES Employee(EmployeeID);


# Add foreign key constraint for EmployeeID in the Employee_Salary table
ALTER TABLE Employee_Salary
ADD CONSTRAINT FK_EmployeeSalary_Employee
FOREIGN KEY (EmployeeID)
REFERENCES Employee(EmployeeID);


# Add foreign key constraint for EmployeeID in the Employee_Position table
ALTER TABLE Employee_Position
ADD CONSTRAINT FK_EmployeePosition_Employee
FOREIGN KEY (EmployeeID)
REFERENCES Employee(EmployeeID);


# Add foreign key constraint for PositionID in the Employee_Position table
ALTER TABLE Employee_Position
ADD CONSTRAINT FK_EmployeePosition_Position
FOREIGN KEY (PositionID)
REFERENCES Position(PositionID);
```



```
# Add foreign key constraint for DepartmentID
ALTER TABLE Employee_Department
ADD CONSTRAINT FK_EmployeeDepartment_Department
FOREIGN KEY (DepartmentID)
REFERENCES Department(DepartmentID);


# Add foreign key constraint for EmployeeID
ALTER TABLE Employee_Department
ADD CONSTRAINT FK_EmployeeDepartment_Employee
FOREIGN KEY (EmployeeID)
REFERENCES Employee(EmployeeID);


# Add foreign key constraint for ManagerID
ALTER TABLE Department
ADD CONSTRAINT FK_Department_Manager
FOREIGN KEY (ManagerID)
REFERENCES Department_Manager(ManagerID);
```

References

Gibbs, M. (2020, March 14). *Third normal form in DBMS with examples*. Study. https://study.com/academy/lesson/third-normal-form-in-dbms-with-examples.html

Gibbs, M. (2020, March 16). *First normal form in DBMS with examples*. Study. https://study.com/academy/lesson/first-normal-form-in-dbms-with-examples.html

Odugbesan, T. (2017, December 30). *What is normal form in DBMS? - types & examples*. Study. https://study.com/academy/lesson/what-is-normal-form-in-dbms-types-examples.html

Wert, M. (2018, November 21). *What is an attribute in a database?* Study. https://study.com/academy/lesson/what-is-an-attribute-in-a-database.html