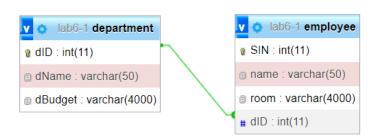


The "Works In" relationship above can be efficiently represented in different ways in our SQL code.

Approach 1: Using a foreign key

Assumption: *Employee* and *Department* tables have a 1:N (one-to-many) relationship i.e., one employee works in one department, but one department can have many employees. SQL Queries:

```
CREATE TABLE Employee (
                             NOT NULL
      SIN
           INT
                                         PRIMARY KEY,
                             NOT NULL,
      name VARCHAR(50)
      room VARCHAR(4000)
                             NOT NULL,
      dID INT,
      FOREIGN KEY (dID) REFERENCES Department (dID)
);
CREATE TABLE Department (
      dID
                 INT
                              NOT NULL
                                         PRIMARY KEY,
                 VARCHAR(50)
      dName
                               NOT NULL,
      dBudget
                 VARCHAR(4000) NOT NULL
);
```



In the *Department* table, *dID* is the primary key followed by *dName* and *DBudget* which are non-prime attributes.

In the *Employee* table, *SIN* is a primary key followed by *name* and *room* which are non-prime attributes. *dID* is the foreign key referencing the primary key from the *Department* table, used

to link the two tables together. The *dID* column is very crucial as it prevents broken relationships between the two tables and orphan rows (i.e., referential integrity).

Approach 2: Using a junction table

Assumption: *Employee* and *Department* tables have a N:M (many-to-many) relationship i.e., one employee can work in multiple departments and, one department can have many employees.

```
SQL Queries:
CREATE TABLE Employeex
              SIN
                     INT
                                         NOT NULL
                                                       PRIMARY KEY,
             name VARCHAR(50)
                                         NOT NULL,
             room VARCHAR(4000)
                                         NOT NULL
);
CREATE TABLE Departmentx
              dID
                    INT
                                         NOT NULL
                                                       PRIMARY KEY,
             dName
                           VARCHAR(50)
                                                NOT NULL.
                           VARCHAR(4000)
                                                NOT NULL
             dBudget
);
CREATE TABLE EmployeeDepartment
             SIN INT NOT NULL.
             dID INT NOT NULL.
             CONSTRAINT PK_EmpDept PRIMARY KEY (SIN, dID),
             FOREIGN KEY (SIN) REFERENCES Employeex(SIN),
             FOREIGN KEY (dID) REFERENCES Departmentx(dID)
);
                                                   v 💠 lab6-1 employeex
   v 🐧 lab6-1 departmentx
                                                   SIN : int(11)
    @ dID : int(11)
                                                   name: varchar(50)
    dName : varchar(50)
                                                   noom: varchar(4000)
    g dBudget : varchar(4000)
                        v 💠 lab6-1 employeedepartment
                         SIN : int(11)
```

@ dID : int(11)

As there is an N:M (many-to-many) relationship between the tables *Employeex* and *Departmentx*, we created a third table *EmployeeDepartment* which holds a composite primary key with *SIN* and *dID*. Here, the foreign key *SIN* refers to the primary key from *Employeex* table whereas the foreign key *dID* refers to the primary key from *Departmentx* table.

Explanation:

In terms of design principles, if the relationship is 1:N, it makes no sense to create a third table (junction table) since a foreign key can do the job better. In the case of an N:M relationship, the existing tables should be broken down into smaller tables or a junction table connecting

the main tables, to achieve normalization. Considering the relationship between the tables is crucial to implementing an efficient approach. However, under certain constraints, the contents of the junction table can be folded down into one of the base tables as additional columns, improving efficiency.

When two alternatives for a database design exist, it would be wise to examine which alternative more closely fits the underlying conceptual and relational models, which can entail performance advantages and space savings.

Attribution:

These assignments were completed by **Ravi Chandan Pandi**, and they represent his original work completed for academic purposes during his studies and self-learning purposes.

Please note that the documents shared here are intended for educational and informational purposes only. Any unauthorized use or reproduction is strictly prohibited. If you have any questions or would like to reach out to Ravi, you can contact him on LinkedIn. [https://www.linkedin.com/in/ravichandan/].