UNIVERSITÄT OSNABRÜCK

Institute of Computer Science ROSEN-Group-Endowed Chair of Semantic Information Systems

Semantische Informationssysteme

Prof. Dr. Martin Atzmüller, Leonid Schwenke SoSe 2023

Projektübung. Übungsblatt zur Vorlesung «Datenbanksysteme»

Projektübungsblatt (insbesondere relevant für Klausurzulassung)

Woche: 2023-05-11 – 2023-06-20

Hinweise

- *Wichtig*: Das bearbeitete Übungsblatt (Datei) mit den entsprechenden Lösungen muss über VIPS bis *Dienstag, 20.06.2023, 07:59 Uhr* erfolgreich eingereicht werden!
- Eine erfolgreiche Abgabe/Bearbeitung des Übungsblatts mit der entsprechenden Lösung, sowie darauf folgend das Bestehen der entsprechenden Projektübung ("Testat") sind zum Erwerben der Prüfungszulassung notwendig.
- Für eine erfolgreiche Bearbeitung des Übungsblattes sind 50 Punkte hinreichend.
- In Folge eines Täuschungsversuches/Plagiates wird das Übungsblatt mit 0 Punkten bewertet.
- Bei den Aufgaben können Sie ggf. auch eine Grafik/Skizze nutzen bzw. einen (lesbaren!) Scan einbauen.
- Die Abgabe erfolgt per VIPS. Geben Sie dafür ein PDF und die implementierte .py Datei als .zip File ab. Der Dateiname der Zip-Datei sollte ihrer Gruppennummer entsprechen. In dem PDF sollte ihre Gruppennummer und die Namen der Gruppenangehörigen aufgelistet sein. Zur Benennung der PDF- bzw. Python-Datei siehe unten.

Anforderungen

Geben Sie ein PDF-Dokument und die implementierte Python-Datei ab, welche die unten aufgeführten Aufgaben lösen. Die im Laufe der Aufgaben zu erstellende Python-Datei MUSS lauffähig sein und die bereits definierten Schnittstellen nutzen.

Benennen Sie PDF und Python-Date (.py) nach folgendem Schema:

- PDF: Projektuebung-Gruppe<Gruppennummer>.pdf
  Beispiel: Projektuebung-Gruppe4311.pdf
- Python: dbs2023ExerciseProject-<Gruppennummer>.py
  Beispiel: dbs2023ExerciseProject-Gruppe4311.py

Aufgabe 1. (ER-Diagramm, Relationen-Schema, Normalisierung (12 Punkte))

Betrachten Sie folgendes Szenario:
Gegeben sei eine Fantasy-Stadt. In der Stadt sei dabei jede *Person* eine Instanz einer Entität. Eine Person hat dabei einen Namen, Alter, Beruf und einen Status. Personen haben dabei Relationen zueinander in Form von Verwandten. Die Stadt besitzt dabei verschiedene Art von *Gebäuden*. Dazu gehört das typische *Wohngebäude*. Jedes Gebäude hat einen Wert und hat eine Adresse + Hausnummer. Die Gebäude werden weiter unterteilt in *Luxus-Villa*, *Mittelstands-Haus* und *Wohnblock*. Eine Luxus-Villa bringt dabei maximal 6 Bewohner unter. Ein Mittelstands-Haus maximal 10 und ein Wohnblock maximal 30 Bewohner. Gebäude haben dabei Relationen zu anderen Gebäuden haben in Form von direkten Nachbarn. Dabei hat jedes Gebäude mindestens 2 direkte Nachbarn und maximal 4. Jedes Gebäude hat auch immer genau einen Besitzer, wobei eine Person beliebig viele Wohnungen besitzen kann. Des Weiteren existieren *Laden* und *Gilden-Haus* als Gebäude. Ein Laden hat einen Namen, einen Verkäufer und hat eine Liste von *Waren*, die dieser anbietet. Waren werden dabei nicht einzeln betrachtet, sondern als Konzept. Folglich existieren zum Beispiel nicht n viele Äpfel, sondern nur die Ware Apfel, wobei z.B. 2 Läden diese führen. Die Menge der Ware wird dabei in der Relation abgebildet. Eine *Gilde* ist dabei ein Zusammenschluss aus mindestens 100 Personen, mindestens einem Gilden-Haus und einem *Beruf*. Berufe sind dabei Instanzen, die einen Namen, Personen, die den Beruf ausüben und ein durchschnittliches Gehalt haben.

1. Modellieren Sie nun ein Datenbank-Schema in einem ER-Diagramm, welches die beschriebene Welt darstellen kann. Beachten Sie dabei die verschiedenen Relationen, Notationen, Schlüssel und funktionale Abhängigkeiten. Beschreiben Sie ihr Model und begründen Sie kurz verschiedene Design-Entscheidungen. Das Diagramm soll die Funktionalitäts- und min-max Notation beinhalten.

2. Überführen Sie Ihr Schema in ein Relationen-Schema. Beachten Sie dabei Attribute-Typen. Analysieren Sie Ihr Model nach ggf. vorhandenen Anomalien. Handeln Sie jeweils mindestens eine Lösch-Anomalie, Update-Anomalie und Einfüge-Anomalie ab. Falls ihr Modell keine Anomalien aufweist, nennen Sie Beispiele, wie diese aussehen könnten.

3. Normalisieren Sie Ihr Modell in die dritte Normalform. Dokumentieren Sie Ihre Schritte, wie Sie dabei vorgehen. Beschreiben Sie, warum eine fehlende Normalisierung ein Problem darstellen könnte. Ist ihr Modell bereits normalisiert, beschreiben Sie warum und wie Sie dabei vorgegangen sind.

Szenario Fantasy-Welt (für Aufgaben 2–3)

Um spätere Tests zu ermöglichen und die Einheitlichkeit der Datenbanken zu gewährleisten, wird für die Aufgaben 2 - 3 ein Schema vorgegeben.

Beschreibung Gegeben ist eine Fantasy-Welt, in welcher verschiedene Avatare leben. Ein Avatar hat einen Namen, ein Geburtsdatum, einen Rang, eine bevorzugte Waffe, besitzt eine bestimmte Menge an Geld und hat ein oder mehrere Werte, die seine Kampfkraft widerspiegeln. Da die Avatare Ränge haben, können Avatare sich auch untereinander duellieren, um Ränge aufzusteigen. Ein Avatar kann sich nicht selber duellieren. Ein Avatar kann an diversen Orten sein. Zum Beispiel kann er in einem Dungeon sein, welcher einen Namen, eine Adresse und einen Schwierigkeitsgrad hat. Alternativ kann er auch in einem Laden befinden, welcher einen Namen, Besitzer und eine Adresse hat. Des Weiteren existieren in der Welt Items, die einen Namen, diverse Eigenschaften aufweisen können und einen Geldwert besitzen. Die Items können im Dungeon, beim Avatar oder im Shop aufgefunden werden. Alles, was Items besitzen kann, kann auch Geld besitzen. Weiterhin existieren auch verschiedene Haustiere in dieser Welt. Ein Avatar wird dadurch definiert, dass es genau ein Haustier als Partner hat. Zusammen bilden Sie ein Team. Herrenlose Haustiere existieren nicht. Haustiere haben einen Namen, eine Rasse, eine Kampfkraft und einen Niedlichkeitsfaktor. Zwischen dem Avatar und seinem Haustier existiert eine prozentuale Affinität.

Relationales Schema & ER-Diagramm

- Besitzer(BesitzerID, Name, Geld)
- Ort(BesitzerID, Adresse)
- Shop(BesitzerID, LadenBesitzer)
- Dungeon(BesitzerID, Schwierigkeitsgrad)
- Avatar(BesitzerID, Stärke, Magie, Geschwindigkeit, Rang, WaffenPref, Geburtsdatum, Geburtsort, IstIn)
- Eigenschaften(EigenschaftenID, Beschreibung)
- Item(ItemID, Name, Geldwert, Besitzer)
- EigenschaftenBesitzen(ItemID, EigenschaftenID)
- Duellieren(AID1, AID2)
- Haustier(HaustierID, Name, Kampfkraft, Rasse, Niedrlichkeitsfaktor)
- Team(BesitzerID, HaustierID, Affinität)

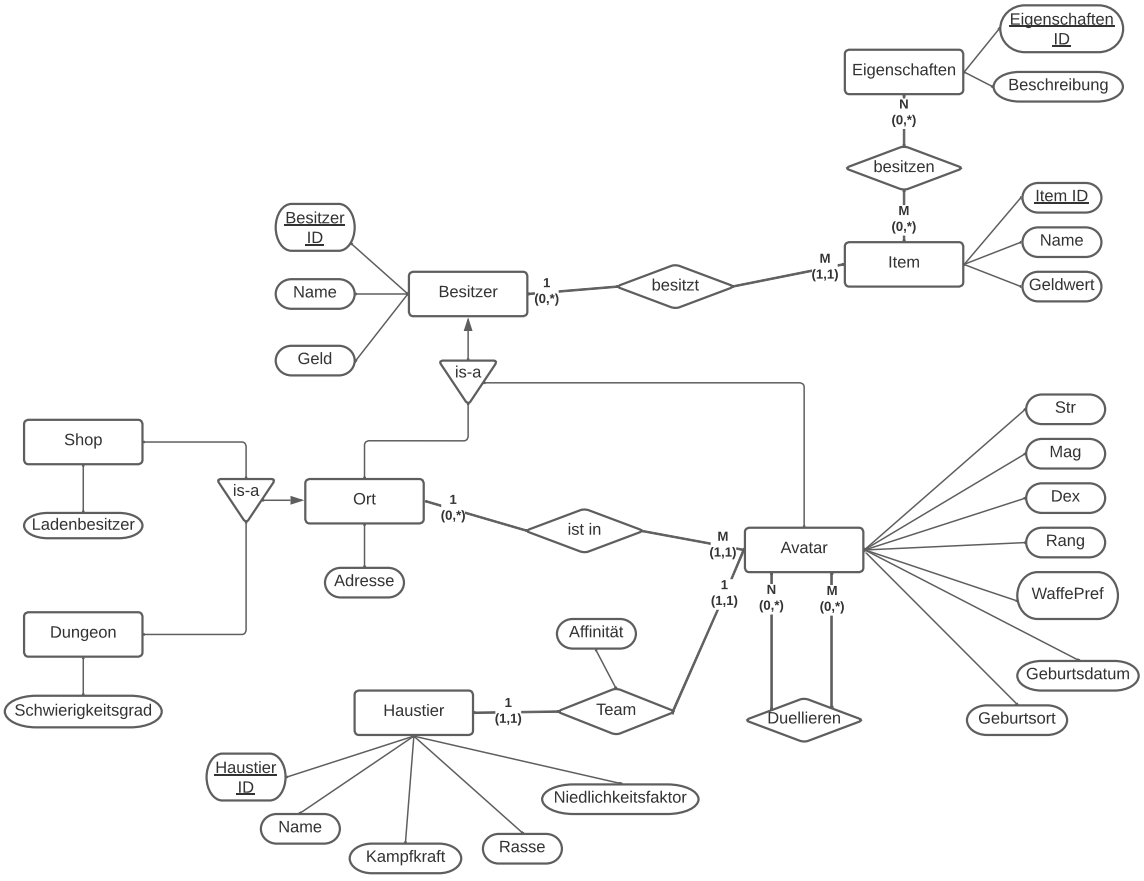

Abbildung 1: ER-Diagramm für die Aufgaben 2–3 (Modellierung einer Fantasy-Welt).

Aufgabe 2. (Algebra und Kalküle (27 Punkte))
Geben Sie folgende Anfragen in relationaler Algebra an:

1. Geben Sie eine List aus, mit allen Rassen von existierenden Haustieren.
2. Geben Sie jede Avatar ID aus, welche sich im Dungeon mit dem Namen ”Datenbanksysteme” befindet.
3. Gib jedes Avatar-ID-Paar aus, welches sich noch nicht duelliert hat.

Geben Sie folgende Anfragen im Tupelkalkül an:

1. Selektieren Sie ein Tripel, welches aus Item-Name, Avatar-Name und Shop-Name besteht, in welchem der Avatar und der Shop beide ein gleichnamiges Item besitzen, der Geldwert im Shop des Items jedoch größer ist, als beim Item des Avatars.
2. Selektieren Sie alle Eigenschaften, wo jedes Item mit dieser Eigenschaft einen Geldwert von größer 1000 hat.
3. Selektieren Sie jeden Dungeon, in welchem jeder sich vor Ort befindenden Avatar nicht das Item ”Datenbanksysteme-Schein” besitzt, es jedoch im Dungeon enthalten ist.

Geben Sie folgende Anfragen im Domänenkalkül an:

1. Geben Sie jede Avatar ID aus, welcher zu seinem Haustier-Partner eine Affinität größer 80% hat.
2. Geben Sie jede Avatar ID aus, der sich mit dem Avatar ”Atzmüller” duelliert hat und nicht das Item ”Datenbanksysteme-Schein” besitzt.
3. Geben Sie die Namen aller Haustiere aus, mit einer Kampfstärke größer 9000 und einem Niedlichkeitsfaktor von mindestens 80% an. Nutzen Sie dabei keine $\exists$ Quantifizierung.

Aufgabe 3. (SQL, Programmierung (61 Punkte))
Im folgenden werden Sie zunächst SQL-Anfragen erstellen, und diese dann auch in eine Implementierung einbauen. Achten Sie bei der Formatierung der SQL-Anfragen im Programmcode auf gute Lesbarkeit.

1. Erstellen Sie eine lauffähige Python Datei, welches für eine gegebene MariaDB Datenbank-Verbindung Ihres Schemas erstellt und mit Beispielinstanzen füllt. Implementieren Sie hierfür die Schnittstellen der *DatabaseProject* Klasse aus der *dbs2023ExerciseProject.py* Datei. Beachten Sie, dass das Verändern der Signaturen nicht erlaubt ist. Stellen Sie durch Integritätsbedingungen sicher, dass nur sinnvolle Eingaben erlaubt sind. Achten Sie auf eine korrekte Handhabung der Verbindung, ggf. mit aussagekräftigen Fehlermeldungen. Weiterhin sind nur folgende nicht Default Python-Packages zulässig: *mariadb*, *sys*, *getpass*.

   Wichtig: Das *mariadb*-Package unterstützt keine Erzeugung von Datenbanken. Hierfür müssten Sie eine entsprechende Datenbank vor dem Verbindungsaufbau erzeugen.

2. Überführen Sie im Laufe dessen die Anfragen aus Teil 2 in SQL-Abfragen und implementieren Sie die entsprechenden Methoden der Klasse *DatabaseProject* aus. Die Methoden sind nach einem Schema benannt, z.B. *doExerciseRA1* für die erste Anfrage (Aufgabe 2.1) – siehe Beschreibungen in den Kommentaren.

   Hinweis: Achten Sie bei der Formatierung der SQL-Anfragen im Programmcode auf gute Lesbarkeit.

   Wichtig: Sorgen Sie dafür, dass zu jeder Anfrage entsprechende Instanzen existieren und die Rückgabemenge weder leer ist, noch aus allen Instanzen der Tabelle bestehen.

3. Implementieren Sie weiterhin noch folgende Anfragen mittels SQL sowie der in der Klasse *DatabaseProject* vorgegebenen Schnittstellen bzw. Methoden, die Sie entsprechend ausimplementieren. Die Methoden sind wiederum nach einem bestimmten Schema benannt, z.B. *doExerciseSQL1* für die Anfrage aus Teilaufgabe 3.1. Achten Sie bei der Formatierung der SQL-Anfragen im Programmcode wiederum auf gute Lesbarkeit.

   a) Geben Sie die Waffe wieder, welche gerade am häufigsten bevorzugt ist.

   b) Geben Sie die durchschnittliche Niedlichkeit aller Haustiere aus, die in einem Team mit einem Avatar sind, welcher gerade weniger als 500 Geld hat und sich gleichzeitig in einem Dungeon befindet, der ein Item beinhaltet, der in keinem Laden vorhanden ist.

   c) Geben Sie aus, wie viele Avatare das Item ”Datenbanksysteme-Schein” besitzen, geteilt durch die Anzahl aller ”Datenbanksysteme-Schein”-Items die existieren.

   d) Geben Sie jeden Avatar aus, der nach 1.1.2000 geboren wurden. Absteigend sortiert nach Geburtsdatum.

   e) Ändern Sie den Ort von jedem Avatar, welcher sich in einem Laden befindet und weniger als 100 Geld hat zum Dungeon ”Arbeitswelt”.

   f) Löschen Sie alle Avatare, die sich mit dem Avatar ”Prüfungsamt” duelliert haben und dreimal das Item ”Fehlversuch” haben.

   g) Erstellen Sie eine Sicht für einen Dieb, die dem Dieb nur erlaubt den aufsummierten durchschnittlichen Geld-Besitzt (Besessenes Geld + Geldwert aller Items) in 500er-Schritten eines Besitzers zu sehen, jedoch nicht die genauen Items.

   h) Erstellen Sie einen Trigger, welcher einen Avatar zum Dungeon ”Arbeitswelt” verschiebt, wenn dieser sich im Dungeon ”Datenbanksysteme” befindet und das Item ”Datenbanksysteme-Schein” besitzt.

   Wichtig: Sorgen Sie dafür, dass zu jeder Anfrage entsprechende Instanzen existieren und die Rückgabemenge weder leer ist, noch aus allen Instanzen der Tabelle bestehen.

```python
# Example use in for example jupyter notebook:
# from dbs2023ExerciseProject import DatabaseProject
# dbsp = DatabaseProject()
# dbsp.connectToMariaDB('root', getpass.getpass(), 'localhost', 3306, 'dbName')
# dbsp.createProjectDBTables()
# ...

#NOTE: Don't change the interfaces.
class DatabaseProject():

    # connect to mariaDB. Handle connection object with class variable self.connection#
    # NOTE: Database should be created beforehand because the mariadb package does not support the CREATE DATABASE statement!
    def connectToMariaDB(self, user: str, pw, host: str, port: int, database: str):
        pass

    #disconnect from mariaDB
    def disconnect(self):
        pass

    # create all tables for the project.
    def createProjectDBTables(self):
        pass

    # delete all project related tables
    def deleteAllProjectTables(self):
        pass

    #yes handling IDs manually is not optimal, however for our little project and for testing this should be managable!
    # Implement all create and delete methods for each entity
    def createEigenschaft(self, itemID: int, name: str):
        pass

    def createItem(self, eigenschaftenID: int, name: str, geldwert: int, besitzerID: int):
        pass

    def createEigenschaftenBesitzen(self, itemID: int, eigenschaftenID: int):
        pass

    def createShop(self, besitzerID: int, name: str, geld: int, adresse: str, ladenBesitzer: str):
        pass

    def createDungeon(self, besitzerID: int, name: str, geld: int, adresse: str, schwierigkeitsgrad: int):
        pass

    def createTeam(self, besitzerID: int, avatarName: str, geld: int, stärke: int, magie: int, geschwindigkeit: int, rang: int, waffenPref: str, geburtsdatum: str, geburtsort: str, istIn: int, affinität: int, haustierID: int, haustierName: str, kampfkraft: int, rasse: str, niedichkeitsfaktor: float):
        pass

    def createDuellieren(self, avatar1: int, avatar2: int):
        pass

    def deleteEigenschaft(self, id: int):
        pass
```

```
53
54      def deleteItem(self, id: int):
55          pass
56
57      def deleteEigenschaftenBesitzen(self, itemId: int, eigenschaftenID: int):
58          pass
59
60      def deleteShop(self, id: int):
61          pass
62
63      def deleteDungeon(self, id: int):
64          pass
65
66      def deleteTeam(self, besitzerID: int, haustierID: int):
67          pass
68
69      def deleteDuellieren(self, besitzerID1: int , besitzerID2: int):
70          pass
71
72      # Befüllt die Tabellen mit den Instanzen ihrer Wahl um die SQL-Abfragen zu testen
73      def createWorld(self):
74          pass
75
76      # Geben Sie eine List aus, mit allen Rassen von existierenden Haustieren.
77      def doExerciseRA1(self):
78          pass
79
80      # Geben Sie jede Avatar ID aus, welche sich im Dungeon mit dem Namen
        ↪  "Datenbanksysteme" befindet.
81      def doExerciseRA2(self):
82          pass
83
84      # Gib jedes Avatar-ID-Paar aus, welches sich noch nicht duelliert hat.
85      def doExerciseRA3(self):
86          pass
87
88      # Selektieren Sie ein Tripel, welches aus Item-Name, Avatar-Name und Shop-Name
        ↪  besteht, in welchem der Avatar und der Shop beide ein gleichnamiges Item
        ↪  besitzen, der Geldwert im Shop des Items jedoch größer ist, als beim Item des
        ↪  Avatars.
89      def doExerciseTK1(self):
90          pass
91
92      # Selektieren Sie alle Eigenschaften, wo jedes Item mit dieser Eigenschaft einen
        ↪  Geldwert von größer 1000 hat.
93      def doExerciseTK2(self):
94          pass
95
96      # Selektieren Sie jeden Dungeon, in welchem jeder sich vor Ort befindenden Avatar
        ↪  nicht das Item ''Datenbanksysteme-Schein'' besitzt, es jedoch im Dungeon
        ↪  enthalten ist.
97      def doExerciseTK3(self):
98          pass
99
100     # Geben Sie jede Avatar ID aus, welcher zu seinem Haustier-Partner eine Affinität
        ↪  größer 80% hat.
101     def doExerciseDK1(self):
102         pass
103
104     # Geben Sie jede Avatar ID aus, der sich mit dem Avatar ''Atzmüller'' duelliert
        ↪  hat und nicht das Item ''Datenbanksysteme-Schein'' besitzt.
```

```
105     def doExerciseDK2(self):
106         pass
107 
108     # Geben Sie die Namen aller Haustiere aus, mit einer Kampfstärke größer 9000 und
        ↪  einem Niedlichkeitsfaktor von mindestens 80% an. Nutzen Sie dabei keine
        ↪  Exists Quantifizierung.
109     def doExerciseDK3(self):
110         pass
111 
112     # Geben Sie die Waffe wieder, welche gerade am häufigsten bevorzugt ist.
113     def doExerciseSQL1(self):
114         pass
115 
116     # Geben Sie die durchschnittliche Niedlichkeit aller Haustiere aus, von den
        ↪  Avataren, die gerade mit weniger als 500 Geld, die gleichzeitig in einem
        ↪  Dungeon sind, welcher ein Item beinhaltet, der in keinem Laden vorhanden ist.
117     def doExerciseSQL2(self):
118         pass
119 
120     # Geben Sie aus, wie viele Avatare das Item ''Datenbanksysteme-Schein'' besitzen,
        ↪  geteilt durch die Anzahl aller ''Datenbanksysteme-Schein''-Items die
        ↪  existieren.
121     def doExerciseSQL3(self):
122         pass
123 
124     # Geben Sie jeden Avatar aus, der nach 1.1.2000 geboren wurden. Absteigend
        ↪  sortiert nach Geburtsdatum.
125     def doExerciseSQL4(self):
126         pass
127 
128     # Ändern Sie den Ort von jedem Avatar, welcher sich in einem Laden befindet und
        ↪  weniger als 100 Geld hat zum Dungeon ''Arbeitswelt''.
129     def doExerciseSQL5(self):
130         pass
131 
132     # Löschen Sie alle Avatare, die sich mit dem Avatar ''Prüfungsamt'' duelliert
        ↪  haben und dreimal das Item ''Fehlversuch'' haben.
133     def doExerciseSQL6(self):
134         pass
135 
136     # Erstellen Sie eine Sicht für einen Dieb, die dem Dieb nur erlaubt den
        ↪  aufsummierten durchschnittlichen Geld-Besitzt (Besessenes Geld + Geldwert
        ↪  aller Items) in 500er-Schritten eines Besitzers zu sehen, jedoch nicht die
        ↪  genauen Items.
137     def doExerciseSQL7(self):
138         pass
139 
140     # Erstellen Sie einen Trigger, welcher einen Avatar zum Dungeon ''Arbeitswelt''
        ↪  verschiebt, wenn dieser sich im Dungeon ''Datenbanksysteme'' befindet und das
        ↪  Item ''Datenbanksysteme-Schein'' besitzt.
141     def doExerciseSQL8(self):
142         pass
```