Prof. Dr. Stefan Decker, PD Dr. Ralf Klamma
K. Fidomski, M. Slupczynski, S. Welten

17.05.2021

Datenbanken und Informationssysteme (Sommersemester 2021)

Übung 4

Abgabe bis 31.05.2021 14:00 Uhr.

Zu spät eingereichte Übungen werden nicht berücksichtigt.

Wichtige Hinweise

- Bei Nichtbeachtung dieser Hinweise wird die Abgabe mit 0 Punkten bewertet!
- Bitte reichen Sie Ihre Lösung nur in Dreier- oder Vierergruppen ein.
- Achten Sie auch darauf, dass Ihre Gruppe im Moodle korrekt eingerichtet ist.
- **Bitte laden Sie Ihren schriftlichen Teil der Lösungen ins Moodle als ein zusammenhängendes PDF-Dokument hoch. Benutzen Sie dafür die entsprechend markierte Abgabefunktion.**
- **Bitte bearbeiten Sie den digitalen Teil dieses Übungsblatts in diesem Notebook. Nach der Bearbeitung laden Sie bitte dieses Notebook ins Moodle hoch. Benutzen Sie dafür die entsprechend markierte Abgabefunktion.**
- Bitte geben Sie Namen, Matrikelnummern und Moodle-Gruppennummer auf der schriftlichen Lösung an.
- Wird offensichtlich die gleiche Lösung von zwei Gruppen abgegeben, dann erhalten beide Gruppen 0 Punkte.

Die Lösung zu diesem Übungsblatt wird in den Übungen am 31. Mai und 02. Juni 2021 vorgestellt. Bitte beachten Sie auch die aktuellen Ankündigungen im Moodle-Lernraum zur Vorlesung. * bezeichnet Bonusaufgaben.

Nummer der Abgabegruppe: [124]

Gruppenmitglieder: [Andrés Montoya, 405409], [Marc Ludevid, 405401], [Til Mohr, 405959]

Vergessen Sie nicht alle Gruppenmitglieder einzutragen!
Der Bearbeitungsmodus kann mit Doppelklick aktiviert und mit der Tastenkombination **Strg+Enter** beendet werden.

Ausdrücke der relationalen Algebra in Markdown mittels LaTeX

Möchten Sie die folgenden Übungsaufgaben über den JupyterHub bearbeiten, können Sie die Informationsbedürfnisse im Tupelkalkül und Domänenkalkül in Markdown mittels LaTeX niederschreiben. Die Ausdrücke können Sie mit den entsprechenden, nachfolgend beschriebenen, LaTeX-Befehlen erzeugen.

Mit **Doppelklick** auf diese Zelle können Sie die Befehle einsehen.

- Existenzquantor: $\exists$
- Allquantor: $\forall$
- Implikation: $\rightarrow$
- Element: $\in$
- Logisches Und: $\wedge$
- Logisches Oder: $\vee$
- Logische Negation: $\neg$
- Vergleichsoperatoren: $<, >, \leq, \geq, =, \neq$
- Für Schlüssel: $\underline{Unterstrichen}$
- A ist eine Teilmenge von B: A $\subseteq$ B
- A ist eine echte Teilmenge von B: A $\subset$ B
- Senkrechter Strich für die Megendefinition: $\mid$
- Mengenklammern: { }

Beachten Sie: Mit **$** werden mathematische Ausdrücke in LaTeX in Markdown eingefasst.

Aufgabe 4.1 (Tupelkalkül) - Schriftlich

(4 + 2* Punkte)

WICHTIG:

Aufgabe 4.1 und Aufgabe 4.2 sind **schriftlich** zu bearbeiten und werden manuell bewertet.
Die Lösungen der Übungsaufgaben müssen in einem zusammenhängenden .pdf Dokument abgegeben werden.

Gegeben sei das folgende relationale Datenbankschema:

Relationen:

$$KuenstlerIn(\underline{Vorname}, \underline{Nachname}, Geburtsjahr)$$
$$Kunststil(\underline{Name}, Startjahr, Endjahr)$$
$$Werk(\underline{Titel}, Erscheinungsjahr, Vorname, Nachname, Stil)$$

Interrelationale Abhängigkeiten:

$$Werk[Vorname] \subseteq KuenstlerIn[Vorname]$$
$$Werk[Nachname] \subseteq KuenstlerIn[Nachname]$$
$$Werk[Stil] \subseteq Kunststil[Name]$$

Formulieren Sie die folgenden Informationsbedürfnisse im Tupelkalkül. Verwenden Sie dazu die in der Vorlesung vorgestellte Notation:

a) Titel der Werke, die dem Stil 'Expressionismus' zugeordnet werden und von KuenstlerInnen stammen, die in den 1860er Jahren geboren wurden.

b) Name, Startjahr und Endjahr der Kunststile, die später als der Kunststil mit dem Namen 'Impressionismus' begonnen haben.

c) Vorname und Nachname von dem/der ältesten KuenstlerIn, der/die mindestens ein Werk geschaffen hat, welches dem Kunststil mit Namen 'Barock' zugeordnet wird. Es kann mehrere älteste KuenstlerInnen geben. *(Bonusaufgabe)*

Formulieren Sie das durch Tupelkalkül dargestellte Informationsbedürfnis in natürlicher Sprache:

d) $\{[w.Titel, w.Erscheinungsjahr] \mid w \in Werk$
$\wedge \exists ks \in Kunststil\ (ks.Name = w.Stil$
$\wedge\ ks.Endjahr < w.Erscheinungsjahr)\}$

a) $\{[w.Titel] \mid w \in Werk \wedge w.Stil =' Expressionismus' \wedge \exists k \in KuenstlerIn(w.Vorname = k.Vorname \wedge w.Nachname = k.Nachname$
$\wedge\ k.Geburtsjahr \geq 1860 \wedge k.Geburtsjahr < 1870)\}$
b) $\{s \mid s \in Kunststil \wedge \exists a \in Kunststil(a.Name =' Impressionismus' \wedge a.Startjahr < s.Startjahr)\}$
c) $\{[k.Vorname, k.Nachname] \mid k \in KuenstlerIn \wedge \exists w \in Werk(w.Vorname = k.Vorname \wedge w.Nachname = k.Nachname \wedge w.Stil$
$=' Barock' \wedge \forall a \in KuenstlerIn((a \neq k \wedge \exists v \in Werk(v.Vorname = a.Vorname \wedge v.Nachname = a.Nachname \wedge v.Stil =' Barock'))$
$\rightarrow (k.Geburtsjahr \leq a.Geburtsjahr)))\}$
d) Titel und Erscheinungsjahr von den Werken, die nach dem Ende ihres Kunststils erschienen sind.

Aufgabe 4.2 (Domänenkalkül) - Schriftlich

(6 + 2* Punkte)

WICHTIG:

Aufgabe 4.1 und Aufgabe 4.2 sind **schriftlich** zu bearbeiten und werden manuell bewertet.
Die Lösungen der Übungsaufgaben müssen in einem zusammenhängenden .pdf Dokument abgegeben werden.

Bitte beachten Sie das folgende relationale Datenbankschema eines Festivals:

Relationen:

$$Song(\underline{TrackID}, Name, Genre, Dauer)$$
$$Buehne(\underline{BuehnenName}, ZuschauerKapazitaet, Groesse)$$
$$Band(\underline{BandName}, AnzahlBandMitglieder, JahreAktiv)$$
$$spielt(\underline{TrackID}, \underline{BuehnenName}, \underline{BandName}, \underline{Datum}, \underline{VonUhrzeit}, \underline{BisUhrzeit})$$

Interrelationale Abhängigkeiten:

$$spielt[TrackID] \subseteq Song[TrackID]$$
$$spielt[BuehnenName] \subseteq Buehne[BuehnenName]$$
$$spielt[BandName] \subseteq Band[BandName]$$

Formulieren Sie die folgenden Informationsbedürfnisse im Domänenkalkül. Verwenden Sie dazu die in der Vorlesung vorgestellte Notation:

a) TrackID und Genre aller Songs, welche von der Band 'BillyTalent' gespielt worden sind.

b) BuehnenName, ZuschauerKapazitaet aller Bühnen und BandName aller Bands, welche auf diesen Bühnen gespielt haben, 5 Jahre aktiv sind und Songs aus dem Genre 'Schlager' spielen.

c) TrackID aller Songs, welche eine Dauer von mindestens fünf Minuten haben (>= 5) und noch nie auf einer Bühne gespielt wurden.

d) Alle BandNamen-Paare (zum Beispiel (Band1, Band2)), welche mindestens ein Mal gleichzeitig auf zwei verschiedenen Bühnen gespielt haben (Datum gleich, VonUhrzeit gleich, BisUhrzeit ist beliebig). 'Gleichzeitig' bedeutet also hier, dass die Bands zur selben Uhrzeit beginnen zu spielen. Beachten Sie, dass Band1 und Band2 unterschiedlich sein müssen, um ein valides Paar zu bilden.

e) Die TrackID des Songs mit dem Namen 'Holzmichl' und den BandNamen der Bands, welche diesen Song ('Holzmichl') mindestens drei Mal auf der Bühne mit dem Namen 'Konzertmuschel' gespielt haben. *(Bonusaufgabe)*

a) $\{tid, gen \mid \exists nam, dau(Song(tid, nam, gen, dau) \wedge \exists bue, band, dat, von, bis(spielt(tid, bue, band, dat, von, bis) \wedge band =' BillyTalent'))\}$
b) $\{bue, zus, ban \mid \exists gro(Buehne(bue, zus, gro) \wedge \exists abm, jaa(Band(ban, abm, jaa) \wedge jaa = 5 \wedge \exists tid, dat, von, bis(spielt(tid, bue, ban, dat, von, bis))))\}$
c) $\{tid \mid \exists nam, gen, dau(Song(tid, nam, gen, dau) \wedge dau \geq 5 \wedge \neg\exists bue, ban, dat, von, bis(spielt(tid, bue, band, dat, von, bis)))\}$
d) $\{band1, band2 \mid band1 \neq band2 \wedge \exists tid1, tid2, bue1, bue2, dat, von, bis1,$
$bis2(spielt(tid1, bue1, band1, dat, von, bis1) \wedge spielt(tid2, bue2, band2, dat, von, bis2) \wedge bue1 \neq bue2)\}$
e) 3-mal spielen: Entweder am selben Datum zu unterschiedlichen Zeiten (unterschiedliche vonUhrzeit reicht hier) oder an verschiedenen Tagen.

$\{tid, band \mid \exists nam, gen, dau(Song(tid, nam, gen, dau) \wedge nam =' Holzmichl' \wedge \exists bue, dat1, dat2, dat3, von1, von2, von3, bis1, bis2,$
$bis3(spielt(tid, bue, band, dat1, von1, bis1) \wedge spielt(tid, bue, band, dat2, von2, bis2) \wedge spielt(tid, bue, band, dat3, von3, bis3) \wedge bue$
$=' Konzertmuschel' \wedge ((dat1 = dat2) \rightarrow (von1 \neq von2) \wedge ((dat1 = dat3) \rightarrow (von1 \neq von3) \wedge ((dat2 = dat3) \rightarrow (von2 \neq von3))))\}$

Aufgabe 4.3 (SQL-Struktur) - Digital

(6 Punkte)

WICHTIG:

Aufgabe 4.3 ist **digital** zu bearbeiten und wird maschinell bewertet.
Die Lösungen der Übungsaufgaben müssen in Form dieses Notebooks abgegeben werden.

ACHTUNG! Die nachfolgenden Code Zellen sind Teil der Abgabe und werden automatisch bewertet.
Bitte füllen Sie Ihren SQL Code in die gekennzeichneten Stellen ein. Wir empfehlen die Aufgaben in chronologischer Reihenfolge zu bearbeiten.
Lesen Sie bitte jede Zelle und Hinweise sorgfältig durch und benennen Sie die Spalten exakt so wie es in der Aufgabenstellung beschrieben ist.

Sie wurden als Datenbankentwickler bei einer Firma eingestellt. Herzlichen Glückwunsch! Ihr neuer Chef übergibt Ihnen direkt am Tag Ihrer Einstellung den ersten Arbeitsauftrag, welcher den Start der Datenbank beinhaltet. Sie sind sehr gehorsam, führen die nächsten Zellen aus und warten bis der Server **"Server OK! Es kann los gehen!"** ausgibt, damit Sie die nächsten Arbeitsaufträge entgegennehmen können.

Datenbank starten

Die folgende Zelle muss am Anfang der Bearbeitung ausgeführt werden.
Sie startet den Server und erstellt die Datenbank.
Außerdem überprüft sie, ob der Server Verbindungen annimmt.

Diese Zelle ist auch verantwortlich dafür, dass `%sql` und `%%sql` Befehle ausgeführt werden können.

Sie sollten die weiteren Zellen erst ausführen, wenn **"Server OK! Es kann los gehen!"** am Ende der Ausgabe der nächsten Zelle steht.
Andere Ausgaben dieser Zelle können ignoriert werden.
Wenn der Server nicht starten kann, versuchen Sie, den Kernel neu zu starten und versuchen Sie es dann erneut.

WICHTIG:

Falls Sie die Datenbank resetten möchten, um Inkonsistenzen zu vermeiden, können Sie einfach die Serverinitialisierung erneut starten. Der Server löscht alles und sollte nach jedem Neustart eine leere Datenbank beinhalten.

In [27]:
```
from IPython.display import Markdown, display
path = "assets/_pgdata"
try:
    running_tests
except NameError:
    import os.path
    if not os.path.exists(path):
        display(Markdown("# Datenbank wird initialisiert."))
        display(Markdown("### Datenbank wird extrahiert."))
        !tar -zx --touch --checkpoint=.50 -f assets/pgdata.tar.gz -C assets/
        display(Markdown("### Datenbank initialisiert"))
    !chmod 700 $path
    display(Markdown("# Server wird (neu)gestartet."))
    if os.path.exists(path + "/postmaster.pid"):
        !pg_ctl -D $path restart
        display(Markdown("### Datenbank restart OK"))
    else:
        !pg_ctl -D $path start
        display(Markdown("### Datenbank start OK"))
display(Markdown("#### Einrichtung der Übungsdatenbank"))
!psql -c "DROP DATABASE IF EXISTS exercise_sheet;" postgres
!psql -c "CREATE DATABASE exercise_sheet;" postgres
display(Markdown("#### Verbindung mit Datenbank wird hergestellt..."))
%reload_ext sql
%sql postgresql://localhost/exercise_sheet
%sql ABORT;--NOOP
check=!pg_isready -h localhost -d exercise_sheet -q;echo $?
if check[0]=='0':
    x = %sql SELECT 'OK' AS "status"
    display(x)
    display(Markdown("# Server OK! Es kann los gehen!"))
else:
    display(Markdown("# Server nicht OK! Versuche, den Kernel neu zu starten und die Zelle erneut auszuführen."))
```

Server wird (neu)gestartet.

```
waiting for server to shut down.... done
server stopped
waiting for server to start....2021-05-28 11:19:44.662 GMT [173] LOG:  could not open directory "/opt/conda/share/zo
neinfo": No such file or directory
2021-05-28 11:19:44.686 UTC [173] LOG:  starting PostgreSQL 12.3 on x86_64-conda-linux-gnu, compiled by x86_64-conda
-linux-gnu-cc (crosstool-NG 1.24.0.133_b0863d8_dirty) 7.5.0, 64-bit
2021-05-28 11:19:44.687 UTC [173] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2021-05-28 11:19:44.687 UTC [173] LOG:  could not bind IPv6 address "::1": Cannot assign requested address
2021-05-28 11:19:44.687 UTC [173] HINT:  Is another postmaster already running on port 5432? If not, wait a few seco
nds and retry.
2021-05-28 11:19:44.700 UTC [173] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
2021-05-28 11:19:44.818 UTC [174] LOG:  database system was shut down at 2021-05-28 11:19:44 UTC
2021-05-28 11:19:44.864 UTC [173] LOG:  database system is ready to accept connections
 done
server started
```

Datenbank restart OK

Einrichtung der Übungsdatenbank

```
DROP DATABASE
CREATE DATABASE
```

Verbindung mit Datenbank wird hergestellt...

```
 * postgresql://localhost/exercise_sheet
(psycopg2.errors.AdminShutdown) terminating connection due to administrator command
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.

[SQL: ABORT;--NOOP]
(Background on this error at: http://sqlalche.me/e/13/e3q8)
 * postgresql://localhost/exercise_sheet
1 rows affected.
```

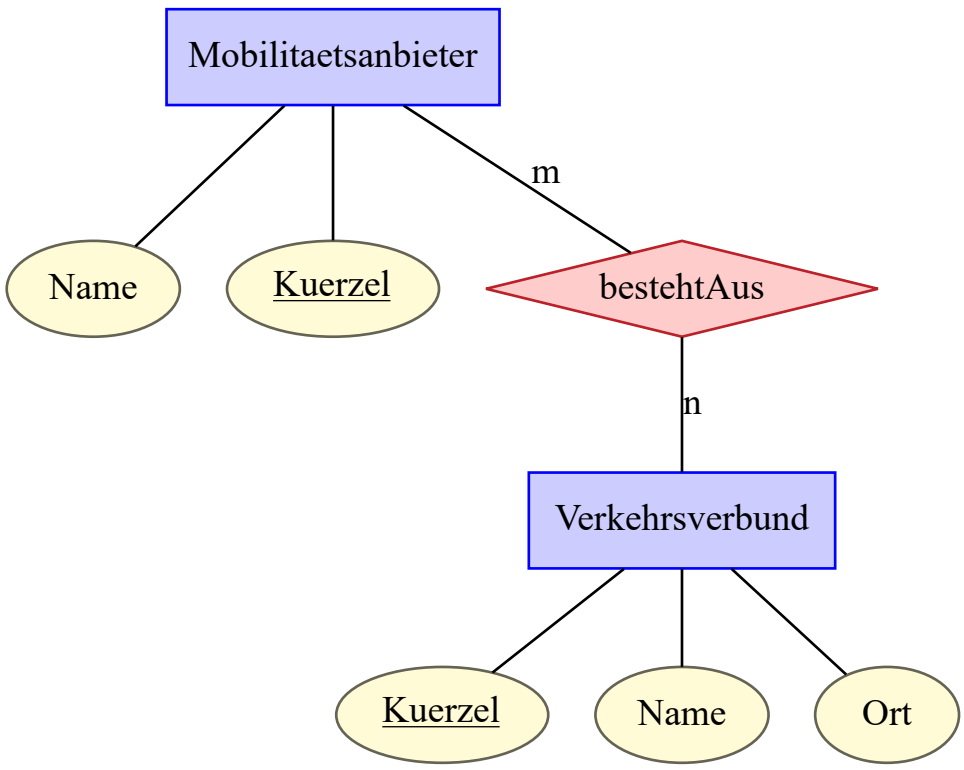
status
OK

Server OK! Es kann los gehen!

Falls der Server die gewünschte Ausgabe hat, warten Sie geduldig auf die nächste Arbeitsanweisung Ihres Chefs.... .... der selbstverständlich mitbekommen hat, dass Sie gerade fertig geworden sind und Ihnen direkt die nächste Aufgabe übergibt.

Nach einer Anforderungsanalyse mit einem Kunden Ihrer neuen Firma, ergab sich das folgende ER-Modell:

In [2]:
```
#Ein paar Vorkehrungen treffen: Import des ER-Tools. Anleitung siehe ER_Tutorial.ipynb
from assets.ER import ERDiagram
#Neues Diagramm initiieren
g = ERDiagram()


g.neuer_knoten("Verkehrsverbund")
g.neues_attribut("Verkehrsverbund", "Kuerzel", primaer=True)
g.neues_attribut("Verkehrsverbund", "Name")
g.neues_attribut("Verkehrsverbund", "Ort")


g.neuer_knoten("Mobilitaetsanbieter")
g.neues_attribut("Mobilitaetsanbieter", "Name")
g.neues_attribut("Mobilitaetsanbieter", "Kuerzel", primaer=True)


g.neue_relation("Mobilitaetsanbieter", "bestehtAus", "Verkehrsverbund", "m", "n")

g.display()
```



a) Ihre Aufgabe lautet nun, dieses ER-Modell in ein SQL Datenbankschema zu überführen. Geben Sie die verwendeten SQL-Befehle an. Verwenden Sie bitte die in der DBIS Vorlesung vorgestellten SQL-Standard-Befehle.

Sie erinnern sich zurück und Ihnen fällt folgende Schritt-für-Schritt Anleitung ein, um den Auftrag Ihres Chefs zu bearbeiten:

Um das ER-Modell in ein SQL Datenbankschema zu überführen, beginnen Sie zunächst mit der Erstellung einer 'Verkehrsverbund' Tabelle. Die drei Attribute haben den Typ varchar(255). Beachten Sie, dass die 'Verkehrsverbund' Tabelle einen Primärschlüssel hat und dieser auch erstellt werden muss. Benutzen Sie für die Erstellung die nächste Zelle.

In [28]:
```
%%sql
CREATE TABLE Verkehrsverbund
( Kuerzel varchar(255),
  Name varchar(255),
  Ort varchar(255),
  PRIMARY KEY (Kuerzel) )
```

```
 * postgresql://localhost/exercise_sheet
Done.
```

Out[28]:
```
[]
```

In [29]:
```
#VISIBLE TEST 0.5P
table = %sql SELECT EXISTS (SELECT FROM information_schema.tables WHERE table_name   = 'verkehrsverbund');
table = table.dict()
tabelleWurdeErstellt = table['exists'][0]
assert tabelleWurdeErstellt, "Die Tabelle Verkehrsverbund wurde nicht erstellt! Überprüfen Sie Ihren Code!"
```

```
 * postgresql://localhost/exercise_sheet
1 rows affected.
```

In [ ]:
```
#HIDDEN TEST 0.5P
```

Als nächsten Schritt wird die 'Mobilitaetsanbieter' Tabelle erstellt. Auch hier haben die Attribute den Typ varchar(255). Benutzen Sie für die Erstellung die nächste Zelle.

In [30]:
```
%%sql
CREATE TABLE Mobilitaetsanbieter
( Kuerzel varchar(255),
  Name varchar(255),
  PRIMARY KEY (Kuerzel) )
```

```
 * postgresql://localhost/exercise_sheet
Done.
```

Out[30]: []

In [31]:
```
#VISIBLE TEST 0.5P
table = %sql SELECT EXISTS (SELECT FROM information_schema.tables WHERE table_name   = 'mobilitaetsanbieter');
table = table.dict()
tabelleWurdeErstellt = table['exists'][0]
assert tabelleWurdeErstellt, "Die Tabelle Mobilitaetsanbieter wurde nicht erstellt! Überprüfen Sie Ihren Code!"
```

```
 * postgresql://localhost/exercise_sheet
1 rows affected.
```

In [ ]:
```
#HIDDEN TEST 0.5P
```

Als letzten Schritt müssen die beiden erstellten Tabellen mit einer 'bestehtAus' Tabelle verbunden werden, um die m:n-Beziehung korrekt in SQL abzubilden. Dazu erstellen Sie zunächst eine neue 'bestehtAus' Tabelle. Die Tabelle besteht **NUR** aus zwei Spalten mit den Namen 'vbKuerzel' (für Verkehrsverbund) und 'mKuerzel' (für Mobilitaetsanbieter) - beide haben den Typ varchar(255). Beide Spalten ergeben zusammen den Primärschlüssel der Tabelle 'bestehtAus'. Vergessen Sie nicht 'vbKuerzel' und 'mKuerzel' jeweils als Fremdschlüssel zu definieren.

In [32]:
```
%%sql
CREATE TABLE bestehtAus
( vbKuerzel varchar(255),
  mKuerzel varchar(255),
  PRIMARY KEY (vbKuerzel, mKuerzel),
  FOREIGN KEY (vbKuerzel) REFERENCES Verkehrsverbund(Kuerzel),
  FOREIGN KEY (mKuerzel) REFERENCES Mobilitaetsanbieter(Kuerzel) )
```

```
 * postgresql://localhost/exercise_sheet
Done.
```

Out[32]: []

In [33]:
```
#VISIBLE TEST 0.5P
table = %sql SELECT EXISTS (SELECT FROM information_schema.tables WHERE table_name   = 'bestehtaus');
table = table.dict()
tabelleWurdeErstellt = table['exists'][0]
assert tabelleWurdeErstellt, "Die Tabelle bestehtaus wurde nicht erstellt! Überprüfen Sie Ihren Code!"
```

```
 * postgresql://localhost/exercise_sheet
1 rows affected.
```

In [ ]:
```
#HIDDEN TEST 0.5P
```

In [ ]:
```
#HIDDEN TEST 0.5P
```

Ihr Chef bekommt mit, dass Sie sehr effizient arbeiten und bittet Sie direkt um einen weiteren Gefallen. Ihr Chef bittet Sie die folgenden Zellen auszuführen und somit neue Tabellen in die Datenbank zu schreiben. Die Tabellen entstammen ebenfalls aus einem Kundengespräch.

In [34]:
```
%%sql
CREATE TABLE Patient (
        KVNummer INTEGER PRIMARY KEY,
        Name varchar(255),
        Adresse varchar(255),
        PraxisWebsite varchar(255)
);
```

```
 * postgresql://localhost/exercise_sheet
Done.
```

Out[34]: []

In [35]:
```
%%sql
CREATE TABLE Arzt (
    ZulassungsNr varchar(255) PRIMARY KEY,
    Name varchar(255)
);
```

```
 * postgresql://localhost/exercise_sheet
Done.
```

Out[35]: []

In [36]:
```
#VISIBLE TEST
table = %sql SELECT EXISTS (SELECT FROM information_schema.tables WHERE table_name   = 'patient');
table = table.dict()
tabelleWurdeErstellt = table['exists'][0]
assert tabelleWurdeErstellt, "Die Tabelle Patient wurde nicht erstellt! Lassen Sie die Zelle oben unberührt und führen Sie die Zelle oben aus!"
```

```
 * postgresql://localhost/exercise_sheet
1 rows affected.
```

In [37]:
```
#VISIBLE TEST
table = %sql SELECT EXISTS (SELECT FROM information_schema.tables WHERE table_name   = 'arzt');
table = table.dict()
tabelleWurdeErstellt = table['exists'][0]
assert tabelleWurdeErstellt, "Die Tabelle Arzt wurde nicht erstellt! Lassen Sie die Zelle oben unberührt und führen Sie die Zelle oben aus!"
```

```
 * postgresql://localhost/exercise_sheet
1 rows affected.
```

Ihnen fällt auf, dass bei der 'Patient' Tabelle ein/e 'PraxisWebsite' Attribut/Spalte eingefügt wurde. Sie denken, es handelt sich um einen Fehler und fragen Ihren Chef. Ihr Chef gesteht Ihnen, dass ihm beim Design der beiden Tabellen ein Fehler unterlaufen ist. 'PraxisWebsite' gehört eigentlich zur Tabelle 'Arzt' und nicht zur 'Patient' Tabelle. Er bittet Sie seinen Fehler zu berichtigen:

b) Löschen Sie 'PraxisWebsite' aus der 'Patient' Tabelle und fügen Sie eine neue Spalte 'PraxisWebsite' (gleicher Typ) zur Tabelle 'Arzt' hinzu. Ihr Chef weist Sie darauf hin, dass Sie nur 'ALTER TABLE' benutzen dürfen. 'DROP TABLE' ist nicht zulässig.

In [38]:
```
%%sql
ALTER TABLE Patient DROP COLUMN PraxisWebsite; ALTER TABLE Arzt ADD PraxisWebsite varchar(255)
```

```
 * postgresql://localhost/exercise_sheet
Done.
Done.
```

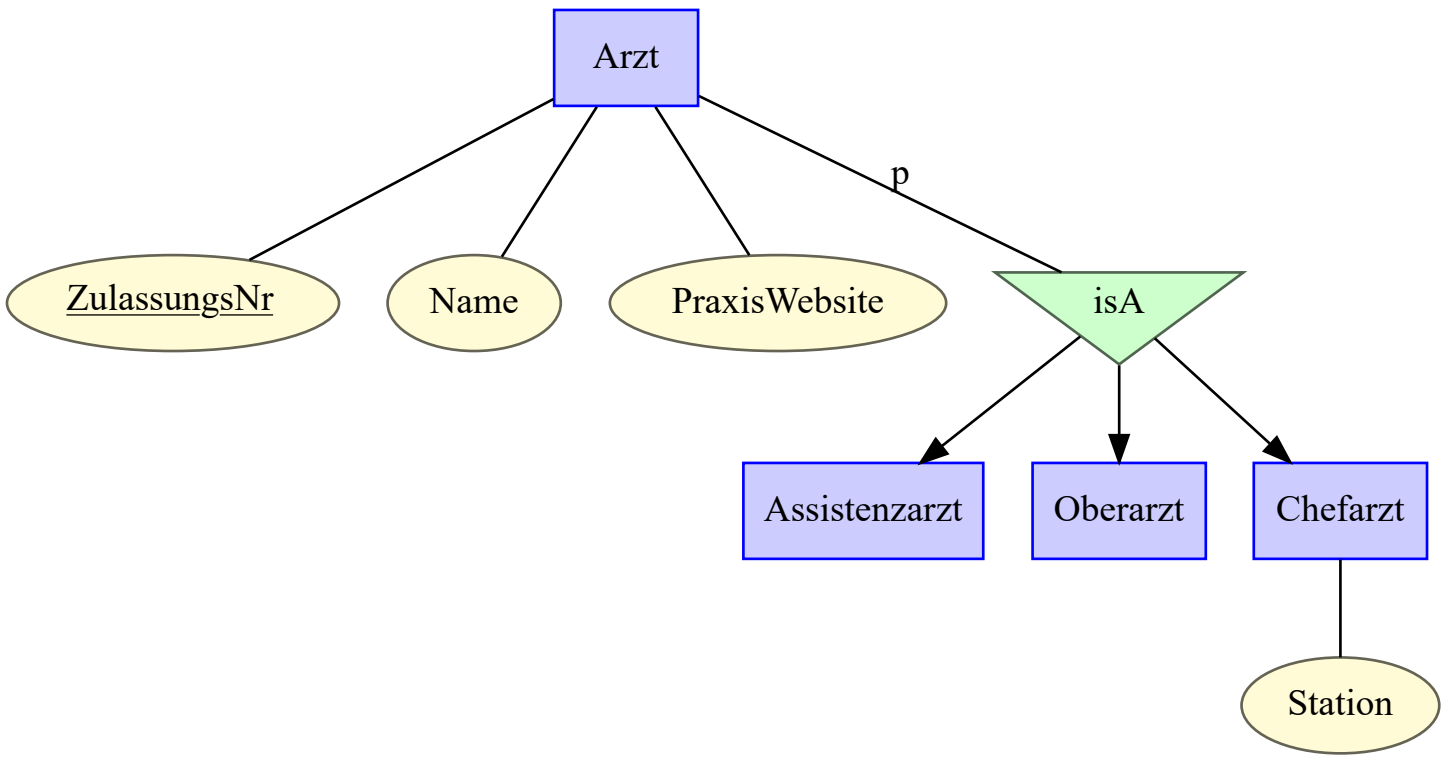
Out[38]:
```
[]
```

In [ ]:
```
#HIDDEN TEST #halber Punkt für das Entfernen von PraxisWebsite bei der Patienten Tabelle
```

In [ ]:
```
#HIDDEN TEST #halber Punkt für das Hinzufügen von PraxisWebsite bei der Arzt Tabelle
```

Kurz vor Feierabend bekommen Sie eine Email vom besagten Kunden aus b). Er bittet Sie um eine Anpassung der Datenbank, da sich noch einige Spezialisierungen der 'Arzt' Tabelle ergeben haben. Der Kunde schickt Ihnen zur Hilfe das folgende ER-Modell:

In [21]:
```
#Ein paar Vorkehrungen treffen: Import des ER-Tools. Anleitung siehe ER_Tutorial.ipynb
from assets.ER import ERDiagram
#Neues Diagramm initiieren
g = ERDiagram()



g.neuer_knoten("Arzt")
g.neues_attribut("Arzt", "ZulassungsNr", primaer=True)
g.neues_attribut("Arzt", "Name")
g.neues_attribut("Arzt", "PraxisWebsite")

g.neuer_knoten("Assistenzarzt")
g.neuer_knoten("Oberarzt")
g.neuer_knoten("Chefarzt")
g.neues_attribut("Chefarzt", "Station")

g.ist_ein("Arzt", ["Assistenzarzt", "Oberarzt", "Chefarzt"], "p", disjunkt=True)

g.display()
```



c) Da Sie die 'Arzt' Tabelle gemäß ER-Modell bereits in SQL überführt haben, fällt Ihnen auf, dass sie nur die drei Spezialisierungen übertragen und die isA-Beziehung richtig modellieren müssen. Aus Ihrer Zeit als fleißiger DBIS-Studierender wissen Sie, dass der Primärschlüssel der Spezialisierungen dem Primärschlüssel der Generalisierung entspricht und auf diesen referenziert (Nutzen Sie PRIMARY KEY REFERENCES). Die Station selber ist vom Typ varchar(255).

In [39]:
```
%%sql
CREATE TABLE Assistenzarzt (ZulassungsNr varchar(255), PRIMARY KEY (ZulassungsNr), FOREIGN KEY (ZulassungsNr) REFERENCE
S Arzt(ZulassungsNr));
CREATE TABLE Oberarzt (ZulassungsNr varchar(255), PRIMARY KEY (ZulassungsNr), FOREIGN KEY (ZulassungsNr) REFERENCES Arz
t(ZulassungsNr));
CREATE TABLE Chefarzt (ZulassungsNr varchar(255), Station varchar(255), PRIMARY KEY (ZulassungsNr), FOREIGN KEY (Zulass
ungsNr) REFERENCES Arzt(ZulassungsNr))
```

```
 * postgresql://localhost/exercise_sheet
Done.
Done.
Done.
```

Out[39]:
```
[]
```

In [ ]:
```
#HIDDEN TEST 0.5P
```

In [ ]:
```
#HIDDEN TEST 0.5P
```

In [ ]:
```
#HIDDEN TEST 0.5P
```

Da Sie nun alle Aufgaben erledigt haben, schickt Ihr Chef Sie in den wohlverdienten Feierabend...