Exercise 2: Theoretical Problems

Student Number: 100269314

5. Write `CREATE TABLE` commands for corresponding tables with given constraints and default values.

Solution.

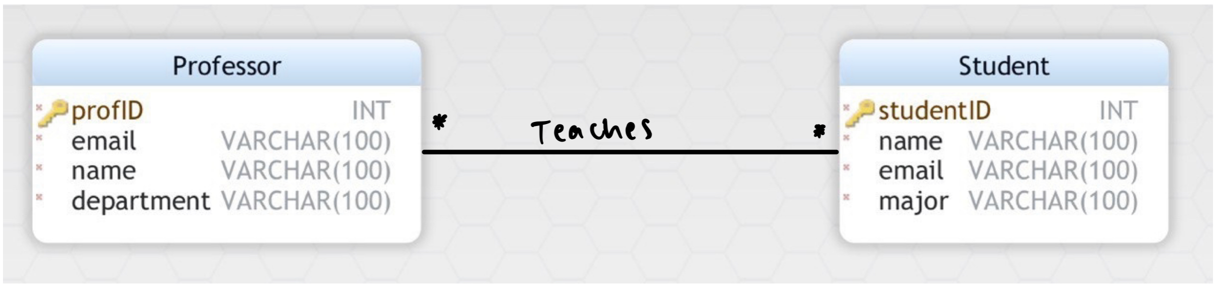
```
CREATE TABLE Student (
    name        VARCHAR (100) NOT NULL,
    studentID   INTEGER       NOT NULL,
    email       VARCHAR (100) NOT NULL,
    studyProgram VARCHAR (100) CHECK (studyProgram IN ('data science',
        'computer science', 'mathematics', 'industrial engineering') ),
    PRIMARY KEY (
        studentID
    )
);

CREATE TABLE Course (
    code    VARCHAR (100) NOT NULL,
    name    VARCHAR (100) NOT NULL,
    credits NUMERIC (3, 1) NOT NULL
                          CHECK (credits > 0.0),
    teacher VARCHAR (100),
    MOOC    BOOLEAN       DEFAULT FALSE,
    PRIMARY KEY (
        code
    )
);

CREATE TABLE Grade (
    studentID INTEGER      NOT NULL,
    courseCode VARCHAR (100) NOT NULL,
    grade     INTEGER      NOT NULL
                           CHECK (grade >= 0 AND
                                  grade <= 5),
    PRIMARY KEY (
        studentID,
        courseCode,
        grade
    ),
    FOREIGN KEY (studentID) REFERENCES Student (studentID),
    FOREIGN KEY (courseCode) REFERENCES Course (code)
);
```

6a. Create an example of a many-many relationship. Give at least 3 attributes for both of the classes

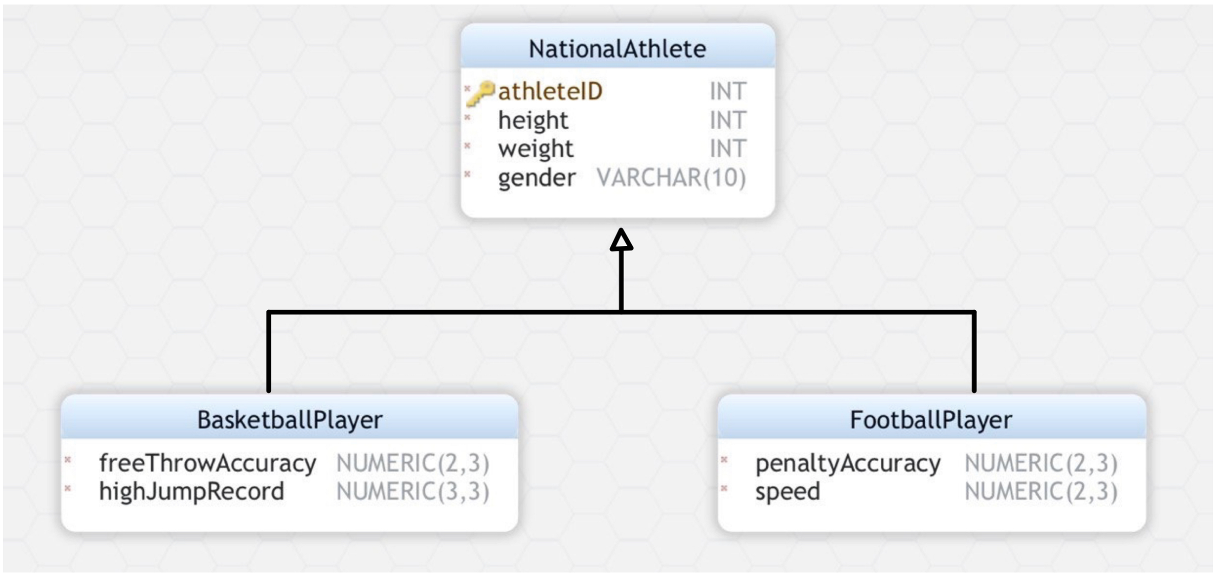
Solution.



Each Professor can teach arbitrarily many Students and each Student can be taught by arbitrarily many Professors that they are associated with. Therefore, this is a many-many relationship.

6b. Create an example of a subclass structure with at least two subclasses. Give at least 3 attributes for the superclass and at least 1 attribute for each of the subclasses

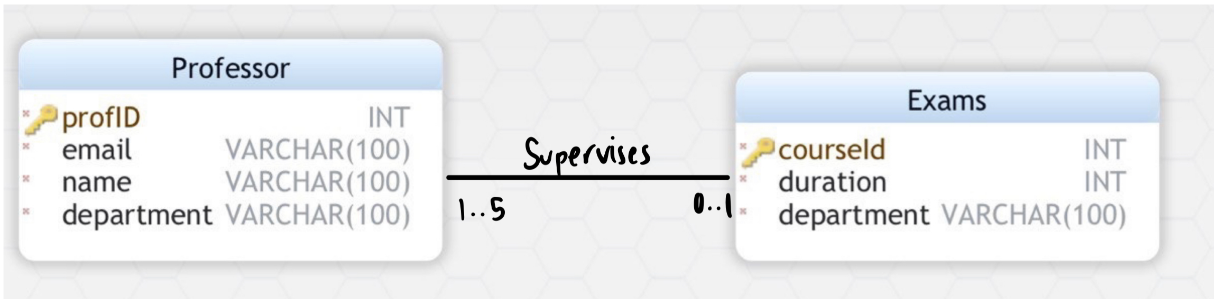
Solution.



Only some objects of the class National Athletes have certain features (e.g. only football players have feature `freeThrowAccuracy`) so we define subclasses. Objects of Basketball Player or Football Player have all attributes and associations of its superclass and also the attributes and associations of their own subclass.

6c. Create an example of an association class between two classes. Give at least 3 attributes for both of the classes.

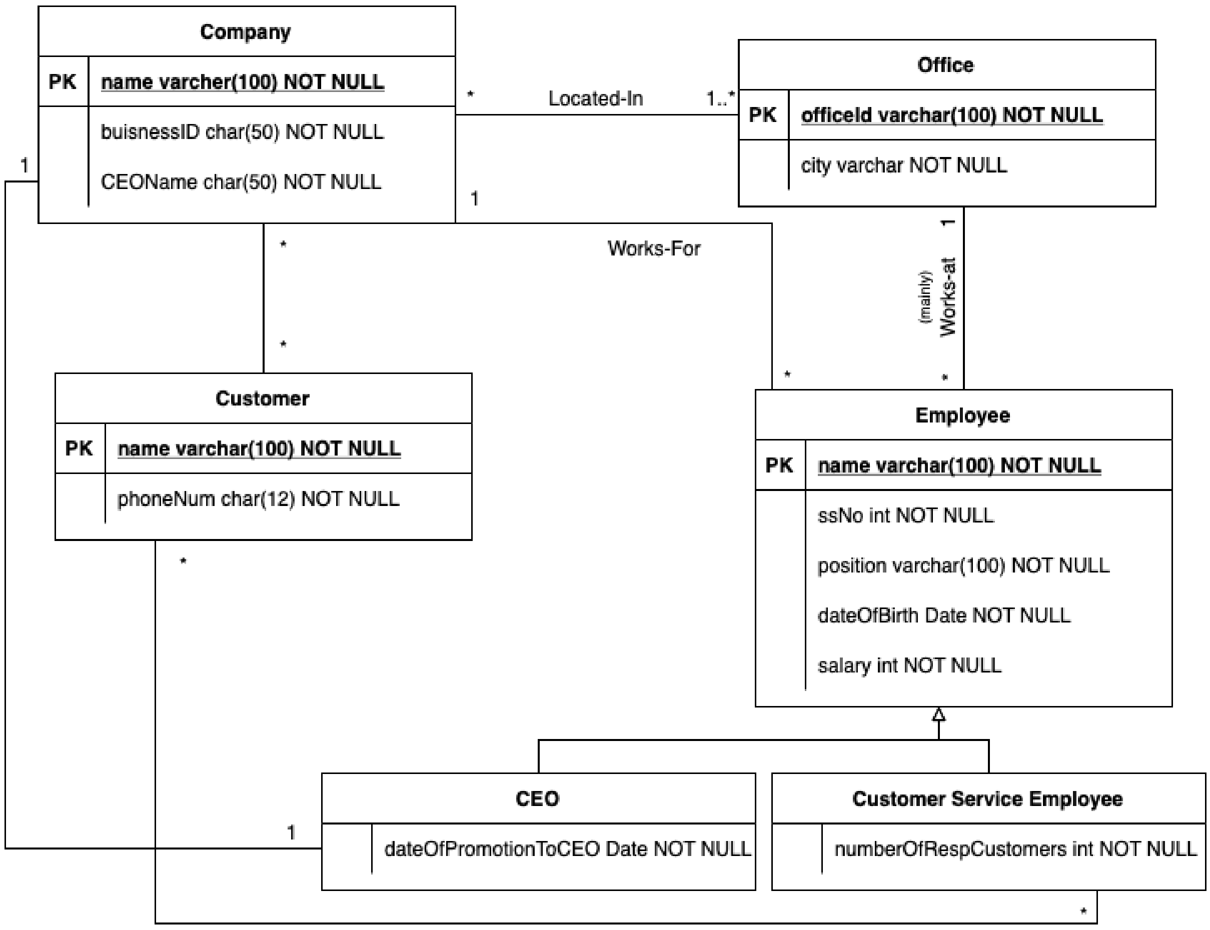
Solution.



Professor class/entity is associated to Exams class/entity as Professors supervise Exams (and as corollary, Exams are supervised-by Professors). The multiplicity of association is indicated in the figure above.

7. Give an UML diagram for a database recording information about companies, employees, and customers

Solution.


Company
PK name varcher(100) NOT NULL
buisnessID char(50) NOT NULL
CEOName char(50) NOT NULL
Located-In
*
1..*
Office
PK officeId varchar(100) NOT NULL
city varchar NOT NULL
1
1
Works-For
*
*
*
(mainly)
Works-at
1
*
Customer
PK name varchar(100) NOT NULL
phoneNum char(12) NOT NULL
*
Employee
PK name varchar(100) NOT NULL
ssNo int NOT NULL
position varchar(100) NOT NULL
dateOfBirth Date NOT NULL
salary int NOT NULL
CEO
dateOfPromotionToCEO Date NOT NULL
1
Customer Service Employee
numberOfRespCustomers int NOT NULL
*

Note: The inheritance arrow may be replaced by an open diamond to indicate aggregation (but not composition as an employee need not be either a CEO or Customer Service Employee).

8. List all the non-trivial functional dependencies. You may assume that the emails and phone numbers are unique, and that one person can only own one library card.

Solution.

`LibraryItem.ID` $\longrightarrow$ `loanPeriod`

`LibraryCustomer.libraryCardNumber` $\longrightarrow$ `name, phone, email`
and all of its variations (of different combinations of dependants) that can be created by *Splitting & Combining FD rules* explained during lecture

`Book.ID` $\longrightarrow$ `Book.title, Book.author, Book.language, Book.year`
and all of its variations (of different combinations of dependants) that can be created by *Splitting & Combining FD rules* explained during lecture

`CD.ID` $\longrightarrow$ `CD.title, CD.artist, CD.year, CD.length`
and all of its variations (of different combinations of dependants) that can be created by *Splitting & Combining FD rules* explained during lecture

`DVD.ID` $\longrightarrow$ `DVD.title, DVD.year, DVD.length`
and all of its variations (of different combinations of dependants) that can be created by *Splitting & Combining FD rules* explained during lecture

`Loan.itemID, Loan.libraryCardNumber, Loan.startDate` $\longrightarrow$ `endDate, returned`
and all of its variations (of different combinations of dependants) that can be created by *Splitting & Combining FD rules* explained during lecture

Note: I was advised by a TA that the above method of listing the functional dependencies is sufficient (instead of actually listing all of the combinations) - please do not penalize me for not explicitly listing all the FD's :-)

9. Consider the relation and the functional dependencies. Which of the following functional dependencies can be derived from the listed ones? Justify your answer by computing the closures of the left-hand-sides.

Solution. Functional dependencies a and c can be derived, but b cannot be derived. I demonstrate this by computing the closures of the left-hand-sides.

$$R(A,B,C,D,E,F)$$

$$S = \{A \to F,\ D \to C,\ CF \to BE,\ B \to DE\}$$

9a

$$\{A,D\}^+ \overset{A\to F\ \&\ D\to C}{=} \{A,C,D,F\}$$

$$\Rightarrow \{A,D\}^+ \overset{CF\to BE}{=} \{A,B,C,D,E,F\}$$

$$\Rightarrow \underline{AD \to E}$$

9b

$$\{B,C\}^+ \overset{B\to DE}{=} \{B,C,D,E\}$$

$$\Rightarrow \underline{BC \nrightarrow ABCDEF}$$

9c

$$\{A,C\}^+ \overset{A\to F}{=} \{A,C,F\}$$

$$\Rightarrow \{A,C\}^+ \overset{CF\to BE}{=} \{A,B,C,E,F\}$$

$$\Rightarrow \{A,C\}^+ \overset{B\to DE}{=} \{A,B,C,D,E,F\}$$

$$\Rightarrow \underline{AC \longrightarrow BDEF}$$

10. Which of the following functional dependencies are possible considering the contents of the table? Justify your answers.

Solution. By studying the shared attribute values across the tuples in the table provided, we can quickly identify which attribute(s) cannot be PK. From this information, we can assess the feasibility of the following FD's. As stated in the lecture, a functional dependency implies that if two tuples of R agree on all of the attributes on the left (of the arrow; determinants), then they will also have the same values in the attributes on the right (dependents). Let us now consider the supposed functional dependencies.

(a) $A \longrightarrow E$: This functional dependency is possible as it abides by (or rather, doesn't break) the principle of functional dependency mentioned above. We can see that when A is 123, E is always "locked-in" at 0.12 in the visible tuples. While this doesn't prove that $A \longrightarrow E$ is definitively true, this indicates that it is a possible functional dependency.

(b) $E \longrightarrow A$: This functional dependency is not possible as it breaks the principle of functional dependency mentioned above. We can see that when E is 0.12, A is not "locked-in" at one value but is 123 or 811 in the visible tuples. This indicates that it is an impossible functional dependency.

(c) $D \longrightarrow ABE$: This functional dependency is not possible as it breaks the principle of functional dependency mentioned above. We can see this in the first two tuple where the value of attribute D is identical (i.e. "A") but the value of B - which is on the RHS of the FD - differs (i.e. 14 and 42). This indicates that it is an impossible functional dependency.

`(d)` $DC \longrightarrow ABCDE$: This functional dependency is possible as it does not break the principle of functional dependency mentioned above in the visible tuples. Because there is no visible tuple that share the same DC, it is a possibility that the two keys are PK(s) that functionally determine all other attributes of the relation. While this may not be true, this functional dependency is certainly a possibility.

`(e)` $B \longrightarrow C$: This functional dependency is possible for the same reason as in part a. We can see that when B is 14, C is always "locked-in" at 1 and when when B is 12, C is always "locked-in" at 1 in the scope of the visible tuples. While this doesn't prove that $B \longrightarrow C$ is definitively true, this indicates that it is a possible functional dependency.

`(f)` $BD \longrightarrow E$: This functional dependency is possible for the same reason as part d. Because there is no visible tuple that share the same values for BD, it is a possibility that the two keys compose of a PK(s) that functionally determine all other attributes of the relation. The FD does not infringe the principle of functional dependency mentioned above in the visible tuples and is a possible FD.