Praktikum Datenbanksysteme

Praktikum Datenbanksysteme

Praktikum Datenbanksysteme

Hochschule Aalen

Fakultät Elektronik und Informatik

Praktikumsbericht

WS 2024/2025

Betreut von: Dr. Gregor Grambow

Erstellt von: Jonas Thiele und Maximilian Luther

Matrikelnummer: 88490, 3005364

Inhalt

Abbildungsverzeichnis

1 Entwurf des ER-Modells

1.1 Diagramme

Aus anderer Vorlesung bekanntes Klassendiagramm (auch als Anhang):

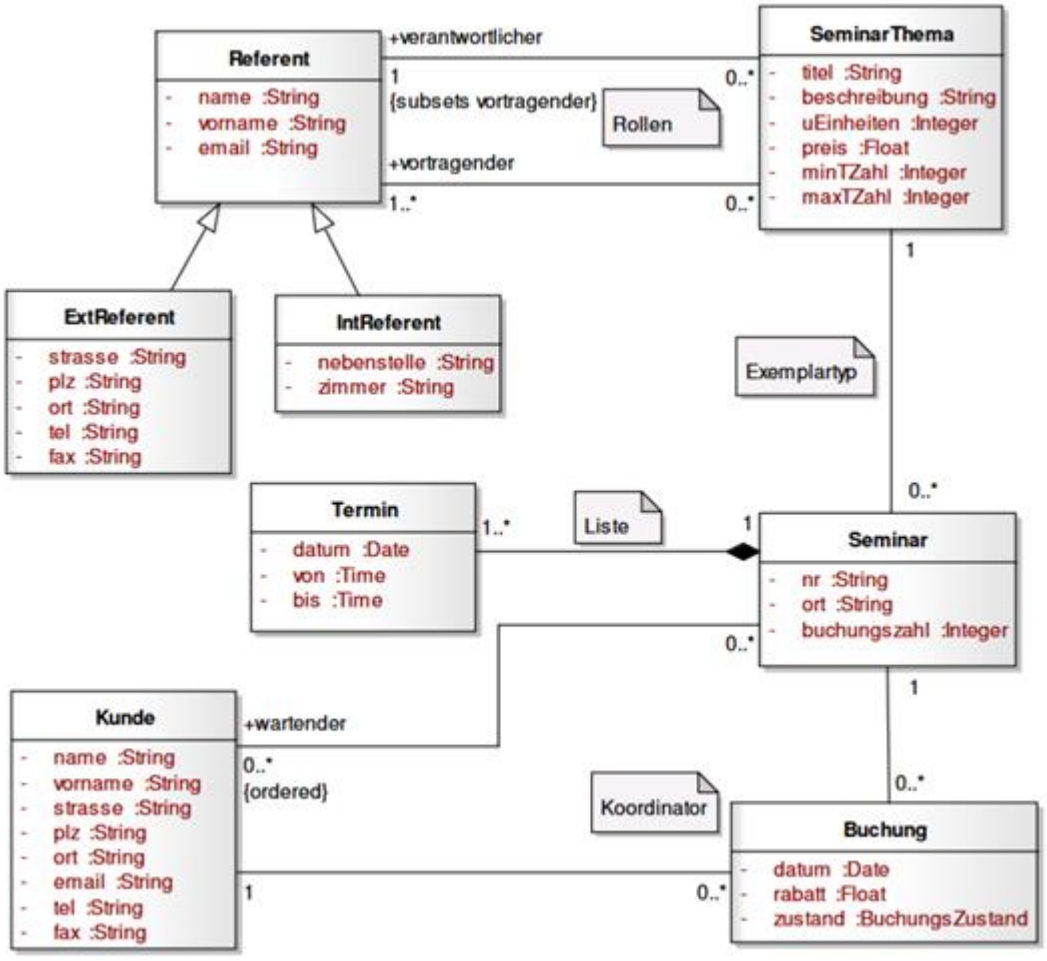

Abbildung 1: Klassendiagramm

Aus Klassendiagramm erstelltes ER-Modell (auch als Anhang):

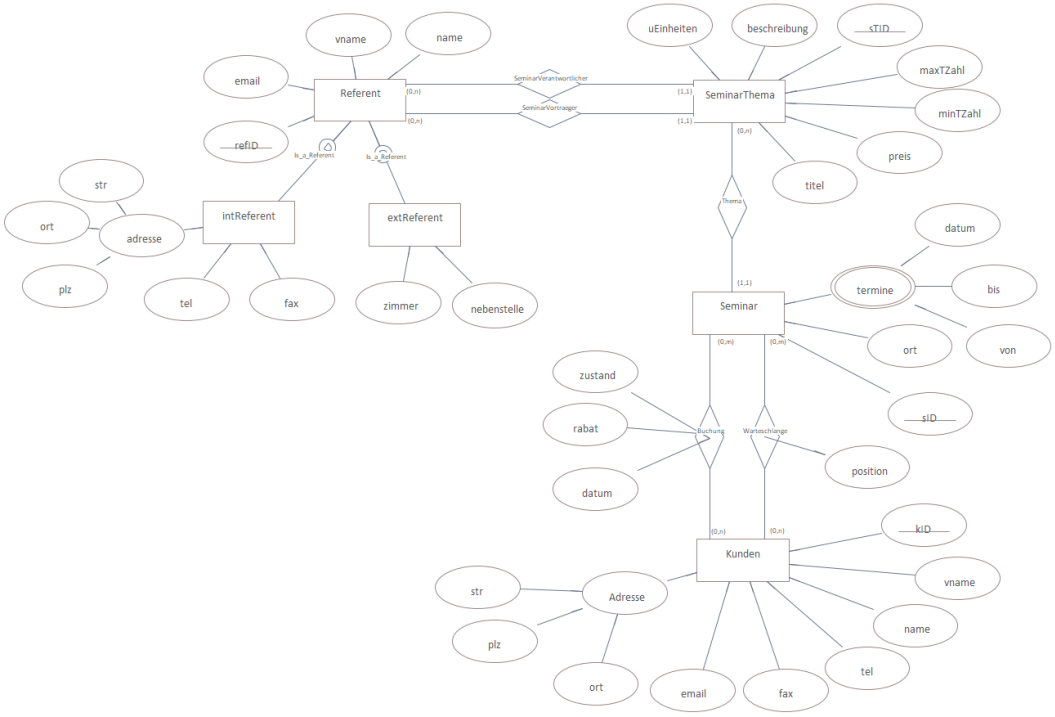

Abbildung 2: Entwurf des Entity-Relationship-Modells

1.2 Gegenüberstellung zum objektorientierten Modell

Entitäten und Attribute:

Die Entitäten wurden im Wesentlichen nach den Klassen des Klassendiagramms modelliert. Die einzigen Ausnahmen hierbei sind, dass die Koordinator-Klasse *Buchung* zu einer Beziehung und die Klasse *Termin* zu einem mehrwertigen Attribut überführt worden sind.

Die Attribute wurden inkl. ihrer Datentypen übernommen. Speziell definierte Datentypen (bspw. *Adresse*) wurden als zusammengesetzte Attribute dargestellt.

Beziehungen und Kardinalitäten:

Die Beziehungen/Assoziationen und Kardinalitäten/Multiplizitäten wurden auch hier kaum verändert. Die Vererbung mit wechselnden Rollen aus dem Klassendiagramm wurde als zwei einfache Vererbungsbeziehungen *Is_a_Referent* dargestellt. Die Buchungs-Assoziation über die Koordinator-Klasse *Buchung* wurde, wie schon erwähnt, in eine Beziehung gefasst. Die Kardinalitäten wurden somit auch im Wesentlichen übernommen und in die (min, max)-Notation umgewandelt.

1.3 Entscheidungen / Vorüberlegungen

Es wurde entschieden, aus dem mehrfach vorkommenden Attribut *Adresse* keine eigene Entität zu erstellen, da die Daten dieser Entität ausschließlich in ihrem jeweiligen Kontext verwendet werden.

Ebenfalls wurde aus der Klasse *Termin* keine Entität erstellt, da diese nur mit der Entität *Seminar* eine Beziehung hätte und somit auch als mehrwertiges Attribut darstellbar ist.

Die Koordinator-Klasse *Buchung* wurde nicht als Entität realisiert, sondern als Beziehung, da die wesentliche Aufgabe darin besteht, einen Kunden mit einem Seminar zu verknüpfen.

1.4 Ausführliche Beschreibung des ER-Modells

Mit den Vorüberlegungen zur Überführung des objektorientierten Modells und diversen Umformungen, entstanden folgende *Entitäten* und *Beziehungen.*

1.4.1 Entitäten und ihre Attribute

Referent = ({refID: Zahl, eMail: ZK, name: ZK, vName: ZK})

extReferent = ({fax: ZK, tel: ZK, adresse: (ort: ZK, plz: ZK, str: ZK)}) IS-A Referent

intRefernt = ({nebenstelle: ZK, zimmer: ZK}) IS-A Referent

SeminarThema = ({sTID: Zahl, beschreibung: ZK, maxTZahl: Zahl, minTZahl: Zahl, preis: FKZ, titel: ZK, uEinheiten: Zahl})

Seminar = ({sID: ZK, ort: ZK, termine: (von: Zeit, bis: Zeit, datum: Datum)*})

Kunden = ({kID: Zahl, eMail: ZK, fax: ZK, name: ZK, vName: ZK, tel: ZK, adresse: (ort: ZK, plz: ZK, str: ZK)})

1.4.2 Beziehungen

Buchung = (Seminar x Kunde, {datum: Datum, rabatt: FKZ, zustand: {offen, berechnet, bezahlt, storniert}})

SeminarVerantwortlicher = (Referent x SeminarThema)

SeminarVortraeger = (Referent x SeminarThema)

Thema = (Seminar x SeminarThema)

Is_a_Referent = (Referent x extReferent)

Is_a_Referent = (Referent x intReferent)

Warteschlange = (Seminar x Kunde, {position: Zahl})

1.4.3 Bedeutung der Beziehungstypen

1. Eine konkrete Beziehung des Typs *Buchung* besteht zwischen einem Entity e_1 des Typs *Seminar* und einem Entity e_2 des Typs *Aufsatz* genau dann, wenn *Kunde* e_1 am *Datum* das *Seminar* e_2 bucht und dabei einen *rabatt* (inkl. 0.0) und der *Zustand* Status hat.
2. Eine konkrete Beziehung des Typs *SeminarVerantwortlicher* besteht zwischen einem Entity e_1 des Typs *Referent* und einem Entity e_2 des Typs *SeminarThema* genau dann, wenn *Referent* e_1 für das *SeminarThema* e_2 verantwortlich und eingetragen ist.
3. Eine konkrete Beziehung des Typs *SeminarVortraeger* besteht zwischen einem Entity e_1 des Typs *Referent* und einem Entity e_2 des Typs *SeminarThema* genau dann, wenn *Referent* e_1 die vortragende Person des *SeminarThema* e_2 ist.
4. Eine konkrete Beziehung des Typs *Thema* besteht zwischen einem Entity e_1 des Typs *Seminar* und einem Entity e_2 des Typs *SeminarThema* genau dann, wenn *SeminarThema* e_1 das Thema für das *Seminar* e_2 ist.
5. Eine konkrete Beziehung des Typs *Is_a_Referent* besteht zwischen einem Entity e_1 des Typs *Referent* und einem Entity e_2 des Typs *extReferent* genau dann, wenn *extReferent* e_1 auch ein *intReferent* e_2 ist.
6. Eine konkrete Beziehung des Typs *Is_a_Referent* besteht zwischen einem Entity e_1 des Typs *Referent* und einem Entity e_2 des Typs *intReferent* genau dann, wenn *intReferent* e_1 auch ein *intReferent* e_2 ist.

7. Eine konkrete Beziehung des Typs *Warteschlange* besteht zwischen einem Entity e_1 des Typs *Seminar* und einem Entity e_2 des Typs *Kunde* genau dann, wenn *Kunde* e_1 als Wartende/-r für *Seminar* e_2 mit einer Warte-*position* eingetragen ist.

1.4.4 Kardinalitäten

Buchung: Kunde (0, n) bucht (0, m) Seminar.

(0, n)	(0, m)
min 0: Seminare müssen (bspw. Beim Anlegen) keine Buchungen haben	min 0: Kunden müssen kein Seminar besuchen
max: Ein Seminar kann von mehreren Kunden gebucht werden	max: Ein Kunde kann mehrere Seminare buchen

SeminarVerantwortlicher: Referent (0, n) ist Seminarverantwortlicher für (1, 1) Seminarthema.

(0, n)	(1, 1)
min 0: Ein Referent muss kein Seminarverantwortlicher sein	min/max 1: Seminarthemen müssen genau einen Seminarverantwortlichen haben
max: Ein Referent kann in verschiedenen Seminarthemen Seminarverantwortlicher sein	

SeminarVortraeger: Referent (0, n) ist Seminarvorträger für (1, m) Seminarthema.

(0, n)	(1, m)
min 0: Ein Referent muss kein Seminarvorträger sein	min 1: Ein Seminar muss einen Seminarvorträger haben.
max: Ein Referent kann in verschiedenen Seminarthemen Seminarvorträger sein	max: Ein Seminar kann mehrere Seminarvorträger haben.

Thema: Seminar (1, 1) hat als Thema (0, n) Seminarthema.

(1, 1)	(0, n)
min/max 1: Seminare müssen genau ein Seminarthema haben	min 0: Es muss kein Seminar zu einem Seminarthema geben (bspw. beim Anlegen)
	max: Es kann mehrere Seminare zu einem Seminarthema geben

Warteschlange: Kunde (0, n) steht auf Warteschlange für (0, m) Seminar

(0, n)	(0, m)
min 0: Ein Kunde muss nicht auf einer Warteschlange für ein Seminar stehen	min 0: Ein Seminar muss keine Kunden auf der Warteschlange stehen haben
max: Ein Kunde kann auf mehreren Warteschlangen für verschiedene Seminare stehen	max: Ein Seminar kann mehrere Kunden auf der Warteschlange stehen haben

2 Entwurf Relationales Modell

Um das Relationale Modell aus dem oben beschriebenen ER-Modell abzuleiten, wurde sich an das in der Vorlesung besprochene Vorgehen gehalten. Dabei wurden manche *Entitäten* in andere integriert. Daraus ist folgendes Modell entstanden.

2.1 Relationenschemata zu Entity-Typen

Referent = ({refID, eMail, name, vName}, {})

ExtReferent = ({refID, fax, tel, ort, plz, str}, {})

IntRefernt = ({refID, nebenstelle, zimmer}, {})

SeminarThema = ({sTID, beschreibung, maxTZahl, minTZahl, preis, titel, uEinheiten, verantwortlicher}, {})

Seminar = ({sID, ort, sTID}, {})

Termin = ({tID, von, bis, datum, sID}, {})

Kunden = ({kID, eMail, fax, name, vName, tel, ort, plz, str}, {})

2.2 Relationenschemata zu Beziehungs-Typen

Buchung = ({datum, rabatt, zustand, sID, kID}, {})

Warteschlange = ({position, sID, kID}, {})

SeminarVortraeger = ({refID, sTID}, {})

SeminarVerantwortlicher: Ist in Seminarthema enthalten, da 1:n Beziehung.

Thema: Ist in Seminar enthalten, da 1:n Beziehung.

Is_a_Referent: Ist in den spezifischen Referent-Typen enthalten.

2.3 Interrelationale Abhängigkeiten

$$extReferent|_{(refID)} \subseteq Referent|_{(refID)}$$

$$intReferent|_{(refID)} \subseteq Referent|_{(refID)}$$

$$SeminarVortreager|_{(refID)} \subseteq Referent|_{(refID)}$$

$$SeminarVortreager|_{(sTID)} \subseteq SeminarThema|_{(sTID)}$$

$$SeminarThema|_{(verantwortlicher)} \subseteq Referent|_{(refID)}$$

$$Seminar|_{(sTID)} \subseteq SeminarThema|_{(sTID)}$$

$$Termin|_{(sID)} \subseteq Seminar|_{(sID)}$$

$$Buchung|_{(sID)} \subseteq Seminar|_{(sID)}$$

$$Buchung|_{(kID)} \subseteq Kunde|_{(kID)}$$

$$Warteschlange|_{(sID)} \subseteq Seminar|_{(sID)}$$

$$Warteschlange|_{(kID)} \subseteq Kunde|_{(kID)}$$

3 SQL-Statements

Bei der Erstellung der Tabellen wurde darauf geachtet, dass es bei einzutragenden Personen immer eine Möglichkeit gibt, über die Sie konstatierbar ist. Daher sind viele Personenbezogene Spalten `NOT NULL`.

An Stellen, an denen es Sinn ergibt, wurde auch `ON DELETE CASCADE` in den `FOREIGN KEY` gesetzt, um nicht mehr gebrauchte Referenzen zu löschen. An den Stellen, an denen dies nicht gesetzt ist, war aus der Aufgabe nicht eindeutig, was beim Löschen passieren soll, weshalb sich für `NO ACTION` entschieden wurde.

3.1 Abfragen

Die folgenden Abschnitte enthalten verschiedene Beispiel-SQL-Abfragen, um relevante Daten aus der Datenbank strukturiert auszuwerten.

3.1.1 Anzahl aller Referenten

```
SELECT COUNT(*) as total, COUNT(fax) as ext, COUNT(*)-COUNT(fax) as int
    FROM g03_Referent as a
    LEFT JOIN g03_ExtReferent USING(RefID)
```

Das SQL-Statement zählt alle Referenten sowie die Anzahl interner und externer Referenten. Ein `LEFT JOIN` wird verwendet, um sicherzustellen, dass auch Referenten ohne Eintrag in der Tabelle für externe Referenten berücksichtigt werden. Die Unterscheidung zwischen internen und externen Referenten erfolgt über das Attribut *fax*, das nur bei externen Referenten vorhanden ist.

3.1.2 Seminare mit Seminarterminen

```
SELECT DISTINCT SID, COUNT(TID)
FROM g03_Seminar JOIN g03_Termin USING(SID)
GROUP BY SID
ORDER BY SID
```

Das SQL-Statement zählt für jedes Seminar (*SID*) die zugehörigen Termine (*TID*). Ein `INNER JOIN` verknüpft die Tabellen g03_Seminar und g03_Termin über die gemeinsame *SID*, und durch `GROUP BY` wird die Anzahl der Termine pro Seminar aggregiert. `DISTINCT` stellt sicher, dass jede SID nur einmal in der Ausgabe erscheint.

3.1.3 Seminare mit Teilnehmeranzahl

```
SELECT DISTINCT SID, COUNT(SID)
FROM g03_Seminar JOIN g03_Buchung USING(SID)
WHERE zustand = 'bezahlt'
GROUP BY SID
ORDER BY SID
```

Das SQL-Statement zählt für jedes Seminar (*SID*) die Anzahl der bezahlten Buchungen. Ein `INNER JOIN` verbindet g03_Seminar und g03_Buchung über *SID*, und durch die `WHERE`-Bedingung werden nur Einträge mit zustand = 'bezahlt' berücksichtigt. `GROUP BY` aggregiert die Daten nach Seminar, und `DISTINCT` stellt sicher, dass jede *SID* nur einmal erscheint.

3.1.4 Seminare mit längster Warteliste

```
SELECT SID, COUNT(KID) as groesse_warteschlange
FROM g03_Seminar JOIN g03_Warteschlange USING(SID)
GROUP BY SID
HAVING COUNT(KID) >= All(
    SELECT COUNT(KID)
    FROM g03_Warteschlange
    GROUP BY KID)
```

Das SQL-Statement ermittelt Seminare (*SID*) mit der größten Warteschlange. Ein `INNER JOIN` verbindet g03_Seminar und g03_Warteschlange über *SID*, und `GROUP BY` aggregiert die Warteschlangengrößen pro Seminar. Die `HAVING`-Bedingung filtert Seminare, deren Warteschlange größer oder gleich der maximalen Warteschlangengröße aller Seminare ist, basierend auf einem verschachtelten Subquery.

3.1.5 Teilnehmerzahl der verantworteten Seminare

```
SELECT RefID, COUNT(KID) as Verantw_Teilnehmer
FROM g03_Referent re LEFT JOIN g03_SeminarThema st ON (re.RefID =
st.verantwortlicher)
    LEFT JOIN (g03_Seminar se JOIN g03_Buchung be ON se.SID = be.SID AND
zustand = 'bezahlt') USING(STID)
GROUP BY RefID
```

Das SQL-Statement zählt die Teilnehmer (*KID*) pro Referent (*RefID*) aus bezahlten Buchungen. Ein `INNER JOIN` verbindet Seminare und Buchungen, um nur bezahlte Buchungen zu berücksichtigen, während zwei `LEFT JOINs` sicherstellen, dass alle Referenten ausgegeben werden, auch wenn sie keine Seminare oder Teilnehmer haben. `GROUP BY` aggregiert die Teilnehmerzahlen pro Referent.

3.1.6 Einnahmen der Referenten

```
SELECT RefID, SUM((1 - be.rabatt/100) * st.preis) as money
FROM g03_Referent re LEFT JOIN g03_SeminarThema st ON (re.RefID =
st.verantwortlicher)
    LEFT JOIN (g03_Seminar se JOIN g03_Buchung be ON se.SID = be.SID AND
zustand = 'bezahlt') USING(STID)
GROUP BY RefID
```

Das SQL-Statement berechnet den Gesamtumsatz (*money*) pro Referent (*RefID*) basierend auf den Preisen der Seminare und gewährten Rabatten aus bezahlten Buchungen. Ein `INNER JOIN` verbindet Seminare und Buchungen, um nur bezahlte Buchungen einzubeziehen, während zwei `LEFT JOINs` sicherstellen, dass alle Referenten berücksichtigt werden, auch ohne Seminare oder Buchungen. `GROUP BY` aggregiert die Umsätze pro Referent.

4 Anhänge

4.1 Entity-Relationship-Modell

Entity-Relationship-Modell in separatem Dokument:

g03_ERM.png

4.2 Klassendiagramm

Klassendiagramm in separatem Dokument:

g03_KD.png

4.3 SQL-Statements

SQL-Statements in separatem Dokument:

g03_SQL.txt